second inputs of Times 1 And Times 2 Multiply Shifter (MULTSHFT12) 20366 and Times 4 And Times 8 Multiply Shifter (MULTSHFT48) 20368. Outputs of MULTSHFT12 and MULTSHFT8 are connected, respectively, to first and second inputs of First Multiplier Arithmetic and Logic Unit (MULTALU1) 20370. MULTALU1 20370's output is connected to input of Multiplier Working Register (MWR) 20372. Output of MWR 20372 is connected to a first input of Second Multiplier Arithmetic and Logic Unit (MULTALU2) 20374. A second input of MULTALU2 20374 is connected from output of RFR 20336. Output of MULTALU2 is connected to a second input of FRS 20362. As described above, first input of FRS 20362 is connected from output of NIBSHF 20368. Output of FRS 20362 is connected to input of RFR 20336.

As described above, output of RFR 20336 is connected to second input of MULTALU2 20374, to first input of MULTRF 20350, to first input of MULTRM 20334, and to second input of FROM 20324. Output of RFR 20336 is also connected to input of Leading Zero Detector (LZD) 20376 of MULTCNTL 20318, and to inputs of Exception Logic (ECPT) 20378, CONSIZE 20352, and TSTINT 20320.

4. Exponent Logic 20316

Referring to EXP 20316, as previously described EXP 20316 performs certain operations with respect to exponent fields of single and double precision floating point number in EU 10122 floating point operations. EXP 20316 includes a second portion of EU 10122's general register file, shown herein as Exponent Register File (EXPRF) 20380. Although indicated as individual register files, MULTRF 20350 and EXPRF 20380 comprise, as in GRF 10354, a unitary register file structure with common, parallel addressing of corresponding registers therein.

Output of EXPRF 20380 is connected to a second input of INSELA 20330. A first input of EXPRF 20380 is connected from output of EXRM 20332. As previously described, a first input of EXRM 20332 is connected from second output of OPB 20322 through EXPQ Bus 20325. A second input of EXRM 20332 is connected from output Scale Register (SCALER) 20338. A second input of EXPRF 20380 is connected from output of Sign Logic (SIGN) 20382. Input of SIGN 20382 is connected from second output of SCALER 20338.

INSELA 20330, INSELB 20348, Exponent ALU (EXPALU) 20384 and SCALER 20338 comprise EXP 20316's arithmetic circuitry for manipulating exponent fields of floating point numbers. INSELA 20330 and INSELB 20348 select, respectively, first and second inputs to EXPALU 20384. As previously described, a first input of INSELA 20330 is connected from second output of OPB 20322 through EXPQ Bus 20325. Second input of INSELA 20330 is connected from output of EXPRF 20380. Output of INSELA 20330 is connected to first input of EXPALU 20384. First input of INSELB 20348 is, as previously described, connected from a second output of mCRD 20346. Second input of INSELB 20348 is connected from output of OPB 20322 through EXPQ Bus 20325. Third input of INSELB 20348 is connected from output of SCALER 20338 and fourth input of INSELB 20348 is connected from output of LZD 20376. Output of INSELB 20348 is connected to second input of EXPALU 20348. Output of EXPALU 20348 is connected to input of SCALER 20338.

As previously described, second output of SCALER 20338 is connected with input of SIGN 20382 and first output is connected to second input of EXRM 20332 and to third input of INSELB 20348. First output of SCALER 20338 is also connected to EXPQ Bus 20325, to first input of EXOM 20326, and to a second input of MULTCNT 20364.

5. Multiplier Control 20318

As previously described, MULTCNTL 20318 provides certain control signals and information for controlling and coordinating operation of EXP 20316 and MULT 20314 in performing arithmetic operations on floating point numbers. MULTCNTL 20318 includes LZD 20376 and MULTCNT 20364. Input of LZD 20376 is connected from output of RFR 20336 through FR Bus 20337. Output of LZD 20376 are connected to a second input of MULTCNT 20364 and to fourth input of INSELB 20348. A second input of MULTCNT 20364 is connected from output of SCALER 20338. As previously described, control output of MULTCNT 20364 is connected to control inputs of NIBSHF 20358.

6. Test and Interface Logic 20320

Finally, TSTINT 20320 includes ECPT 20378, CONSIZE 20352, and Testing Condition Logic (TSTCON) 20386. Input of ECPT 20378 and first input of CONSIZE 20352 are connected from output of RFR 20336 through FR Bus 20337. A second input of CONSIZE 20352 is connected from LENGTH Bus 20226. An output of CONSIZE 20352 is connected, together with other inputs from EU 10122 (not shown for clarity of presentation) to TSTCON 20386. Output of TSTCON 20386 (not shown for clarity of presentation) are connected to NAG 20340. TSTCON 20386 and ECPT 20378 have outputs to and inputs from FU 10120's FUINT 20298.

Having described the overall structure of EU 10122 above, operation of EU 10122 will be described next below with aid of further diagrams which will be introduced as required. Finally, operation of TSTINT 20320 will be described, including a description of the detailed control signal interface between EU 10122 and FU 10120 through TSTINT 20320 and FUINT 20298. In addition to defining the interface between EU 10122 and FU 10120, certain features of EU 10122 operation will be described wherein those operations are executed

in cooperation with MEM 10112 and FU 10120. For example, EU 10122's Stack Mechanisms, comprising in part portions of MULTRF 20350 and EXPRF 20380, resides partly in MEM 10112 so that operation of EU 10122's Stack Mechanisms requires cooperative operations by EU 10122, MEM 10112 and FU 10120.

b. Execute Unit 10122 Operation (Fig. 255)

1. Execute Unit Control Logic 20310 (Fig. 255)

Referring to Fig. 255, a more detailed block diagram of EUCL 20310 is shown. As described above, EUCL 20310 receives EU 10122 Dispatch Pointers through EUDIS Bus 20206 from EUSDT 20266 and FUCTL 20214. EU 10122 Dispatch Pointers select certain EU 10122 microinstruction sequences for executing EU 10122 arithmetic operations as required to execute user's programs, that is SOPs, and to assist in handling JP 10114 Events. As described above, major elements of EUCL 20310 include COMQ 20342, EUSITT 20344, mCRD 20346, and NAG 20340.

a.a. Command Queue 20342

Inputs of COMQ 20342 are connected from EUDIS Bus 20206 to receive and store EU 10122 Dispatch pointers provided from EUSDT 20266. Each such EU 10122 Dispatch Pointer is comprised of two information fields. A first information field contains a 10 bit starting address of a corresponding sequence of microinstructions residing in EUSITT 20344. Second field of each EU 10122 Dispatch Pointer is a 6 bit field containing certain control information, such as information identifying data format of corresponding operands to be operated upon. In this case unit dispatch pointer control field bits specify whether operands to be operated upon comprise signed or unsigned integer, packed or unpacked decimal, or single or double precision floating point numbers.

COMQ 20342 is comprised of two one word wide by two word deep register files. A first of these register fields is comprised of SOP Command Queue Control Store (CQCS) 25510 and SOP Command Queue Address Store (CQAS) 25512. Together, CQCS 25510 and CQAS 25512 comprise a one word wide by two word deep register file for receiving and storing EU 10122 Dispatch Pointers corresponding to SOPs, that is Dispatch Pointers for initiating EU 10122 operations directly concerned with executing a user's program. Address fields of these SOPs are received in CQAS 25512, while control fields are received and stored in CQCS 25510. COMQ 20342 is thereby capable of receiving and storing up to two sequential EU 10122 Dispatch Pointers corresponding to user program SOPs These SOP derived Dispatch Pointers are executed in the order received from FU 10120. EU 10122 is thereby capable of receiving and storing one currently executing SOP Dispatch Pointer and one pending SOP Dispatch Pointer. Further SOP Dispatch Pointers may be read into COMQ 20342 as previous SOPs are executed.

b.b. Command Queue Event Control Store 25514 and Command Queue Event Address Control Store 25516

Command Queue Event Control Store (CQCE) 25514 and command Queue Event Address Control Store (CQAE) 25516 are similar in function and operation to, respectively, CQCS 25510 ad CQAS 25512. CQCE 25514 and CQAE 25516 receive and store, however, EU 10122 Dispatch Pointers initiating EU 10122 operations requested by FU 10120 as required to handle JP 10114 Events. Again, CQCE 25514 and CQAE 25516 comprise a one word wide by two word deep register file. CQAE 25516 receives and stores address fields of Event Dispatch Pointers, while CQCE 25514 receives and stores corresponding control fields of Event Dispatch Pointers. Again, COMQ 20342 is capable of receiving and storing up to two sequential Event Dispatch Pointers at a time.

As indicated in Fig. 255, outputs of CQAS 25512 and CQAE 25516, that is address fields of EU 10122 Dispatch Pointers are provided as inputs to Select Case Multiplexer (SCASE) 25518 and Starting Address Select Multiplexer (SAS) 25520 and NAG 20340, which will be described further below. Control field outputs of CQCS 25510 and CQCE 25514 are provided as inputs to OPB 20322, described further below.

c.c. Execute Unit S-Interpreter Table 20344

Referring to EUSITT 20344, as described above EUSITT 20344 is a memory for storing sequences of microinstructions for controlling operation of EU 10122 in response to EU 10122 Dispatch Pointers received from FU 10120. These microinstruction sequences may, in general, direct operation of EU 10122 to execute arithmetic operations in response to SOPs of user's programs, or aid direct execution of EU 10122 operations required to service JP 10114 Events. EUSITT 20344 may be, for example, a 60 bit wide by 1,280 word long memory structured as pages of 128 words per page. A portion of EUSITT 20344's pages may be contained in Read Only Memory, for example for storing sequence of microinstructions for handling JP 10114 Events. Remaining portions of EUSITT 20344 may be constructed of Random Access Memory, for example for storing sequences of microinstructions for executing EU 10122 operations in response to user program SOPs. This structure allows EU 10122 microinstruction sequences concerned with operation of JP 10114's internal mechanisms, for example handling of JP 10114 Events, to be effectively permanently

stored in EUSITT 20344. That portion of EUSITT 20344 constructed of Random Access Memory may be used to store sequences of microinstructions for executing SOPs. These Random Access Memories may be used as writable control store to allow sequences of microinstructions for executing SOPs of one or more S-Languages currently being utilized by CS 10110 to be written into EUSITT 20344 from MEM 10112 as required.

As previously described, EUSITT 20344's second input is a Data (DATA) input connected from JPD Bus 10142. EUSITT 20344's data input is utilized to write sequences of microinstructions into EUSITT 20344 from MEM 10112 through JPD Bus 10142. EUSITT 20344's first input is an address (ADR) input connected from output of Address Driver (ADRD) 25522 and NAG 20340. Address inputs provided by ADRD 25522 select word locations within EUSITT 20344 for writing of microinstructions into EUSITT 20344, or for reading of microinstructions from EUSITT 20344 to mCRD 20346 to control operation of EU 10122. Generation of these address inputs to EUSITT 20344 by NAG 20340 will be described further below.

d.d. Microcode Control Decode Register 20346

Output of EUSITT 20344 is connected to input of mCRD 20346. As previously described, mCRD 20346 is a register for receiving microinstructions from EUSITT 20344, and decoding logic for decoding those microinstructions and providing corresponding control signals to EU 10122. As indicated in Fig. 255, Diagnostic processor Micro-Program Register (DPmR) 25524 is a 60 bit register connected in parallel with output of EUSITT 20344 to input of mCRD 20346. DPmR 25524 may be loaded with 60 bit microinstructions by DP 10118. Diagnostic microinstructions may thereby be provided directly to input of mCRD 20346 to provide direct microinstruction by microinstruction control of EU 10122.

Outputs of mCRD 20346 are provided, in general, to all portions of EU 10122 to control detailed operations of EU 10122. Certain outputs of mCRD 20346 are connected to inputs of Next Address Source Select Multiplexer (NASS) 25526 and Long Branch Page Address Gate (LBPAG) 25528 and NAG 20340. As will be described further below, these outputs of mCRD 23046 are used in generating address inputs to EUSITT 20344 when particular microinstructions sequences call for Jumps or Long Branches to other microinstruction sequences. Outputs of mCRD 20346 are also connected in parallel to inputs of Execution Unit Micro-Instruction Parity Check Logic (EUmIPC) 25530. EUmIPC 25530 checks parity of all microinstruction outputs of mCRD 20346 to detected errors in mCRD 20346's outputs.

e.e. Next Address Generator 20340

As described above, read and write addresses to EUSITT 20344 provided by NAG 20340 through ADRD 25522. Address inputs to ADRD 25522 are provided from either NASS 25526 or Diagnostic Processor Address Register (DPAR) 25532. In normal operation, address inputs to EUSITT 20344 are provided from NASS 25526 as will be described momentarily. DP 10118, however, may load EUSITT 20344 addresses into DPAR 25532. These addresses may then be read from DPAR 25532 through ADRD 25522 to individually select address locations within EUSITT 20344. DPAR 25532 may be utilized, in particular, to provide addresses to allow stepping through of EU 10122 microinstruction sequences microinstruction by microinstruction.

As described above, NASS 25526 is a multiplexer having inputs from three NAG 20340 address sources. NASS 25526's first address input is from Jump (JMP) output of mCRD 20346 and LBPAG 25528. These address inputs are utilized, in part, when a current microinstruction calls for a Jump or Long Branch to another microinstruction or microinstruction sequence. Second address source is provided from SAS 25520 and, in general, is comprised of starting addresses of microinstruction sequences. SAS 25520 is a multiplexer having a first input from COAS 25512 and COAE 25516, that is starting addresses of microinstruction sequences corresponding to SOPs or for servicing JP 10114 Events. A second SAS 25520 input is provided from Sub-routine Return Address Stack (SUBRA) 25534. In general, and as will be described further below, SUBRA 25534 operates as a stack mechanism for storing current microinstruction addresses of interrupted microinstruction sequences. These stored addresses may subsequently be utilized to resume execution of those interrupted microinstruction sequences. Third address source to NASS 25526 is provided from Sequential and Case Address Generator (SCAG) 25536. In general, SCAG 25536 generates address to select sequential microinstructions within particular microinstruction sequences. SCAG 25536 also generates microinstruction address for microinstruction Case operations. As indicated in Fig. 255, outputs of SCAG 25536 and of SAS 25520 are bused together to comprise a single NASS 25526 input. Selection between outputs of SCAG 25536 and SAS 25520 are provided by control inputs (not shown for clarity of presentation) to SCAG 25536 and SAS 25520. Selection between NASS 25526's address inputs is controlled by Next Address Source Select Control Logic (NASSC) 25538, which provides control inputs to NASS 25526. NASSC 25538 is effectively a multiplexer receiving control inputs from TSTCON 20386 and TSTINT 20320. As will be described further below, TSTCON 20386 monitors certain operating conditions or states within EU 10122 and provides corresponding inputs to NASSC 25538. NASSC 25538 effectively decodes these control inputs from TSTCON 20386 to provide selection control input to NASS 25526.

Having described overall structure and operation of NAG 20340, operation of NAG 20340 will be

described in further detail next below.

Referring first to NASS 25526's address inputs provided from JMP output of mCRD 20346 and LBPAG 25528, this address source is provided to allow selection of a next microinstruction by a current microinstruction. JMP output of mCRD 20346 allows a current microinstruction to direct a Jump to another microinstruction within the same page of EUSITT 20344. NASS 25526's input through LBPAG 25528 is provided from another portion of mCRD 20346's output specifying pages within EUSITT 20344. This input through LBPAG 25528 allows execution of Long Branch operations, that is jumps from a microinstruction in one page of EUSITT 20344 to a microinstruction in another page. In addition, NASS 25526's input from JMP output of mCRD 20346 and through LBPAG 25528 is utilized to execute an Idle, or Standby, routine when EU 10122 is not currently executing a microinstruction sequence requested by FU 10120. In this case, Idle routine directs TSTCON 20386 to monitor EU 10122 Dispatch Pointer inputs to EU 10122 from FU 10120. If no EU 10122 Dispatch Pointers are present in COMQ 20342, or none are pending, TSTCON 20386 will direct NASSC 25538 to provide control inputs to NASS 25526 to select NASS 25526's input from mCRD 20346 and LBPAG 25528. Idle routine will continually test for EU 10122 Dispatch pointer inputs until such a Dispatch Pointer is received into COMQ 20342. At this time, TSTCON 20386 will detect the pending Dispatch Pointer and direct NASS 25538 to provide control outputs to NASS 25526 to select NASS 25526's input from, in general, SAS 25520. TSTCOND 20386 and NASSC 25538 will also direct NASS 25526 to select inputs from SAS 25520 upon return from a called microinstruction to a previously interrupted microinstruction sequence.

As described above, SAS 25520 receives starting addresses from COMQ 20342 and from SUBRA 25534. SAS 25520 will select the output of CQAS 25512 or of CQAE 25516 as the input to NASS 25526 when a new microinstruction sequence is to be initiated to execute a user's program SOP or to service a JP 10114 Event. SAS 25520 will select an address output of SUBRA 25534 upon return from a called sub-routine to a previously executing but interrupted sub-routine. SUBRA 25534, as described above, is effectively a stack mechanism for storing addresses of currently executing microinstructions when those microinstruction sequences are interrupted. SUBRA 25534 is an 11 bit wide by 8 word deep register with certain registers dedicated for use in stacking Event Handling microinstruction sequences. Other portions of SUBRA 25534 are utilized for stacking of microinstruction sequences for executing SOPs, that is for stacking microinstruction sequences wherein a first microinstruction sequence calls for a second microinstruction sequence. SUBRA 25534 is not operated as a first-in-first out stack, but as a random access memory wherein address inputs selecting registers and SUBRA 25534 are provided by microinstruction control outputs of mCRD 20346. Operations of SUBRA 25534 as a stack mechanism is thereby controlled by the microinstruction sequences stored in EUSITT 20344. As indicated in Fig. 255, addresses of current microinstructions of interrupted microinstruction sequences are provided to data input of SUBRA 25534 from output of SCAG 25536, which will be described next below.

As described above, SCAG 25536 generates sequential addresses to select sequential microinstructions within microinstruction sequences and to generate microinstruction addresses for Case operations. SCAG 25536 includes Next Address Register (NXTR) 25540, Next Address Arithmetic and Logic Unit (NAALU) 25542, and SCASE 25518. NAALU 25542 is a 12 bit arithmetic and logic unit. A first eleven bit input of NAALU 25542 is connected from output of ADRD 25522 and is thereby current address provided to EUSITT 20344. A second four bit input to NAALU 25542 is provided from output of SCASE 25518. During sequential execution of a microinstruction sequence, output of SCASE 25518 is binary zeros and carry input of NAALU is forced to 1. Output of NAALU 25542 will thereby be and address one greater than the current microinstruction address provided to EUSITT 20344 and will thereby be the address of the next sequential microinstruction. As indicated in Fig. 255, SCASE 25518 receives an input from output of SCALER 20338. This input is utilized during Case operations and allows a data sensitive number to be selected as SCASE 25518's output into second input of NAALU 25542. SCASE 25518's input from SCALER 20338 thereby allows NAG 20340 to perform microinstruction Case operations wherein Case Values are determined by the contents of SCALER 20338.

Next address outputs of NAALU 25542 are loaded into NXTR 25540, which is comprised of tri-state output registers. Next address outputs of NXTR 25540 are connected, in common with outputs of SAS 25520, to second input of NASS 25526 as described above. During normal execution of microinstruction sequences, therefore, SCAG 25536 will, through NASS 25526 and ADRD 25522, select sequential microinstructions from EUSITT 20344. SCAG 25536 may also, as just described, provide next microinstruction addresses in microinstruction Case operations.

In summary, NAG 20340 is capable of performing all usual microinstruction sequence addressing operations. For example, NAG 20340 allows selection of next microinstructions by current microinstructions, either for Jump operations or Long Branch operations, through NASS 25526's input from mCRD 20346's JMP or through LBPAG 25528. NAG 20340 may provide microinstruction sequence starting addresses through COMQ 20342 and SAS 25520, or may provide return addresses to interrupted and stacked microinstruction sequences through SUBRA 25534 and SAS 25520. NAG 20340 may sequentially address microinstructions of a particular microinstruction sequence through operation of SCAG 25536, or may perform microinstruction Case operations through SCAG 25536.

2. Operand Buffer 20322

Having described structure and operation of EUCL 20310, structure and operation of OPB 20322 will be described next below. As previously described, OPB 20322 receives operands, that is data, from MEM 10112 and FU 10120 through MOD Bus 10144 and JPD Bus 10142. OPB 20322 may then perform certain operand format translations to provide data to MULT 20314 and EXP 20316 in the formats most efficiently utilized by MULT 20314 and EXP 20316. As previously described, EU 10122 may perform arithmetic operations on integer, packed and unpacked decimal, and single or double precision floating point numbers.

In summary, therefore, OPB 20322 is capable of accepting integer, single and double precision floating point, and packed and unpacked decimal operands from MEM 10112 and FU 10120 and providing appropriate fields of those operands to MULT 20314 and EXP 20316 in the formats most efficiently utilized by MULT 20314 and EXP 20316. In doing so, OPB 20322 extracts exponent and mantissa fields from single and double precision floating point operands to provide exponent and mantissa fields of these operands to, respectively, EXP 20316 and MULT 20314, and also unpacks, or converts, unpacked decimal operands to packed decimal operands most efficiently utilized by MULT 20314.

Having described structure and operation of OPB 20322, structure and operation of MULT 20314 will be described next below.

3. Multiplier 20314 (Figs. 257, 258)

MULT 20314, as previously described, performs addition, subtraction, multiplication, and division operations on mantissa fields of single and double precision floating point operands, integer operands, and decimal operands. As described above with reference to OPB 20322, OPB 20322 converts unpacked decimal operands to packed decimal operands to be operated upon by MULT 20314. MULT 20314 is thereby effectively capable of performing all arithmetic operations on unpacked decimal operands.

a.a. Multiplier 20314 Data Paths and Memory (Fig. 257)

Referring to Fig. 257, a more detailed block diagram of MULT 20314's data paths and memory is shown. As previously described, major elements of MULT 20314 include memory elements comprised of MULTRF 20350 and CONST 20360, operand input and result output multiplexing logic including MULTIM 20328 and MULTRM 20334, and arithmetic operation logic. MULT 20314's operand input and result output multiplexing logic and memory elements will be described first, followed by description of MULT 20314's arithmetic operation logic.

As previously described, input data, including operands, is provided to MULT 20314's arithmetic operation logic through MULTIN Bus 20354. MULTIN Bus 20354 may be provided with data from three sources. A first source is CONST 20360 which is a 512 word by 32 bit wide Read Only Memory. CONST 20360 is utilized to store constants used in arithmetic operations. In particular, CONST 20360 stores zone fields for unpacked decimal, that is ASCI character, operands. As previously described, unpacked decimal operands are received by OPB 20322 and converted to packed decimal operands for more efficient utilization by MULT 20314. As such, final result outputs generated by MULT 20314 from such operands are in packed decimal format. As will be described below, MULT 20314 may be utilized to convert these packed decimal results into unpacked decimal results by insertion of zone fields. As indicated in Fig. 257, address inputs are provided to CONST 20360 from EXPQ Bus 20325 and from output of mCRD 20346. Selection between these address inputs is provided through CONST Address Multiplexer (CONSTAM) 25710. CONST 20360 addresses will, in general, be provided from EUCL 20310 but alternately may be provided from EXPQ Bus 20325 for special operations.

Operand data is provided to MULTIN Bus 20354 through MULTIM 20328, which is a dual input, 64 bit multiplexer. A first input of MULTIM 20328 is provided from OPQ Bus 20323 and is comprised of operand information provided from OPB 20322. OPQ Bus 20323 is a 56 bit wide bus and operand data appearing thereon may be comprised of 32 bit integer operands; 32 bit packed decimal operands, either provided directly from OPB 20322 or as a result of OPB 20322's conversion of an unpacked decimal to a packed decimal operand; 24 bit single precision operand mantissa fields; or 56 bit double precision floating point operand mantissa fields. As previously described, certain OPQ Bus 20323 may be zero or sign extension filled, depending upon the particular operand.

Second input of MULTIM 20328 is provided from MULTRF 20350. MULTRF 20350 is a 16 word by 64 bit wide random access memory. As indicated in Figs. 203 and 257, MULTRF 20350 is connected between output of RFR 20336, through FR Bus 20337, and to input of MULT 20314's arithmetic operation logic through MULTIM 20328 and MULTIN Bus 20354. MULTRF 20350 may therefore be utilized as a scratch pad memory for storing intermediate results of arithmetic operations, including reiterative arithmetic operations. In addition, a portion of MULTRF 20350 is utilized, as in GRF 10354, as an EU 10122 Stack Mechanism similar to MIS 10368 and MOS 10370 in FU 10120. Operation of EU 10122 Stack Mechanism will be described in a following descripion of EU 10122's interfaces to MEM 10112 and FU 10120. Address inputs (ADR) of MULTRF 20350 are provided from Multiplier Register File Address Multiplexer (MULTRFAM) 25712.

MULTRFAM 25712 is a dual four bit multiplexer comprised, for example, of SN74S258s. In addition to address inputs to MULTRF 20350, MULTRFAM 25712 provides address inputs to EXPRF 20380. As previously described, MULTRF 20350 and EXPRF 20380 together comprise an EU 10122 general register file similar to GRF 10354 and FU 10120. As such, MULTRF 20350 and EXPRF 20380 are addressed in parallel to read and write parallel entries from and to MULTRF 20350 and EXPRF 20380. Address inputs to MULTRFAM 25712 are provided, first, from outputs of mCRD 20346, thus providing microinstruction control of addressing of MULTRF 20350 and EXPRF 20380. Second address input to MULTRFAM 25712 is provided from output of Multiplier Register File Address Counter (MULTRFAC) 25714.

MULTRFAC 25714 is a four bit counter and is used to generate sequential addresses to MULTRF 20350 and EXPRF 20380. Initial addresses are loaded into MULTRFAC 25714 from Multiplier Register File Address Counter Multiplexer (MULTRFACM) 25716. MULTRFACM 25716 is a dual four bit multiplexer. Inputs to MULTRFACM 25716 are provided, first, from outputs of mCRD 20346. This input allows microinstruction selection of an initial address to be loaded into MULTRFAC 25714 to be subsequently used and generating sequential MULTRF 20350 and EXPRF 20380 addresses. Second address input to MULTRFACM 25716 is provided from OPQ Bus 20323. MULTRFACM 25716's input from OPQ Bus 20323 allows a single address, or a starting address of a sequence of addresses, to be selected through JPD Bus 10142 or MOD Bus 10144, for example from MEM 10112 or FU 10120.

Intermediate and final result outputs of MULT 20314 arithmetic logic are provided to data inputs of MULTRF 20350 directly from FR Bus 20337 and from MULTRM 20334. Inputs to MULTRM 20334, in turn, are provided from FR Bus 20337 and from output of CONSIZE 20352 and TSTINT 20320.

FR Bus 20337 is a 64 bit bus connected from 64 bit output of RFR 20336 and carries final and intermediate results of MULT 20314 arithmetic operations. As will become apparent in a following description of MULT 20314 arithmetic operation logic, RFR 20336 output, and thus FR Bus 20337, are 64 bits wide. Sixty-four bits are provided to insure retention of all significant data bits of certain MULT 20314 arithmetic operation intermediate results, in particular operations involving double precision floating point 64 bit mantissa fields. In addition, as will be described momentarily and has been previously stated, MULT 20314 may convert a final result in packed decimal format into a final result in unpacked decimal format. In this operation, a single 32 bit, or one word, packed decimal result is converted into a 64 bit, or two word, unpacked decimal format by insertion of zone fields.

As described above, two parallel data paths are provided to transfer information from FR Bus 20337 into MULTRF 20350. First path is directly from FR Bus 20337 and second path is through Unpacked Decimal Multiplexer (UPDM) 25718 of MULTRM 20334. Direct path is utilized for thirty-two bits of information comprising bits 0 to 23 and bits 56 to 63 of FR Bus 20337. Data path through UPDM 25718 may comprise either bits 24 to 55 of FR Bus 20337, which are connected into a first input of UPDM 25718, or bits 40 through 55 which are connected to a second input of UPDM 25718. Single precision floating point numbers are 32 bit numbers plus two or more guard bits and are thus written into MULTRF 20350 through bits 0 to 23 of the direct path into MULTRF 20350 and through first input (bits 24 to 55) of UPDM 25718. Double precision floating point numbers are 5 bits wide, plus guard bits, and thus utilize the direct path into MULTRF 20350 and the path through first input of UPDM 25718. Bits 56 to 63 of direct path are utilized for guard bits of double precision floating point numbers. Both integer and packed decimal numbers utilize bits 24 through 55 of FR Bus 20337, and are thus written into MULTRF 20350 through first input of UPDM 25718. As previously described, bits 0 to 23 of these operands are filled by sign extension.

a.a.a. Container Size Check

As stated above, MULTRM 20334 has an Input from CONSIZE 20352. As will be described below with reference to TSTINT 20320, CONSIZE 20352 performs a "container size" check upon each store back of results from EU 10122 to MEM 10112. CONSIZE 20352 compares the number of significant bits in a result to be stored back to the logical descriptor describing the MEM 10112 address space that result is to be written into. Where reiterative write operations to MEM 10112 are required to transfer a result into MEM 10112, that is a string transfer, container size information may read from CONSIZE 20352 through Container Size Driver (CONSIZED) 25720 and MULTRM 20334 and written into MULTRF 20350. This allows EU 10122, using container size information stored in MULTRM 20350, to perform continuous container size checking during a string transfer of result from EU 10122 to MEM 10112. In addition, as will be described momentarily, container size information may be read from CONSIZE 20352 to JPD Bus 10144.

b.b.b. Final Result Output Multiplexer 20324

Referring finally to FROM 20324, as previously described FROM 20324 is utilized to transfer, in general, results of EU 10122 arithmetic operations onto JPD Bus 10142 for transfer to MEM 10112 or FU 10120. As indicated in Fig. 257, FROM 20324 is comprised of 24 bit Final Result Bus Driver (FRBD) 25722 and Result Bus Driver (RBR) 25724. Input of FRBD 25722 is connected from FR Bus 20337 and allows data appearing thereon to be transferred onto JPD Bus 10142. In particular, FRBD 25722 is utilized to transfer 24 bit mantissa fields of single precision floating point results onto JPD Bus 10142 in parallel with a corresponding exponent field from EXP 20316. RBR 25724 input is connected from RSLT Bus 20388 to allow

output of UPDM 25718 to be transferred onto JPD Bus 10142. RBR 25724, RSLT Bus 20388, and UPDM 25718 are used, in general, to transfer final results of EU 10122 operations from output of MULT 20314 onto JPD Bus 10142. Final results transferred by this data path include integer, packed and unpacked decimal results, and mantissa fields of double precision floating point results. Both unpacked decimal numbers and mantissa fields of double precision floating point numbers are comprised of two 32 bit words and are thus transferred onto JPD Bus 10142 in two sequential transfer operations.

Having described structure and operation of MULT 20314's memory elements and input and output circuitry, MULT 20314's arithmetic operation logic will be described next below.

4. Test and Interface Logic 20320 (Figs. 260—268)

As previously described, TSTINT 20320 includes CONSIZE 20352, ECPT 20328, TSTCOND 20384, and INTRPT 20388. CONSIZE 20352, as previously described, performs "container size" check operations when results of EU 10122 operations are to be written into MEM 10112. That is, CONSIZE 20352 compares size or number of significant bits, of an EU 10122 result to the capacity, or container size, of the MEM 10112 location that EU 10122 result is to be written into. As indicated, in Fig. 203, CONSIZE 20352 receives a first input, that is the results of EU 10122 operations, from FR Bus 20337. A second input of CONSIZE 20352 is connected to LENGTH Bus 20226 to receive length field of logical descriptors identifying MEM 10112 address space into which those EU 10122 results are to be written. CONSIZE 20352 includes logic circuitry, for example a combination of Read Only Memory and Field Programmable Logic Arrays, for examining EU 10122 operation results appearing on FR Bus 20337 and determining the number of bits of data in those results. CONSIZE 20352 compares EU 10122 result size to logical descriptor length field and, in particular, if result size exceeds logical descriptor length, provides an alarm output to ECPT 20328, described below.

TSTCOND 20384, previously described and which will be described further below, is an interface circuit between FU 10120 and EU 10122. TSTCOND 20384 allows FU 10120 to specify and examine results of certain test operations performed by EU 10122 with respect to EU 10122 operations.

ECPT 20328 monitors certain EU 10122 operations and provides outputs indicating when certain "exceptions" have occurred. These exceptions include attempted divisions by zero, floating point exponent underflow or overflow, and integer container size fault.

INTRPT 20388 is again an interface between EU 10122 and FU 10120 allowing FU 10120 to interrupt EU 10122 operations. INTRPT 20388 allows FU 10120 to direct EU 10122 to execute certain operations to aid in handling of certain FU 10120 events previously described.

Operation of CONSIZE 20352, ECPT 20328, TSTCOND 20384, INTRPT 20388, and other features of EU 10122's interface with FU 10120 will be described further below in the following description of operation of that interface and of operation of certain EU 10122 internal mechanisms, such as FU 10120 Stack Mechanisms.

a.a. FU 10120/EU 10122 Interface

As previously described, EU 10122 and FU 10120 are asychronous processors, each operating under its own microcode control. EU 10122 and FU 10120 operate simultaneously and independently of each other but are coupled, and their operations coordinated, by interface signals described below. Should EU 10122 not be able to respond immediately to a request from FU 10120, FU 10120 will idle until EU 10122 becomes available; conversely, should EU 10122 not receive, or have present, operands or a request for operations from FU 10120, EU 10122 will remain in idle state until operands and requests for operations are received from FU 10120.

In normal operation, EU 10122 manipulates operands under control of FU 10120, which in turn is under control of SOPs of a user's program. When FU 10120 requires arithmetic or logical manipulation of an operand, FU 10120 dispatches a command, that is an Execute Unit Dispatch Pointer (EUDP) to EU 10122. As previously described, an EUDP is basically an initial address into EUSITT 20344. An EUDP identifies starting location of a EU 10122 microinstruction sequence performing the required operation upon operands. Operands are fetched from MEM 10112 under FU 10120 control, as previously described, and are transferred into OPB 20322. Those operands are then called from OPB 20322 by EU 10122 and transferred into MULT 20314 and EXP 20316 as previously described. After the required operation is completed, FU 10120 is notified that a result is ready. At this point, FU 10120 may check certain test conditions, for example through TSTCOND 20384, such as whether an integer or decimal carry bit is set or whether a mantissa sign bit is set or reset. This test operation is utilized by FU 10120 for conditional branching and synchronization of FU 10120 and EU 10122 operations. Exception checking, by ECPT 20328, is also performed at this time. Exception checking determines, for example, whether division by zero was attempted or if a container size fault has occurred. In general, FU 10120 is not informed of exception errors until FU 10120 requests exception checking. After results are transferred into FU 10120 or MEM 10112 by EU 10122, EU 10122 goes to idle operation until a next operation is requested by FU 10120.

Having briefly described overall interface operation between FU 10120 and EU 10122, operation of that interface, referred to as handshaking, will be described in greater detail next below. In general, handshaking operation between EU 10122 and FU 10120 during normal operation may be regarded as following into six operations. These operations may include, for example, loading of COMQ 20342, loading of OPB 20322, storeback or transfer of results from EU 10122 to FU 10120 or MEM 10112, check of test

conditions, exception checking, and EU 10122 idle operation. Handshaking between FU 10120 and EU 10122 will be described below for each of these classes of operation, in the order just referred to.

a.a.a. Loading of Command Queue 20342 (Fig. 260)

Referring to Fig. 260, a schematic representation of EU 10122's interface with FU 10120 for purposes of loading COMQ 20342 as shown. During normal SOP directed JP 10114 operation, 8 bit operation (OP) codes are parsed from the instruction stream, as previously described, and concatenated with dialect information to address EUSDT 20266 also as previously described. EUSDT 20266 provides corresponding addresses, that is EUDPs, to EUSITT 20344.

Dialect information specifies the S-Language currently being executed and, consequently, the group of microinstruction sequences available in EUSITT 20344 for that S-Language. As previously described, FU 10120 may specify four S-Language dialects with up to 256 EU 10122 microinstruction sequences per dialect, or 8 dialects with up to 128 microinstruction sequences per dialect.

EUDPs provided by EUSDT 20266 are comprised of a 9 bit address field, a 2 bit operand information field, and a 1 bit flag field, as previously described. Address field is starting address of a microinstruction sequence in EUSITT 20344 and EU 10122 will perform the operation directed by that microinstruction sequence. EUSITT 20344 requires 11 bits of address field and the 9 bit address field of EUDPs are mapped into an 11 bit address field by left justification and zero filling.

FU 10120 may also dispatch, or select, any EU 10122 microinstruction controlled operation from JPD Bus 10142. Such EUDPs are provided from JPD Bus 10142 to data input of EUSITT 20344 and passed directly through to mCRD 20346. Before a EUDP may be provided from JPD Bus 10142, however, FU 10120 provides a check operation comparing that EUDP to a list of legal, or allowed, EUSITT 20344 addresses stored in MEM 10112. A fault will be indicated if an EUDP provided through JPD Bus 10142 is not a legal EUSITT 20344 address. Alternately, FU 10120 may effectively provide an EUDP, or EUSITT 20344 addresses, from a literal field in a FU 10120 microinstruction word. Such a FU 10120 microinstruction word literal field may be effectively utilized as an SOP into EUSDT 20266.

Handshaking between EU 10122 and FU 10120 during load COMQ 20342 operations may proceed as illustrated in Fig. 260. A twelve bit EUDP may be placed on EUDIS Bus 20206 and Control Signal Load Command Queue (LDCMQ) asserted. If COMQ 20342 is full, EU 10122 raises control signal Command Hold (CMDHOLD) which causes FU 10120 to remain in State M0 until there is room in COMQ 20342. As previously described, COMQ 20342 is comprised of two, two word buffers wherein one buffer is utilized for normal SOP operation and the other utilized for control of FU 10120 and EU 10122 internal mechanism operation.

EUDPs are loaded into COMQ 20342 when state timing signals M1CPT and M1 are asserted. If a EUDP being transferred into COMQ 20342 concerns a double precision floating point operation, control signal Set Double Precision (SETDP) is asserted. SETDP is utilized to control OPB 20322, and because single precision and precision floating point operations otherwise utilize the same SOP and thus would otherwise refer to same EUSITT 20344 microinstruction sequence.

At this point, a EUDP has been loaded into COMQ 20342 and will be decoded to control FU 10120 operation by EUCL 20310 as previously described. Each particular EUDP will be cleared by that EUDPs EUSITT 20344 microinstruction sequence after the requested microinstruction sequence has been executed.

b.b.b. Loading of Operand Buffer 20320 (Fig. 261)

Referring to Fig. 261, a diagramic representation of the interface and handshaking between EU 10122, FU 10120 and MEM 10112 for loading OPB 20322 is shown. Control signal Clear Queue Full (CLQF) from EU 10122 must be asserted by EU 10122 before FU 10120 initiates a request to MEM 10112 for an operand to be transferred to EU 10122. CLQF clears and "EU 10122's OPB 20322 Full" condition in FU 10120. CLQF indicates, thereby, that there is room in OPB 20322 to receive operands. If FU 10120 is in a "EU 10122's OPB 20322 Full" condition and further operand is required to be transferred to EU 10122, FU 10120 will remain in State M1 until CLQF is asserted.

At the beginning of execution of a particular SOP, FU 10120 may transfer two operands to OPB 20322 without "EU 10122's OPB 20322 Full" condition occurring. This is because EU 10122 is idle at the beginning of an SOP execution and generally immediately unloads a first operand from OPB 20322 before a second operand arrives.

Control signal Job Processor Operand (JPOP) provided from FU 10120 must be non-asserted for operands to be transferred from MEM 10112 to OPB 20322 through MOD Bus 10144. This is the normal condition of JPOP. If JPOP is asserted, OPB 20322 is loaded with data from JPD Bus 10142. Data is strobed into OPB 20322 from JPD Bus 10142 by control signals M1CPT and JPOP. Operands read from MEM 10112, however, are transferred into OPB 20322 through MOD Bus 10144 when MEM 10112 asserts DAVEB to indicate that valid data from MEM 10112 is available on MOD Bus 10144. DAVEB is also utilized to strobe data on MOD Bus 10144 into OPB 20322. If control signal ZFILL from MEM 10112 is asserted at this point, ZFILL is interpreted during integer operand operations to indicate that those operands are unsigned and

should be left zero filled, rather than sign extended. If data is being provided from JPD Bus 10142 rather than from MEM 10112, that is if JPOP is asserted, bit 11 of current EUDP may be utilized to perform the same function as ZFILL during loading of OPB 20322 from MOD Bus 10144.

Loading of OPB 20322 is controlled, in part, by bits 9 and 10 of EUDPs provided from FU 10120 through EUDIS Bus 20206. Bit 9 indicates length of a first operand while bit 10 indicates length of a second operand. Operand length, together with operand type specified in address portion of a EUDP, determines how a particular operand is unloaded from OPB 20322 and transferred into MULT 20314 and EXP 20316.

At this point, both COMQ 20342 and OPB 20322 have been loaded with, respectively, EUDPs and operands. It should be noted that operands are generally not transferred into OPB 20322 before a corresponding EUDP is loaded into COMQ 20342. Operands and EUDPs may, however, be simultaneously transferred into EU 10122. If other operands are required for a particular operation, those operands are loaded into OPB 20322 as described above.

c.c.c. Storeback (Fig. 262)

Referring to Fig. 262, a diagramic representation of a storeback, or transfer, of results to MEM 10112 from EU 10122 and handshaking performed therein is shown. When a final result of a EU 10122 operation is available, EU 10122 asserts control signal Data Ready (DRDY). FU 10120 thereupon responds with control signal Transfer to JPD Bus 10142 (XJPD), which gates EU 10122's result onto JPD Bus 10142. In normal operation, that is execution of SOPs, FU 10120 causes EU 10122's result to be stored back into a destination in MEM 10112, as selected by a physical descriptor provided from FU 10120. Alternately, a result may be transferred into FU 10120, 32 bits, or one word, at a time.

FU 10120 may, as described above and described further below, check EU 10122 test conditions during storeback of results. FU 10120 generates control signal Transfer Complete (XFRC) once the storeback operation is completed. XFRC also indicates to EU 10122 that EU 10122's results and test conditions have been accepted by FU 10120, so that EU 10122 need no longer assert these results and test conditions.

d.d.d. Test Conditions (Fig. 263)

Referring to Fig. 263, a diagramic representation of checking of EU10122 test conditions by FU 10120, and handshaking therein, is shown. As previously described, test results indicating certain conditions and operations of EU 10122 are sampled and stored in TSTCOND 20384 and may be examined by FU 10120. When DRDY is asserted by EU 10122, FU 10120 may select, for example, one of 8 EU 10122 conditions to test, as well as transferring results as described above. EU 10122 conditions which may be tested by FU 10120 are listed and described below. Such conditions, as whether a final result is positive, negative, or zero, may be checked in order to facilitate conditional branching of FU 10120 operations as previously described. FU 10120 specifies a condition to be tested through Test Condition Select signals (TEST(24)). FU 10120 asserts control signal EU Test Enable (EUTESTEN) to EU 10122 to gate the selected test condition. That selected test condition then appears as Data Signal Test Condition (TC) from EU 10122 to FU 10120. A TC of logic 1 may, for example, indicate that the selected condition is false while a TC of logic 0 may indicate that the selected condition is true. FU 10120 indicates that FU 10120 has sensed the requested test condition, and that the test condition need no longer be asserted by EU 10122, by asserting control signal XFRC.

e.e.e. Exception Checking (Fig. 264)

Referring to Fig. 264, a diagramic representation of exception checking of EU 10122 exceptions by FU 10120, and handshaking therein, is shown. As previously described, any EU 10122 exception conditions may be checked by FU 10120 as FU 10120 is initiating storeback of EU 10122 results. Exception checking may detect, for example, attempted division by zero, floating point exponent underflow or overflow, or a container size fault. An attempted division by zero or floating point underflow or overflow may be checked before storeback, that is without specific request by FU 10120.

As previously described, a container size fault is detected by CONSIZE 20352 by comparing length of result with size of destination container in MEM 10112. Container size exception checking occurs during store back of EU 10122 results, that is while FU 10120 is in State SB. Container size is automatically performed by EU 10122 hardware, that is by CONSIZE 20352, only on results of less than 33 bits length. Size checking of larger results, that is larger integers and BCD results, is performed by a microcode routine, using CONSIZE 20352's output, as transfer of such larger results is executed as string transfer. It is unnecessary to perform container size check for either single or double precision floating point results as these data types always occupy either 32 or 64 bits. Destination container size is provided to CONSIZE 20352 through LENGTH Bus 20226.

Control signal Length to Memory AON or Random Signals (LMAONRS) is generated by FU 10120 from Type field of the logical descriptor corresponding to a particular EU 10122 result. LMAONRS indicates that the results data type is an unsigned integer. LMAONRS determines the manner in which a required container size of the EU 10122 result is determined. After receivng this information from LMAONRS, EU 10122 determines whether destination container size in MEM 10112 is sufficiently large to contain the EU

10122 result. If that destination container size is not sufficiently large, a container size fault is detected by CONSIZE 20352, or through an EU 10122 microinstruction sequence.

Container size faults, as well as division by zero and exponent underflow and overflow faults, are signaled to FU 10120 when FU 10120 asserts control signal Check Size (CKSIZE). At this time, EU 10122 asserts control signal Exception (EXCPT) if any of the above faults has occurred. If a fault has occurred, an Event request to FU 10120 results. When an Event request is honored by FU 10120, FU 10120 may interrupt EU 10122 and dispatch EU 10122 to a microinstruction routine that transfers those exception conditions onto JPD Bus 10142. If a container size fault has caused that exception condition, EU 10122 may transfer to FU 10120 the required container size through JPD Bus 10142.

f.f.f. Idle Routine

Finally, when a current EU 10122 operation is completed, EU 10122 goes into an Idle loop microinstruction routine. If necessary, FU 10120 may assert control signal Excute Unit Abort (EUABORT) to force EU 10122 into Idle loop microinstruction routine until EU 10122 is required for further operations.

g.g.g. EU 10122 Stack Mechanism (Figs. 265, 266, 267)

As previously described, EU 10122 may perform either of two classes of operations. First, EU 10122 may perform arithmetic operations in execution of SOPs of user's programs. Second, EU 10122 may operate as an arithmetic calculator assisting operation of FU 10120's internal mechanisms and operations, referred to as kernel operations.

In kernel operation, EU 10122 acts as an arithmetic calculator for FU 10120 during address generation, address translation, and other kernel functions. In kernel mode, EU 10122 is executing microinstruction sequences at request of FU 10120 kernel microinstruction sequences, rather than at request of an SOP. In general, these kernel operations are vital to operation of JP 10114. FU 10120 may interrupt EU 10122 operations with regard to SOPs and initiate EU 10122 microinstruction sequences to perform kernel operations.

When interrupted, EU 10122 saves EU 10122's current operating state in a one level deep stack. EU 10122 may then accept an EUDP from that portion of COMQ 20342 utilized to receive and store EUDPs regarding FU 10120's and EU 10122's internal, or kernel, operations. When requesting kernel operations by EU 10122, FU 10120 generally transfers operands to OPB 20322 through JPD Bus 10142, and receives EU 10122 final results through JPD Bus 10142. Operands may also be provided to EU 10122 through MOD Bus 10144. After EU 10122 has completed a requested kernel operation, EU 10122 reloads operating state from its internal stack and continues normal operation from the point normal operation was interrupted.

Should another interrupt from FU 10120 occur while a prior interrupt is being executed, EU 10122 moves current state and data, that is of first interrupt, to MEM 10112. EU 10122 requests FU 10120 store state and date of first interrupt in MEM 10112 by requesting an "EU 10122 Stack Overflow" Event. EU 10122's "normal" state, that is state and data pertaining to the operation EU 10122 is executing at time of occurrence of first interrupt, is stored in an EU 10122 internal stack and remains there. EU 10122 then begins executing second interrupt. When EU 10122 has completed operations for second interrupt, state from first interrupt is reloaded from MEM 10112 by EU 10122 requesting a "EU 10122 Stack Underflow" Event to FU 10120. EU 10122 then completes execution of first interrupt and reloads state and resumes execution of normal operation, that is the operation being executed before the first interrupt.

EU 10122 is therefore capable of handling interrupts from FU 10120 during two circumstances. First interrupt circumstance is comprised of interrupts occurring during normal operation, that is while executing SOPs of user's programs. Second circumstance arises when interrupts occur during kernel operations, that is during execution of microinstruction sequences for handling interrupts. EU 10122 operation will be described next below for each of these circumstances, and in the order referred to.

Referring to Fig. 265, a diagramic representation of EU 10122's stack mechanisms, previously described, is shown. Those portions of EU 10122's stack mechanisms residing within EU 10122 are comprised of EU 10122's Current State Registers (EUCSRs) 26510 and EU 10122's Internal Stack (EUIS) 26512. EUCSR 26510 is comprised of EU 10122's internal registers which contain data and state of current EU 10122 operation. EUCSR 26510 may be comprised, for example, of mCRD 20346, registers of TSTINT 20320, and the previously described registers within MULT 20314 and EXP 20316.

State and data contained in EUCSR 26510 is that of the operation currently being executed by EU 10122. This current state may, for example, be that of a SOP currently being executed by EU 10122, or that of an interrupt, for example a fourth interrupt of a nested sequence of interrupts, requested by FU 10120.

EUIS 26512 is comprised of certain registers of MULTRF 20350 and EXPRF 20380. EUIS 26512 is utilized to store and save current state of an SOP operation currently being executed by EU 10122 and which has been interrupted. State and data of that SOP operation will remain stored in EUIS 26512 regardless of the number of interrupts which may occur on a nested sequence of interrupts requested by FU 10120. State and data of the interrupted SOP operation will be returned from EUIS 26512 to EUCSR 26510 when all interrupts have been completed.

Final portion of EU 10122's stack mechanism is that portion of EU 10122's internal stack (EUES) 26514

residing in MEM 10112. EUES 26514 is comprised of certain MEM 10112 address locations used to store state and data of successive interrupt operations of sequences of nested interrupts. That is, if a sequence of four interrupts is requested by FU 10120, state and data of fourth interrupt will reside in EUCSR 26510 while state and data of first, second, and third interrupts have been transferred, in sequence, into EUES 26514. In this respect, and as previously described operation of EU 10122's stack mechanisms is similar to that of, for example, MIS 10368 and SS 10336 previously described with reference to Fig. 103.

As described above, an interrupt may be requested of EU 10122 by FU 10120 either during EU 10122 normal operation, that is during execution of SOPs by EU 10122, or while EU 10122 is executing a previous interrupt requested by FU 10120. Operation of EU 10122 and FU 10120 upon occurrence of an interrupt during EU 10122 normal operation will be described next below.

Referring to Fig. 266, a diagramic representation of handshaking between EU 10122 and FU 10120 during an interrupt of EU 10122 while EU 10122 is operating in normal mode is shown and should be referred to in conjunction with Fig. 265. For purposes of the following discussions, interrupts of EU 10122 operations by FU 10120 are referred to as nanointerrupts to distinguish from interrupts internal to FU 10120.

FU 10120 interrupts normal operation of EU 10122 by assertion of control signal Nano-Interrupt (NINTP) during State M0 of FU 10120 operation. NINTP may be masked by EU 10122 during certain critical EU 10122 operations, such as arithmetic operations. If NINTP is masked by EU 10122, FU 10120 will remain in State NW until EU 10122 acknowledges the interrupt.

Upon receiving NINTP from FU 10120, EU 10122s transfers state and data of current SOP operation from EUCSR 26510 to EUIS 26512. EU 10122 then asserts control signal Nano-Interrupt Acknowledge (NIACK) to FU 10120 to acknowledge availability of EU 10122 to accept a nanointerrupt. FU 10120 will then enter State M1 and place an EUDP on EUDIS Bus 20206. Loading of COMQ 20342 then proceeds as previously described, with EU 10122 loading nanointerrupt EUDPs into the appropriate registers of COMQ 20342. COMQ 20342 is loaded as previously described and, if JPOP is asserted, data transferred into OPB 20322 from JPD Bus 10142. If JPOP is not asserted, data is taken into OPB 20322 from MOD Bus 10144. EU 10122 then proceeds to execute the required nanointerrupt operation and storing back of results and checking of test conditions proceeds as previously described for EU 10122 normal operation. In general, exception checking is not performed. When EU 10122 has completed execution of the nanointerrupt operation, EU 10122 transfers state and data of the interrupted SOP operation from EUIS 26512 to EUCSR 26510 and resumes execution of that SOP. At this point, EU 10122 asserts control signal Nano-Interrupt Trap Enable (NITE). NITE is received and tested by FU 10120 to indicate end of nanointerrupt processing.

Referring to Fig. 267, a diagramic representation of interfaces between EU 10122, FU 10120, and MEM 10112 during nested, or sequential, EU 10122 interrupts for kernel operations, and handshaking therein, is shown. During the following discussion, it is assumed that EU 10122 is already processing a nanointerrupt for a kernel operation submitted to EU 10122 by FU 10120. FU 10120 may then submit a second, third, or fourth, nanointerrupt to EU 10122 for a further kernel operation. FU 10120 will assert NINTP to request a nanointerrupt of EU 10122. EU 10122's normal mode state and data from a previously executing SOP operation has been stored in, and remains in, EUIS 26512. Current state and data of currently executing nanointerrupt operation in EUCSR 26510 will be transferred to EUES 26514 in MEM 10112 to allow initiation of pending nanointerrupt. EU 10122 will at this time assert NIACK and control signal Execute Unit Event (EXEVT). EXEVT to FU 10120 informs FU 10120 that an EU 10122 Event has occurred, specifically, and in this case, EXEVT requests FU 10120 service of an EU 10122 Stack Overflow. FU 10120 is thereby trapped to an "EU 10122 Stack Overflow" Event Handler microinstruction sequence. This handler transfers current state and data of interrupted nanointerrupt previously executing in EU 10122 into EUES 26514. State and data of interrupted nanointerrupt is transferred to EUES 26514, one 32 bit word at a time. FU 10120 asserts control signals XJPD to gate each of these state and data words onto JPD Bus 10142 and controls transfer of these words into EUES 26514.

Processing of new nanointerrupt proceeds as described above with reference to interrupts occurring during normal operation. If any subsequent nanointerrupts occur, they are handled in the same manner as just described; FU 10120 signals a nanointerrupt to FU 10120, current EU 10122 state and data is saved by FU 10120 in EUES 26514, and new nanointerrupt is processed. After a nested nanointerrupt, that is a nanointerrupt of a sequence of nanointerrupts, has been serviced, EU 10122 asserts control signal EU 10122 Trap (ETRAP) to FU 10120 to request a transfer of a previous nanointerrupt's state and data from EUES 26514 to EUCSR 26510. FU 10120 will retrieve that next previous nanointerrupt state and data from EUES 26514 through MOD Bus 10144 and will transfer that data and state onto JPD Bus 10142. This state and data is returned, one 32 bit word at a time, and is strobed into EU 10122 by JPOP from FU 10120. Processing of that prior nanointerrupt will then resume. The servicing of successively prior nanointerrupts will continue until all previous nanointerrupts have been serviced. Original state and data of EU 10122, that is that of SOP operation which was initially interrupted, is then returned to EUCSR 26510 from EUIS 26512 and execution of that SOP resumed. At this time, EU 10122 asserts NITE to indicate end of EU 10122 kernel operations in regard to nanointerrupts.

Having described structure and operation of EU 10122, FU 10120 and MEM 10112, with respect to servicing of kernel operation nanointerrupts by EU 10122, loading of EU 10122's EUSITT 20344 with microinstruction sequences will be described next below.

h.h.h.h Loading of Execute Unit S-Interpreter Table 20344 (Fig. 268)

Referring to Fig. 268, a diagramic representation of interface and handshaking between EU 10122, FU 10120, MEM 10112, and DP 10118 during loading of microinstructions into EUSITT 20344 is shown. As previously described, EUSITT 20344 contains all microinstructions required for control of EU 10122 in executing kernel nanointerrupt operations and in executing arithmetic operations in response to SOPs of user's programs. EUSITT 20344 may store microinstruction sequences for interpreting arithmetic SOPs of user's programs for, for example, up to 4 different S-Language Dialects. In general, a capacity of storing microinstruction sequences for arithmetic operations in up to 4 S-Language Dialects is sufficient for most requirements, so that EUSITT 20344 need be loaded with microinstruction sequences only at initialization of CS 10110 operation. Should microinstruction sequences for arithmetic operations of more than 4 S-Language Dialects be required, those microinstruction sequences may be loaded into EUSITT 20344 in the manner as will be described below.

As previously described, a portion of the microinstructions stored in EUSITT 20344 is contained in Read Only Memories and is thus permanently stored in EUSITT 20344. Microinstruction sequences permanently stored in EUSITT 20344 are, in general, those required for execution of kernel operations. Microinstruction sequences permanently stored in EUSITT 20344 include those used to assist in writing other EU 10122 microinstruction sequences into EUSITT 20344 as required. Certain microinstruction sequences are stored in a Random Access Memory, referred to as the Writeable Control Store (WCS) portion of EUSITT 20344, and include these for interpreting arithmetic operation SOPs of various S-Language Dialects.

Writing of microinstruction sequences into EU 10122 is initialized by forcing EU 10122 into an Idle state. Initialization of EU 10122 is accomplished by FU 10120 asserting EUABORT or by DP 10118 asserting control signal clear (CLEAR). Either EUABORT or CLEAR will clear a current operation of EU 10122 and force EU 10122 into Idle state, wherein EU 10122 waits for further EUDPs provided from FU 10120. FU 10120 then dispatches a EUDP initiating loading of EUSITT 20344 to EU 10122 through EUDIS Bus 20206. Load EUSITT 20344 EUDP specifies starting address of a two step microinstruction sequence in the PROM portion of EUSITT 20344. This two step microinstruction sequence first loads zeros into SCAG 25536, which as previously described provides read and write addresses to EUSITT 20344. EUSITT 20344 load microinstruction sequence then reads a microinstruction from EUSITT 20344 to mCRD 20346. This microinstruction specifies conditions for handshaking operations with FU 10120 so that loading of EUSITT 20344 may begin. At this time, and from this microinstruction word, EU 10122 asserts control signal DRDY to FU 10120 to indicate that EU 10122 is ready to accept EUDPs from FU 10120 for directing loading of EUSITT 20344. This initial microinstruction also generates a write enable control signal for the WCS portion of EUSITT 20344, inhibits loading of mCRD 20346 from EUSITT 20344, and inhibits normal loading operations of NXTR 25540 and SCAG 25536. This first microinstruction also directs NASS 25526 to accept address inputs from SCAG 25536 and, finally, causes NITE to FU 10120 to be asserted to unmask nanointerrupts from FU 10120.

FU 10120 then generates a read request to MEM 10112, and MEM 10112 transfers a first 32 bit word of a EU 10122 microinstruction word onto JPD Bus 10142. Each such 32 bit word from MEM 10112 comprises one half of a 64 bit microinstruction word of EU 10122. When FU 10120 receives DRDY from EU 10122, FU 10120 generates control signal Load Writeable Control Store (LDWCS). LDWCS in turn transfers a 32 bit word on JPD Bus 10142 into a first address of the WCS portion of EUSITT 20344. A next 32 bit half word of a EU 10122 microinstruction word is then read from MEM 10112 through JPD Bus 10142 and transferred into the second half of that first address within the WCS portion of EUSITT 20344. The address in SCAG 25536 is then incremented to select a next address within EUSITT 20344 and the process just described repeated automatically, including generation of DRDY and LDWCS, until loading of EUSITT 20344 is completed.

After loading of EUSITT 20344 is completed, the loading process is terminated when FU 10120 asserts NINTP, or DP 10118 asserts Control Signal Load Complete (LOADCR). Either NINTP or LOADCR releases control of operation of NAG 20340 to allow EU 10122 to resume normal operation.

The above descriptions have described structure and operation of EU 10122, including: execution of various arithmetic operations utilizing various operand formats; operation of EU 10122, FU 10120, and MEM 10112 with regard to handshaking; loading of EUDPs and operands; storeback of results; checking of test conditions and exceptions; EU 10122 Stack Mechanisms during normal and kernel operations; and loading of EU 10122 microinstruction sequences into EUSITT 20344. IOS 10116 and DP 10118 will be described next below, in that order.

D. I/O System 10116 (Figs. 204, 206, 269)

Referring to Fig. 204, a partial block diagram of IOS 10116 is shown. As previously described, IOS 10116 operates as an interface between CS 10110 and the external world, for example, ED 10124. A primary function of IOS 10116 is the transfer of data between CS 10110, that is MEM 10112, and the external world. In addition to performing transfers of data, IOS 10116 controls access between various data sources and sinks of ED 10124 and MEM 10112. As previously described, IOS 10116 directly addresses MEM 10112's physical address space to write data into or read data from MEM 10112. As such, IOS 10116 also performs address translation, a mapping operation required in transferring data between MEM 10112's physical

address space and address spaces of data sources and sinks in ED 10124.

As shown in Fig. 204, IOS 10116 includes Data Mover (DMOVR) 20410, Input/Output Control Processor (IOCP) 20412, and one or more data channel devices. IOS 10116's data channel devices may include ECLIPSE® Burst Multiplexer Channel (EBMC) 20414, NOVA Data Channel (NDC) 20416, and other data channel devices as required for a particular configuration of a CS 10110 system. IOCP 20412 controls and directs transfer of data between MEM 10112 and ED 10124, and controls and directs mapping of addresses between ED 10124 and MEM 10112's physical address space. IOCP 20412 may be comprised, for example, of a general purpose computer, such as an ECLIPSE® M600 computer available from Data General Corporation of Westboro, Massachusetts.

EBMC 20414 and NDC 20416 comprise data channels through which data is transferred between ED 10124 and IOS 10116. EBMC 20414 and NDC 20416 perform actual transfers of data to and from ED 10124, under control of IOCP 20412, and perform mapping of ED 10124 addresses to MEM 10112 physical addresses, also under control of IOCP 20412. EBMC 20414 and NDC 20416 may respectively be comprised, for example, of an ECLIPSE® Burst Multiplexer Data Channel and a NOVA® Data Channel, also available from Data General Corporation of Westboro, Massachusetts.

DMOVR 20410 comprises IOS 10116's interface to MEM 10112. DMOVR 20410 is the path through which data and addresses are transferred between EBMC 20414 and NDC 20416 and MEM 10112. Additionally, DMOVR 20410 controls access between EBMC 20414, NDC 20416, and other IOS 10116 data channels, and MEM 10112.

ED 10124, as indicated in Fig. 204, may be comprised of one or more data sinks and sources. ED 10124 data sinks and sources may include commercially available disc drive units, line printers, communication lengths, tape units, and other computer systems, including other CS 10110 systems. In general, ED 10124 may include all such data devices as are generally interfaced with a computer system.

a. I/O System 10116 Structure (Fig. 204)

Referring first to the overall structure of IOS 10116, data input/output of ECLIPSE® Burst Multiplexer Channel Adapter and Control Circuitry (BMCAC) 20418 of EBMC 20414 is connected to bi-directional BMC Address and Data (BMCAD) Bus 20420. BMCAD Bus 20420 in turn is connected to data and address inputs and outputs of data sinks and sources of ED 10124.

Similarly, data and address inputs and outputs of NOVA® Data Channel Adapter Control Circuits (NDCAC) 20422 in NDC 20416 is connected to bi-directional NOVA® Data Channel Address and Data (NDCAD) Bus 20424. NDCAD Bus 20424 in turn is connected to address and data inputs and outputs of data sources and sinks of ED 10124. BMCAD Bus 20420 and NDCAD Bus 20424 are paths for transfer of data and addresses between data sinks and sources of ED 10124 and IOS 10116's data channels and may be expanded as required to include other IOS 10116 data channel devices and other data sink and source devices of ED 10124.

Within EBMC 20414, bi-directional data input and output of BMCAC 20418 is connected to bi-directional input and output of BMC Data Buffer (BMCDB) 20426. Data inputs and data outputs of BMCDB 20426 are connected to, respectively, Data Mover Output Data (DMOD) Bus 20428 and Data Mover Input Data (DMID) Bus 20430. Address outputs of BMCAC 20418 are connected to address inputs of Burst Multiplexer Channel Address Translation Map (BMCATM) 20432 and address outputs of BMCATM 20432 are connected onto DMID Bus 20430. A bi-directional control input and output of BMCATM 20432 is connected from bi-directional IO Control Processor Control (IOCPC) Bus 20434.

Referring to NDC 20416, as indicated in Fig. 204 data inputs and outputs of NDCAC 20422 are connected, respectively, from DMOD Bus 20428 and to DMID Bus 20430. Address outputs of NDCAC 20422 are connected to address inputs of NOVA® Data Channel Address Translation Map (NDCATM) 20436. Address outputs of NDCATM 20436 are, in turn, connected onto DMID Bus 20430. A bi-directional control input and output of NDCATM 20436 is connected from IOCPC Bus 20434.

Referring to IOCP 20412, a bi-directional control input and output of IOCP 20412 is connected from IOCPC Bus 20434. Address and data output of IOCP 20412 is connected to NDCAD Bus 20424. An address output of IOCP Address Translation Map (IOCPATM) 20438 within IOCP 20412 is connected onto DMID Bus 20430. Data inputs and outputs of IOCP 20412 are connected, respectively, to DMOD Bus 20428 and DMID Bus 20430. A bi-directional control input and output of IOCP 20412 is connected to a bi-directional control input and output of DMOVR 20410.

Referring finally to DMOVR 20410, DMOVR 20410 includes Input Data Buffer (IDB) 20440, Output Data Buffer (ODB) 20442, and Priority Resolution and Control (PRC) 20444. A data and address input of IDB 20440 is connected from DMID Bus 20430. A data and address output of IDB 20440 is connected to IOM Bus 10130 to MEM 10112. A data output of ODB 20442 is connected from MIO Bus 10129 from MEM 10112, and a data output of ODB 20442 is connected to DMOD Bus 20428. Bi-directional control inputs and outputs of IDB 20440 and ODB 20442 are connected from bi-directional control inputs and outputs of PRC 20444. A bi-directional control input and output of PRC 20444 is connected from a bi-directional control input and output of IOCP 20412 as described above. Another bi-directional control input and output of PRC 20444 is connected to and from IOMC Bus 10131 and thus from a control input and output of MEM 10112. Having described overall structure of IOS 10116, operation of IOS 10116 will be described next below.

b. I/O System 10116 Operation (Fig. 269)

1. Data Channel Devices

Referring first to EBMC 20414, BMCAC 20418 receives data and addresses from ED 10124 through BMCAD Bus 20420. BMCAC 20418 transfers data into BMCDB 20426, where that data is held for subsequent transmission to MEM 10112 through DMOVR 20410, as will be described below. BMCAC 20418 transfers addresses received from ED 10124 to BMCATM 20432. BMCATM 20432 contains address mapping information correlating ED 10124 addresses with MEM 10112 physical addresses. BMCATM 20432 thereby provides MEM 10112 physical addresses corresponding to ED 10124 addresses provided through BMCAC 20418.

When, as will be described further below, EBMC 20414 is granted access to MEM 10112 to write data into MEM 10112, data stored in BMCDB 20426 and corresponding addresses from BMCATM 20432 are transferred onto DMID Bus 20430 to DMOVR 20410. As will be described below, DMOVR 20410 then writes that data into those MEM 10112 physical address locations. When data is to be read from MEM 10112 to ED 10124, data is provided by DMOVR 20410 on DMOD Bus 20428 and is transferred into BMCDB 20426. BMCAC 20418 then reads that data from BMCDB 20426 and transfers that data onto BMCAD Bus 20420 to ED 10124. During transfers of data from MEM 10112 to ED 10124, MEM 10112 does not provide addresses to be translated into ED 10124 addresses to accompany that data. Instead, those addresses are generated and provided by BMCAC 20418.

NDC 20416 operates in a manner similar to that of EBMC 20414 except that data inputs and outputs of NDCAC 20422 are not buffered through a BMCDB 20426.

As previously described, MEM 10112 has capacity to perform block transfers, that is sequential transfers of four 32 bit words at a time. In general, such transfers are performed through EBMC 20414 and are buffered through BMCDB 20426. That is, BMCDB 20426 allows single 32 bit words to be received from ED 10124 by EBMC 20414 and stored therein until a four word block has been received. That block may then be transferred to MEM 10112. Similarly, a block may be received from MEM 10112, stored in BMCDB 20426, and transferred one word at a time to ED 10124. In contrast, NDC 20416 may generally be utilized for single word transfers.

As indicated in Fig. 204, EBMC 20414, NDC 20416, and each data channel device of IOS 10116 each contain an individual address translation map, for example BMCATM 20432 in EBMC 20414 and NDCATM 20436 in NDC 20416. Address translation maps stored therein are effectively constructed and controlled by IOCP 20412 for each data channel device. IOS 10116 may thereby provide an individual and separate address translation map for each IOS 10116 data channel device. This allows IOS 10116 to insure that no two data channel devices, nor two groups of data sinks and sources in ED 10124, will mutually interfere by writing into and destroying data in a common area of MEM 10112 physical address space. Alternately, IOS 10116 may generate address translation maps for two or more data channel devices wherein those maps share a common, or overlapping, area of MEM 10112's physical address space. This allows data stored in MEM 10112 to be transferred between IOS 10116 data channel devices through MEM 10112, and thus to be transferred between various data sink and source devices of ED 10124. For example, a first ED 10124 data source and a first IOS 10116 data channel may write data to be operated upon into a particular area of MEM 10112 address space. The results of CS 10110 operations upon that data may then be written into a common area shared by that first data device and a second data device and read out of MEM 10112 to a second ED 10124 data sink by that second data channel device. Individual mapping of IOS 10116's data channel devices thereby provides total flexibility in partitioning or sharing of MEM 10112's address space through IOS 10116.

2. I/O Control Processor 20412

As described above, IOCP 20412 is a general purpose computer whose primary function is overall direction and control or data transfer between MEM 10112 and ED 10124. IOCP 20412 controls mapping of addresses between IOS 10116's data channel devices and MEM 10112 address space. In this regard, IOCP 20412 generates address translation maps for IOS 10116's data channel devices, such EBMC 20414 and NDC 20416. IOCP 20412 loads these address translation maps into and controls, for example, BMCATM 20432 of EBMC 20414 and NDCATM 20436 and NDC 20416 through IOCPC Bus 20434. IOCP 20412 also provides certain control functions to DMOVR 20410, as indicated in Fig. 204. In addition to these functions, IOCP 20412 is also provided with data and addressing inputs and outputs. These data addressing inputs and outputs may be utilized, for example, to obtain information utilized by IOCP 20412 in generating and controlling mapping of addresses between IOS 10116's data channel devices and MEM 10112. Also, these data and address inputs and outputs allow IOCP 20412 to operate, in part, as a data channel device. As previously described, IOCP 20412 has data and address inputs and outputs connected from and to DMID Bus 20430 and DMOD Bus 20428. IOCP 20412 thus has access to data being transferred between ED 10124 and MEM 10112, providing IOCP 20412 with direct access to MEM 10112 address space. In addition, IOCP 20412 is provided with control and address outputs to NDCAD Bus 20424, thus allowing IOCP 20412 partial control of certain data source and sink devices in ED 10124.

3. Data Mover 20410 (Fig. 269)

a.a. Input Data Buffer 20440 and Output Data Buffer 20442

As described above, DMOVR 20410 comprises an interface between IOS 10116's data channels and MEM 10112. DMOVR 20410 performs actual transfer of data between IOS 10116's data channel devices and MEM 10112, and controls access between IOS 10116's data channel devices and MEM 10112. IDB 20440 and ODB 20442 are data and address buffers allowing asynchronous transfer of data between IOS 10116 and MEM 10112. That is, ODB 20442 may accept data from MEM 10112 as that data becomes available and then hold that data until an IOS 10116 data channel device, for example EBMC 20414, is ready to accept that data. IDB 20440 accepts data and MEM 10112 physical addresses from IOS 10116's data channel devices. IDB 20440 holds that data and addresses for subsequent transmission to MEM 10112 when MEM 10112 is ready to accept data and addresses. IDB 20440 may, for example, accept a burst, or sequence, of data from EBMC 20414 or single data words from NDC 20416 and subsequently provide that data to MEM 10112 in block, or four word, transfers as previously described. Similarly, ODB 20442 may accept one or more block transfers or data from ODB 20442 and subsequently provide that data to NDC 20416 as single words, or to DMID 20430 as a data burst. In addition, as previously described, a block transfer from MEM 10112 may not appear as four sequential words. In such cases, ODB 20442 accepts the four words of a block transfer as they appear on MIO Bus 10129 and assembles those words into a block comprising four sequential words for subsequent transfer to ED 10124.

Transfer of data through IDB 20440 and ODB 20442 is controlled by PRC 20444, which exchanges control signals with IOCP 20412 and has an interface, previously described, to MEM 10112 through IOMC Bus 10131.

b.b. Priority Resolution and Control 20444 (Fig. 269)

As previously described, PRC 20444 controls access between IOS 10116 data channel devices and MEM 10112. This operation is performed by means of a Ring Grant Access Generator (RGAG) within PRC 20444.

Referring to Fig. 270, a diagramic representation of PRC 20444's RGAG is shown. In general, PRC 20444's RGAG is comprised of a Ring Grant Code Generator (RGCG) 26910 and one or more data channel request comparators. In Fig. 269, PRC 20444's RGAC is shown as including ECLIPSE® Burst Multiplexer Channel Request Comparator (EBMCRC)26912, NOVA® Data Channel Request Comparator (NDCRC) 26914, Data Channel Device X Request Comparator (DCDXRC) 26916, and Data Channel Device Z Request Comparator (DCDZRC) 26918. PRC 20444's RGAG may include more or fewer request comparators as required by the number of data channel devices within a particular IOS 10116.

As indicated in Fig. 269, Request Grant Code (RGC) outputs of RGCG 26910 are connected in parallel to first inputs of EBMCRC 26912, NDCRC 26914, DCDXRC 26916, and DCDZRC 26918. Second inputs of EBMCRC 26912, NDCRC 26914, DCDXRC 26916, and DCDZRC 26918 are connected from other portions of PRC 20444 and receive indications that, respectively, EBMC 20414, NDC 20416, DCDX, or DCDZ has submitted a request for a read or write access to MEM 10112.

Request Grant Outputs (GRANT) of EBMCRC 26912, NDCRC 26914, DCDXRC 26916, and DCDZRC 26918 are in turn connected to other portions of PRC 20444 circuitry to indicate when read or write access to MEM 10112 has been granted in response to a request by a particular IOS 10116 data channel device. When indication of such a grant is provided to those other portions of PRC 20444, PRC 20444 proceeds to generate appropriate control signals to MEM 10112, through IOMC Bus 10131 as previously described, to IDB 20440 and ODB 20442, and to IOCP 20412. PRC 20444's control signals initiate that read or write request to that IOS 10116 data channel device. Grant outputs of EBMCRC 26912, NDCRC 26914, DCDXRC 26916, and DCDZRC 26918 are also provided as inputs to RGCG 26910 to indicate, as described further below, when a particular IOS 10116 has requested and been granted access to MEM 10112.

As indicated in Fig. 269, a diagramic figure above RGCG 26910, RGCG generates a repeated sequence of unique RGCs. Herein indicated as numeric digits 0 to 15. Each RGC identifies, or defines, a particular time slot during which a IOS 10116 data channel device may be granted access to MEM 10112. Certain RGCs are, effectively, assigned to particular IOS 10116 data channel devices. Each such data channel device may request access to MEM 10112 during its assigned RGC identified access slots. For example, EBMC 20414 is shown as being allowed access to MEM 10112 during those access slots identified by RGCs 0, 2, 4, 6, 8, 10, 12, and 14. NDC 20416 is indicated as being allowed access to MEM 10112 during RGC slots 3, 7, 11, and 15. DCDX is allowed access during slots 1 and 9, and DCDZ is allowed access during RGC slots 5 and 13.

As described above, RGCG generates RGCs 0 to 15 in a repetitive sequence. During occurrence of a particular RGC, each request comparator of PRC 20444's RGAG examines that RGC to determine whether its associated data channel device is allowed access during that RGC slot, and whether that associated data channel device has requested access to MEM 10112. If that associated data channel device is allowed access during that RGC slot, and has requested access, that data channel device is granted access as indicated by that request comparator's GRANT output. The request comparators GRANT output is also provided as an input to RGCG 26910 to indicate to RGCG 26910 that access has been granted during that RGC slot.

If a particular data channel device has not claimed and has not been granted access to MEM 10112

during that RGC slot, RGCG 26910 will go directly to next RGC slot. In next RGC slot, PRC 20444's RGAG again determines whether the particular data channel device allowed access during that slot has submitted a request, and will grant access if such a request has been made. If not, RGCG 26910 will again proceed directly to next RGC slot, and so on. In this manner, PRC 20444's RGAG insures that each data channel device of IOS 10116 is allowed access to MEM 10112 without undue delay. In addition, PRC 20444's RGAG prevents a single, or more than one, data channel device from monopolizing access to MEM 10112. As described above, each data channel device is allowed access to MEM 10112 at least once during a particular sequence of RGCs. At the same time, by not pausing within a particular RGC in which no request for access to MEM 10112 has occurred, PRC 20444's RGAG effectively automatically skips over those data channel devices which have not requested access to MEM 10112. PRC 20444's RGAG thereby effectively provides, within a given time interval, more frequent access to those data channel devices which are most busy. In addition, the RGCs assigned to particular IOS 10116 data channel devices may be reassigned as required to adapt a particular CS 10110 to the data input and output requirements of a particular CS 10110 configuration. That is, if EBMC 20414 is shown to require less access to MEM 10112 then NDC 20416, certain RGCs may be reassigned from EBMC 20414 to NDC 20416. Access to MEM 10112 by IOS 10116's data channel devices may thereby be optimized as required.

Having described structure and operation of IOS 10116, structure and operation of DP 10118 will be described next below.

E. Diagnostic Processor 10118 (Figs. 101, 205)

Referring to Fig. 101, as previously described, DP 10118 is interconnected with IOS 10116, MEM 10112, FU 10120, and EU 10122 through DP Bus 10138. DP 10118 is also interconnected, through DPIO Bus 10136, with the external world and in particular with DU 10134. In addition to performing diagnostic and fault monitoring and correction operations, DP 10118 operates, in part, to provide control and display functions allowing an operator to interface with CS 10110. DU 10134 may be comprised, for example, of a CRT and keyboard unit, or a teletype, and provides operators of CS 10110 with all control and display functions which are conventionally provided by a hard console, that is a console containing switches and lights. For example, DU 10134, through DP 10118, allows an operator to exercise control of CS 10110 for such purposes as system initialization and startup, execution of diagnostic processes, fault monitoring end identification, and control of execution of programs. As will be described further below, these functions are accomplished through DP 10118's interfaces with IOS 10116, MEM 10112, FU 10120, and EU 10122.

DP 10118 is a general purpose computer system, for example a NOVA® 4 computer of Data General Corporation of Westboro, Massachusetts. Interface of DP 10118 and DU 10134, and mutual operation of DP 10118 and DU 10134, will be readily apparent to one of ordinary skill in the art. DP 10118's interface and operation, with IOS 10116, MEM 10112, FU 10120, and EU 10122 will be described further next below.

DP 10118, operating as a general purpose computer programmed specificially to perform the functions described above, has, as will be described below, read and write access to registers of IOS 10116, MEM 10112, FU 10120 and EU 10122 through DP Bus 10138. DP 10118 may read data directly from and write data directly into those registers. As will be described below, these registers are data and instruction registers and are integral parts of CS 10110's circuitry during normal operation of CS 10110. Access to these registers thereby allows DP 10118 to directly control or effect operation of CS 10110. In addition, and as also will be described below, DP 10118 provides, in general, all clock signals to all portions of CS 10110 circuitry and may control operation of that circuitry through control of these clock signals.

For purposes of DP 10118 functions, CS 10110 may be regarded as subdivided into groups of functionally related elements, for example DESP 20210 in FU 10120. DP 10118 obtains access to the registers of these groups, and control of clocks therein, through scan chain circuits, as will be described next below. In general, DP 10118 is provided with one or more scan chain circuits for each major functional sub-element of CS 10110.

Referring to Fig. 205, a diagramic representation of DP 10118 and a typical DP 10118 scan chain is shown. As indicated therein, DP 10118 includes a general purpose Central Processor Unit, or computer, (DPCPU) 27010. A first interface of DPCPU 27010 is with DU 10134 through DPIO Bus 10136. DPCPU 27010 and DU 10134 exchange data and control signals through DPIO Bus 10136 in the manner to direct operations of DPCPU 27010, and to display the results of those operations through DU 10134.

Associated with DPCPU 27010 is Clock Generator (CLKG) 27012. CLKG 27012 generates, in general, all clock signals used within CS 10110.

DPCPU 27010 and CLKG 27012 are interfaced with the various scan chain circuits of CS 10110 through DP Bus 10138. As described above, CS 10110 may include one or more scan chains for each major sub-element of CS 10110. One such scan chain, for example DESP 20210 Scan Chain (DESPSC) 27014 is illustrated in Fig. 205.

Interface between DPCPU 27010 and CLKG 27012 and, for example, DESPSC 27014 is provided through DP Bus 10138. As indicated in Fig. 205, DESPSC 27014 includes Scan Chain Clock Gates (SCCG) 27016 and one or more Scan Chain Registers (SCRs) 27018 to 27024.

SCCG 27016 receives clock signals from CLKG 27012 and control signals from DPCPU 27010 through DP Bus 10138. SCCG 27016 in turn provides appropriate clock signals to the various registers and circuits

of, for example, DESP 20210. Clock control signals provided by DPCPU 27010 to SCCG 27016 control, or gate, the various clock signals to these registers and circuits of DESP 20210, thereby effectively allowing DPCPU 27010 to control of DESP 20210.

SCRs 27018 to 27024 are comprised of various registers within DESP 20210. For example, SCRs 27018 to 27024 may include the output buffer registers of AONGRF 20232, OFFGRF 20234, LENGRF 20236, output registers of OFFALU 20242 and LENALU 20252, and registers within OFFMUX 20240 and BIAS 20246. Such registers are indicated in the present description, as previously described, by arrows appended to ends of those registers, with a first arrow indicating an input and a second an output. In normal CS 10110 operations, as previously described, SCRs 27018 to 27024 operate as parallel in, parallel out buffer registers through which data and instructions are transferred. SCRs 27018 to 27024 are also capable of operating as shift registers and, as indicated in Fig. 205, are connected together to comprise a single shift register circuit having an input from DPCPU 27010 and an output to DPCPU 27010. Control inputs to SCRs 27018 to 27024 from DPCPU 27010 control operation of SCRs 27018 to 27024, that is whether these registers shall operate as parallel in, parallel out registers, or as shift registers of DESPSC 27014's scan chain. The shift register scan chain comprising SCRs 27018 to 27024 allows DPCPU 27010 to read the contents of SCRs 27018 to 27024 by shifting the content of these registers into DPCPU 27010. Conversely, DPCPU 27010 may write into SCRs 27018 to 27024 by shifting information generated by DPCPU 27010 from DPCPU 27010 and through the shift register scan chain to selected locations within SCRs 27018 to 27024.

Scan chain clock generator circuits and scan chain registers of each scan chain circuit within CS 10110 thereby allow DP 10118 to control operation of each major sub-element of CS 10110. For example, to read information from the scan chain registers therein, and to write information into those scan chain registers as required for diagnostic, monitoring, and control functions.

Having described structure and operation of each major element of CS 10110, including MEM 10112, FU 10120, EU 10122, IOS 10116, and DP 10118, certain operations of, in particular, FU 10120 will be described further next below. The following descriptions will further disclose operational features of JP 10114, and in particular FU 10120, by describing in greater detail certain operations therein by further describing microcode control of JP 10114.

F. CS 10110 Micromachine Structure and Operation (Figs. 270—274)

a. Introduction

The preceding descriptions have presented the hardware structures and operation of FU 10120 and EU 10122. The following description will describe how devices in FU 10120, and certain EU 10122 devices, function together as a microprogrammable computer, henceforth termed the FU micromachine. The FU micromachine performs two tasks: it interprets SINs, and it responds to certain signals generated by devices in FU 10120, EU 10122, MEM 10112, and IOS 10116. The signals to which the FU micromachine responds are termed Event signals. In terms of structure and operation, the FU micromachine is characterized by the following:

- Registers and ALUs specialized for the handling of logical descriptors.
- Registers organized as stacks for invocations of microroutines (microinstruction sequences).
- Mechanisms allowing microroutine invocations by means of event signals from hardware.
- Mechanisms which allow an invoked microroutine to return either to the microinstruction following the one which resulted in the invocation or to the microinstruction which resulted in the invocation.
- Mechanisms which allow the contents of stack registers to be transferred to MEM 10112, thereby creating a virtual microstack of limitless size.
- Mechanisms which guarantee response to an event signal within a predictable length of time.
- The division of the devices comprising the micromachine into two groups: those devices which may be used by all microcode and those which may be used only by KOS (Kernel Operating System, previously described) microcode.

These devices and mechanisms allow the FU micromachine to be used in two ways: as a virtual micromachine and as a monitor micromachine. Both kinds of micromachine use the same devices in FU 10120, but perform different functions and have different logical properties. In the following discussion, when the FU micromachine is being used as a virtual micromachine, it is said to be in virtual mode, and when it is being used as a monitor micromachine, it is said to be in monitor mode. Both modes are introduced here and explained in detail later.

When the FU micromachine is being used in virtual mode, it has the following properties:

- It runs on an essentially infinite micromachine stack belonging to a Process 610.
- It can respond to any number of event signals in the M0 cycle (state) of a single microinstruction.
- A page fault may occur on the invocation of any microroutine or on return from any microroutine.
- When the FU micromachine is in virtual mode, any microroutine may not run to completion, i.e., complete its execution in a predictable length of time, or complete it at all.
- It is executing a Process 610.

The last four properties are consequences of the first: Event signals result in invocations, and since the micromachine stack is infinite, there is no limit to the number of invocations. The infinite micromachine stack is realized by placing micromachine stack frames on Secure Stack 10336 belonging to a Process 610,

and the virtual micromachine therefore always runs on a micromachine stack belonging to some Process 610. Furthermore, if the invocation of a microroutine or a return from a microroutine requires micromachine frames to be transferred from Secure Stack 10336 to the FU micromachine, a page fault may result, and Process 610 which is executing the microroutine may be removed from JP 10114, thereby making the time required to execute the microroutine unpredictable. Indeed, if process 610 is stopped or killed, the execution of the microroutine may never finish. As will be seen in descriptions below, the Virtual Processor 612 is the means by which the virtual micromachine gains access to a Process 610's micromachine stack.

When in monitor mode, the FU micromachine has the following properties:

- It has a micromachine stack of fixed size, the stack is always available to the FU micromachine, and it is not associated with a Process 610.
- It can respond to only a fixed number of events during the M0 cycle of a single microinstruction.
- In monitor mode, invocation of a microroutine or return from a microroutine will not cause a page fault.
- Microroutines executing on the FU micromachine when the micromachine is in monitor mode are guaranteed to run to completion unless they themselves perform an action which causes them to give up JP 10114.
- Microroutines executing in monitor mode need not be performing functions for a Process 610.

Again, the remaining properties are consequences of the first: because the monitor micromachine's stack is of fixed size, the number of events to which the monitor micromachine can respond is limited; furthermore, since the stack is always directly accessible to the micromachine, microroutine invocations and returns will not cause page faults, and microroutines running in monitor mode will run to completion unless they themselves perform an action which causes them to give up JP 10114. Finally, the monitor micromachine's stack is not associated with a Process 610's Secure Stack 10336, and therefore, the monitor micromachine can both execute functions for Processes 610 and execute functions (which are related to no Process 610, for example,) the binding and removal of Virtual Processors 612 from JP 10114.

The description which follows first gives an overview of the devices which make up the micromachine, continues with descriptions of invocations on the micromachine and micromachine programming, and concludes with detailed discussions of the virtual and monitor modes and an overview of the relationship between the micromachine and CS 10110 subsystems. The manner in which the micromachine performs specific operations such as SIN parsing, Name resolution, or address translation may be found in previous descriptions of CS 10110 components which the micromachine uses to perform the operations.

b. Overview of Device Comprising FU Micromachine (Fig. 270)

Fig. 270 presents an overview of the devices comprising the micromachine. Fig. 270 is based on Fig. 201, but has been simplified to improve the clarity of the discussion. Devices and subdivisions of the micromachine which appear in Fig. 201 have the numbers given them in that figure. When a device in Fig. 270 appears in two subdivisions, it is shared by those subdivisions.

Fig. 270 has four main subdivisions. Three of them are from Fig. 201: FUCTL 20214, which contains the devices used to select the next microinstruction to be executed by the micromachine, DESP 20210, which contains stack and global registers and ALUs for descriptor processing; and MEMINT 20212, which contains the devices which translate Names into logical descriptors and logical descriptors into physical descriptors. The fourth subdivision, EU Interface 27007, represents those portions of EU 10122 which may be manipulated by FU 10120 microcode.

Fig. 270 further subdivides FUCTL 20214 and MEMINT 20212. FUCTL 20214 has four subdivisions:

- I-Stream Reader 27001, which contains the devices used to obtain SINs and parse them into SOPs and Names.
- SOP Decoder 27003, which translates SOPs into locations in FU microcode (FUSITT 11012), and in some cases EU microcode (EUSITT 20344), which contain the microcode that performs the corresponding SINs.
- Microcode Addressing 27013, which determines the location of the next microinstruction to be executed in FUSITT 11012.
- Register Addressing 27011, which contains devices which generate addresses for GRF 10354 registers.

MEMINT 20212 also has three subdivisions:

- Name Translation Unit 27015, which contains devices which accelerate the translation of Names into logical descriptors.
- Memory Reference Unit 27017, which contains devices which accelerate the translation of logical descriptors into physical descriptors.
- Protection Unit 27019, which contains devices which accelerate primitive access checks on memory references made with logical descriptors.

Fig. 270 also simplifies the bus structure of Fig. 202 by combining LENGTH Bus 20226, OFFSET Bus 20228, and AONR Bus 20230 into a single structure, Descriptor Bus (DB) 27021. In addition, internal bus connections have been reduced to those necessary for explaining the logical operation of the

micromachine. The following discussion first describes those devices used by most microcode executing on FU 10120, and then describes devices used to perform special functions, such as Name translation or protection checking.

1. Devices used by Most Microcode

The subdivisions of the micromachine which contain devices used by most microcode are Microcode Addressing 27013, Register Addressing 27011, DESP 20210, and EU Interface 27007. In addition, most microcode uses MOD Bus 10144, JPD Bus 10142, and DB Bus 27021. The discussion begins with the buses and then describes the other devices in the above order.

a.a. MOD Bus 10144, JPD Bus 10142, and DB Bus 27021

MOD Bus 10144 is the only path by which data may be obtained from MEM 10112. Data on MOD Bus 10144 may have as its destination Instruction Stream Reader 27001, DESP 20210, or EU Interface 27007. In the first case, the data on MOD Bus 10144 consists of SINs; in the second, it is data to be processed by FU 10120, and in the third, it is data to be processed by EU 10122. In the present embodiment, data to be processed by FU 10120 is generally data which is destined for internal use in FU 10120, for example in Name Cache 10226. Data to be processed by EU 10122 is generally operands represented by Names in SINs.

JPD Bus 10142 has two uses: it is the path by which data returns to MEM 10112 after it has been processed by JP 10114, and it is the path by which data other than logical descriptors moves between the subdivisions of the micromachine. For example, when CS 10110 is initialized, the microinstructions which are loaded into FUSITT 11012 are transferred from MEM 10112 to DESP 20210 via MOD Bus 10144, and from DESP 20210 to FUSITT 11012 via JPD Bus 10142.

DB 27021 is the path by which logical descriptors are transferred in the micromachine. DB 27021 connects Name Translation Unit 27015, DESP 20210, Protection Unit 27019, and Memory Reference Unit 27017. Typically, a logical descriptor is obtained from Name Translation Unit 27015, placed in a register in DESP 20210, and then presented to Protection Unit 27019 and Memory Reference Unit 27017 whenever a reference is made using a logical descriptor. However, DB 27021 is also used to transmit cache entries fabricated in DESP 20210 to ATU 10228, Name Cache 10226 and Protection Cache 10234.

b.b. Microcode Addressing

As discussed here, microcode addressing is comprised of the following devices: Timers 20296, Event Logic 20284, RCWS 10358, BRCASE 20278, mPC 20276, MCW0 20292, MCW1 20290, SITTNAS 20286, and FUSITT 11012. All of these devices have already been described in detail, and they are discussed here only as they affect microcode addressing. Other devices contained in Fig. 202, State Registers 20294, Repeat Counter 20280, and PNREG 20282 are not directly relevant to microcode addressing, and are not discussed here.

As has already been described in detail, devices in Microcode Addressing 27013 are loaded from JPD Bus 10142. The microcode addresses provided by these devices and by FUSDT 11010 are transmitted among the devices and to FUSITT 11012 by CSADR Bus 20204. There are six ways in which the next microcode address may be obtained:

- Most commonly, the value in mPC 20276 is incremented, by 1 by a special ALU in mPC 20276, thus yielding the address of the microinstruction following the current microinstruction.
- If a microinstruction specifies a call to a microroutine or a branch, the microinstruction contains a literal which an ALU in BRCASE 20278 adds to the value in mPC 20276 to obtain the location of the next microinstruction.
- If a microinstruction specifies the use of a case value to calculate the location of the next microinstruction, BRCASE 20278 adds a value calculated by DESP 20210 to the value in mPC 20716. The value calculated by DESP 20210 may be obtained from a field of a logical descriptor, thus allowing the micromachine to branch to different locations in microcode on the basis of type information contained in the logical descriptor. On return from an invocation of a microroutine, the location at which execution of the microroutine in which the invocation occurred is to continue is obtained from RCWS 10358.
- At the beginning of the execution of an SIN, the location at which the microcode for the SIN begins is obtained from the SIN's SOP by means of FUSDT 11010.
- Certain hardware signals cause invocations of microroutines. There are two classes of such signals: Event signals, which Event Logic 20284 transforms into invocations of certain microroutines, and JAM signals, which are translated directly into locations in microcode.

The addresses obtained as described above are transmitted to SITTNAS 20286, which selects one of the addresses as the location of the next microinstruction to be executed and transmits the location to FUSITT 11012. As the location is transmitted to FUSITT 11012, it is also stored in mPC 20276. All addresses except those for Jams are tranferred to SITTNAS 20286 via CSADR Bus 20204. Addresses obtained from

JAM signals are transferred by separate lines to SITTNAS 20286.

As will be explained in detail below, microroutine calls and returns also involve pushing and popping micromachine stack frames and saving state contained in MCW1 20290.

Register Addressing 27011 controls access to micromachine registers contained in GRF 10354. As explained in detail below, GRF 10354 contains both registers used for the micromachine stack and global registers, that is, registers that are always accessible to all microroutines. The registers are grouped in frames, and individual registers are addressed by frame number and register number. Register Addressing 27011 allows addressing of any frame and register in the GRs 10360 of GRF 10354, but allows addressing of registers in only three frames of the SR's 10362: the current (top) frame, the previous frame (i.e., the frame preceding the top frame), and the bottom frame, that is, the lowest frame in a virtual micromachine stack which is still contained in GRF 10354. The values provided by Register Addressing 27011 are stored in MCW0 20292. As will be explained in the discussion of microroutine invocations which follows, current and previous are incremented on each invocation and decremented on each return.

c.c. Description Processor 20210 (Fig. 271)

DESP 20210 is a set of devices for storing and processing logical descriptors. The internal structure of DESP 20210's processing devices has already been explained in detail; here, the discussion deals primarily with the structure and contents of GRF 10354. In a present embodiment of CS 10110, GRF 10354 contains 256 registers. Each register may contain a single logical descriptor. Fig. 271 illustrates a Logical Descriptor 27116 in detail. In a present embodiment of CS 10110, a Logical Descriptor 27116 has four main fields:

- RS Field 27101, which contains various flags which are explained in detail below.
- AON Field 27111, which contains the AON portion of the address of the data item represented by the Logical Descriptor 27116.
- OFF Field 27113, which contains the offset portion of the address of the data item represented by Logical Descriptor 27116.
- LEN Field 27115, which contains the length of the data item represented by the Logical Descriptor 27116.

RS Field 27101 has subfields as follows:

- RTD Field 27103 and WTD Field 27105 may be set by microcode to disable certain Event signals provided for debuggers by CS 10110. For details, see a following description of debugging aids in CS 10110.
- FIU Field 27107 contains two bits. The fields are set from information in the Name Table Entry used to construct the Logical Descriptor 27116. The bits determine how the data specified by the Logical Descriptor 27116 is to be justified and filled when it is fetched from MEM 10112.
- TYPE Field 27109's four bits are also obtained from the Name Table Entry used to construct the Logical Descriptor 27116. The field's settings vary from S-Language to S-Language, and are used to communicate S-Language-specific type information to the S-Language's S-Interpreter microcode.

The four fields of a Logical Descriptor 27116 are contained in three separately-accessible fields in a GRF 10354 register: one containing RS Field 27101 and AON Field 27111, one containing OFF Field 27113, and one containing LEN Field 27115. In addition, each GRF 10354 register may be accessed as a whole. GRF 10354 is further subdivided into 32 frames of eight registers each. An individual GRF 10354 register is addressed by means of its frame number and its register number within the frame. In a present embodiment of CS 10110, half of the frames in GRF 10354 belong to SR's 10362 and are used for micromachine stacks, and half belong to GRs 10360 for storing "global information". In SR's 10362, each GRF 10354 frame contains information belonging to a single invocation of a microroutine. As previously explained, Register Addressing 27011 allows addressing of only three GRF 10354 frames in SR's tack 10362, the current top frame in the stack, the previous frame, and the bottom frame. Registers are accessed by specifying one of these three frames and a register number.

The global information contained in GRs 10360, is information which is not connected with a single invocation. There are three broad categories of global information:

- Information belonging to Process 610 whose Virtual Processor 612 is currently bound to JP 10114. Included in this information are the current values of Process 610's ABPs and the pointers which KOS uses to manage Process 610's stacks.
- Information required for the operation of KOS. Included in this information are such items as pointers to KOS data bases which occupy fixed locations in MEM 10112.
- Constants, that is, fixed values required for certain frequently performed operations in FU 10120.

Remaining registers are available to microprogrammers as temporary storage areas for data which cannot be stored in a microroutine's stack frame. For example, data which is shared by several microroutines may best be placed in a GR 10360. Addressing of registers in the GRs 10360 of GRF 10354 requires two values: a value of 0 through 15 to specify the frame and a value of 0 through 7 to specify the register in the frame.

As previously discussed in detail, each of the three components AONP 20216, OFFP 20218, and LENP 20220 of DESP 20210 also contains ALUs, registers, and logic which allows operations to be performed on individual fields of GRF 10354 registers. In particular, OFFP 20218 contains OFFALU 20242, which may be

used as a general purpose 32 bit arithmetic and logical unit. OFFALU 20242 may further serve as a source and destination for JPD Bus 10142, the offset portion of DB 27021, and NAME Bus 20224, and as a destination for MOD Bus 10144. Consequently, OFFALU 20242 may be used to perform operations on data on these buses and to transfer data from one bus to another. For example, when an SIN contains a literal value used in address calculation, the literal value is transferred via NAME Bus 20224 to OFFALU 20242, operated on, and output via the offset portion of DB 27021.

d.d. EU 10122 Interface

FU 10120 specifies what operation EU 10122 is to perform, what operands it is to perform it on, and when it is finished, what is to be done with the operands. FU 10120 can use two devices in EU 10122 as destinations for data, and one device as a source for data. The destinations are COMQ 20342 and OPB 20322. COMQ 20342 receives the location in EUSITT 20344 of the microcode which is to perform the operation desired by the FU 10120. COMQ 20342 may receive the location in microcode either from an FU 10120 microroutine or from an SIN's SOP. In the first case, the location is transferred via JPD Bus 10142, and in the second, it is obtained from EUSDT 20266 and transferred via EUDIS Bus 20206. OPB 20322 receives the operands upon which the operation is to be performed. If the operands come directly from MEM 10112, they are transferred to OPB 20322 via MOD Bus 10144; if they come from registers or devices in FU 10120, they are transferred via JPD Bus 10142.

Result Register 27013 is a source for data. After EU 10122 has completed an operation, FU 10120 obtains the result from Result Register 27013. FU 10120 may then place the result in MEM 10112 or in any device accessible from JPD Bus 10142.

2. Specialized Micromachine Devices

Each of the groups of specialized devices serves one of CS 10110's subsystems. I-Stream Reader 27001 is part of the S-Interpreter subsystem, Name Translation Unit 27015 is part of the Name Interpreter subsystem, Memory Reference Unit 27017 is part of the Virtual Memory Management System, and Protection Unit 27019 is part of the Access Control System. Here, these devices are explained only in the context of the micromachine; for a complete understanding of their functions within the subsystems to which they belong, see previous descriptions of the subsystems.

a.a. I-Stream Reader 27001

I-Stream Reader 27001 reads and parses a stream of SINs (termed the I-Stream) from a Procedure Object 604, 606, 608. The I-Stream consists of SOPs (operation codes), Names, and literals. As previously mentioned, in a present embodiment of CS 10110, the I-Stream read from a given Procedure 602 has a fixed format: the SOPs are 8 bits long and the Names and literals all have a single length. Depending on the procedure, the length may be 8, 12, or 16 bits. I-Stream Reader 27001 parses the I-Stream by breaking it up into its constituent SOPs and Names and passing the SOPs and Names to appropriate parts of the micromachine. I-Stream Reader 27001 contains two groups of devices:

— PC Values 27006, which is made up of three registers which contain locations in the I-Stream. When added to ABP PBP, the values contained in these registers specify locations in Procedure Object 901 containing the Procedure 602 being executed. CPC 20270 contains the location of the SOP or Name currently being interpreted; IPC 20272 contains the location of the beginning of the SIN currently being executed; EPC 20274, finally, is of interest only at the beginning of the execution of an SIN; at that time, it contains the location of the last SIN to be executed.

— Parsing Unit 27005, which is made up of INSTB 20262, PARSER 20264, and PREF 20260. The micromachine uses PREF 20260 to create Logical Descriptors 27116 for the I-Stream, which are then placed on DB Bus 27021 and used in logical memory references. The data returned from these references is placed in INSTB 20262, and parsed by PARSER 20264.

SOPs, Names, and literals obtained by PARSER 20264 are placed on NAME Bus 20224, which connects PARSER 20264, SOP Decoder 27003, Name Translation Unit 27015, and OFFALU 20242.

b.b. SOP Decoder 27003

SOP Decoder 27003 decodes SOPs into locations in FU 10120 and EU 10122 microcode. SOP Decoder 27003 comprises FUSDT 11010, EUSDT 20266, Dialect Register (RDIAL) 24212, and LOPDCODE 24210. FUSDT 11010 are further comprised of FUDISP 24218 and FALG 24220. The manner in which these devices translate SOPs contained in SINs into locations in FUSITT 11012 and EUSITT 20344 has been previously described.

c.c. Name Translation Unit 27015

Name Translation Unit 27015 accelerates the translation of Names into Logical Descriptors 27116. This operation is termed name resolution. It is comprised of two components: NC 10226 and Name Trap 20254. NC 10226 contains copies of information from a Procedure Object 604's Name Table 10350, and thereby makes it possible to translate Names into Logical Descriptors 27116 without referring to Name Table 10350. When a Name is presented to Name Translation Unit 27015, it is latched into Name Trap 20254 for later use by Name Translation Unit 27015 if required. As will be explained in detail later, in the present embodiment,

Name translation always begins with the presentation of a Name to NC 10226. If the Name has already been translated, the information required to construct its Logical Descriptor 27116 may be contained in NC 10226. If there is no information for the Name in NC 10226, Name Resolution Microcode obtains the Name from Name Trap 20254, uses information from Name Table 10350 for the procedure being executed to translate the Name, places the required information in NC 10226, and attempts the translation again. When the translation succeeds, a Logical Descriptor 27116 corresponding to the Name is produced from the information in Name Cache 10115, placed on DB Bus 27021, and loaded into a GRF 10354 register.

d.d. Memory Reference Unit 27017

Memory Reference Unit 27017 performs memory references using Logical Descriptors 27116. Memory Reference Unit 27017 receives a command for MEM 10112 and a Logical Descriptor 27116 describing the data upon which the command is to be performed. In the case of a write operation, Memory Reference Unit 27017 also receives the data being written via JPD Bus 10142. Memory Reference Unit 27017 translates Logical Descriptor 27116 to a physical descriptor and transfers the physical descriptor and the command to MEM 10112 via PD Bus 10146. A Memory Reference Unit 27017 has four components: ATU 10228, which contains copies of information from KOS virtual memory management system tables, and thereby accelerates logical-to-physical descriptor translation; Descriptor Trap 20256, which traps Logical Descriptors 27116, Command Trap 27018, which traps memory commands; and Data Trap 20258, which traps data on write operations. When a logical memory reference is made, a Logical Descriptor 27116 is presented via DB Bus 27021 to ATU 10228, and at the same time, Logical Descriptor 27116 and the memory command are trapped in Descriptor Trap 20256 and Command Trap 27018. On write operations, the data to be written is trapped in Data Trap 20258. If the information needed to form the physical descriptor is present in ATU 10228, the physical descriptor is transferred to MEM 10112 via PD Bus 10146. If the information needed to form the physical descriptor is not present in ATU 10228, an Event Signal from ATU 10228 invokes a microroutine which retrieves Logical Descriptor 27116 from Descriptor Trap 20256 and uses information contained in KOS virtual memory management system tables to make an entry in ATU 10228 for Logical Descriptor 27116. When the microroutine returns, the logical memory reference is repeated using Logical Descriptor 27116 from Descriptor Trap 20256, the memory command from Command Trap 27018, and on write operations, the data in Data Trap 20258. As will be described in detail in the discussion of virtual memory management, if the data referenced by a logical memory reference is not present in MEM 10112, the logical memory reference causes a page fault.

e.e. The Protection Unit 27019

On each logical memory reference, Protection Unit 27019 checks whether the subject making the reference has access rights which allow it to perform the action specified by the memory command on the object being referenced. If the subject does not have the required access rights, a signal from Protection Unit 27019 causes MEM 10112 to abort the logical memory reference. Protection Unit 27019 consists of Protection Cache 10234, which contains copies of information from KOS Access Control System tables, and thereby speeds up protection checking, and shares Descriptor Trap 20256, Command Trap 27018, and Data Trap 20258 with Memory Reference Unit 27017. When a logical memory reference is made, the AON and offset portions of the logical descriptor are presented to Protection Cache 10234. If Protection Cache 10234 contains protection information for the object specified by the AON and offset and the subject performing the memory reference has the required access, the memory reference may continue; if Protection Cache 10234 contains protection information and the subject does not have the required access, a signal from Protection Cache 10234 aborts the memory reference. If Protection Cache 10234 does not contain the required access information, a signal from Protection Cache 10234 aborts the memory reference and invokes a microroutine which obtains the access information from KOS Access Control System tables and places it in Protection Cache 10234. When Protection Cache 10234 is ready, the memory access is repeated, using the logical descriptor from Descriptor Trap 20256, the memory command from Command Trap 27018, and in the case of write operations, the data in Data Trap 20258.

f.f. KOS Micromachine Devices

As mentioned in the above introduction to the micromachine, the devices making up the micromachine may be divided into two classes: those which any microcode written for the micromachine may manipulate, and those which may be manipulated exclusively by KOS microcode. The latter class consists of certain registers in GRs 10360 of GRF 10354, the bottom frame of the portion of the virtual micromachine stack in the stack portion (Stack Registers 10362) of GRF 10354, and the devices contained in Protection Unit 27019 and Memory Reference Unit 27017. Because Protection Unit 27019 and Memory Reference Unit 27017 may be manipulated only by KOS microcode, non-KOS microcode may not use Descriptor Trap 20256 or Command Trap 27018 as a source or destination, may not load or invalidate registers in ATU 10228 or Protection Cache 10234, and may not perform physical memory references, i.e., memory references which place physical descriptors directly on PD Bus 10146, instead of presenting logical

descriptors to Memory Reference Unit 27017 and Protection Unit 27019. Similarly, non-KOS microcode may not specify KOS registers in the GRs 10360 of GRF 10354 or the bottom frame of the stack portion of GRF 10354 when addressing GRF 10354 registers. Further, in embodiments allowing dynamic loading of FUSITT 11012, only KOS microcode may manipulate the devices provided for dynamic loading.

In a present embodiment of CS 10110, the distinction between KOS devices and registers and devices and registers accessible to all microprograms is maintained by the microbinder. The microbinder checks all microcode for microinstructions which manipulate devices in Protection Unit 27019, or Memory Reference Unit 27017, or which address GRF 10354 registers reserved for KOS use. However, it is characteristic of the micromachine that KOS devices are logically and physically separate from devices accessible to all microprograms and, consequently, other embodiments of CS 10110 may use hardware devices to prevent non-KOS microprograms from manipulating KOS devices.

c. Micromachine Stacks and Microroutine Calls and Returns (Figs. 272, 273)

1. Micromachine Stacks (Fig. 272)

As previously mentioned, the FU micromachine is a stack micromachine The properties of the FU micromachine's stack depends on whether the FU micromachine is in virtual or monitor mode. In virtual mode, the micromachine stack is of essentially unlimited size; if it contains more frames than allowed for inside FU 10120, the top frames are in GRF 10354 and the remaining frames are in Secure Stack 10336 belonging to Process 610 being executed by the FU micromachine. In the following, the virtual mode micromachine stack is termed the virtual micromachine stack. In monitor mode, the micromachine stack consists of a fixed amount of storage; in a present embodiment of CS 10110, the monitor mode micromachine stack is completely contained in the stack portion, SRs 10362, of GRF 10354; in other embodiments of CS 10110, part or all of the monitor mode micromachine stack may be contained in an area of MEM 10112 which has a fixed size and a fixed location known to the monitor micromachine. In yet other embodiments of CS 10110, monitor mode micromachine stack may be of flexible depth in a manner similar to the virtual micromachine stack. In either mode, microroutines other than certain KOS microroutines which execute state save and restore operations may access only two frames of GRF 10354 stack: the frame upon which the microroutine is executing, called the current frame, and the frame upon which the microroutine that invoked that microroutine executed, called the previous frame. KOS microroutines which execute state save and restore operations may in addition access the bottom frame of that portion of the virtual micromachine stack which is contained in GRF 10354.

Fig. 272 illustrates stacks for the FU micromachine. Those portions of the micromachine stack which are contained in the FU are contained in SR's 10362 (of GRF 10354) and in RCWS 10358. Each register of RCWS 10358 is permanently associated with a GRF frame in SRs 10362 of GRF 10354, and the RCWS 10358 register and the GRF frame together may contain one frame of a micromachine stack. As previously describe, each register of GRF 10354 contains three fields: one for an AON and other information, one for an offset, and one for a length. As illustrated in Fig. 251, each register in RCWS 10358 contains four fields:

- A one bit field which retains the value of the Condition Code register in MCW1 20290 at the time that the invocation which created the next frame occurred.
- A field indicating what Event Signals were pending at the time that the invocation to which the RCWS register belongs invoked another microroutine.
- A flag indicating whether the microinstruction being executed when the invocation occurred was the first microinstruction in an SIN.
- The address at which the execution of the invoking microroutine is to continue.

The uses of these fields will become apparent in the ensuing discussion.

The space available for micromachine stacks in SRs 10362 and RCWS 10358 is divided into two parts: Frames 27205 reserved for MOS 10370 and Frames 27206 available for the MIS 27203. Frames 27206 may contain no MIS Frames 27203, or be partially or completely occupied by MIS Frames 27203. Space which contains no MIS Frames is Free Frames 27207. The size of the space reserved for Monitor Micromachine Stack Frames 27205 is fixed, and Spaces 27203, 27205, and 27207 always come in the specified order. Register Addressing 27011 handles addressing in Stack Portion 27201 of GRF 10354 and RCWS 10358 in such fashion that the values for the locations of current, previous, and bottom frames specifying registers in RCWS 10358 or frames in Stack Portion 27201 automatically "wrap around" when they are incremented beyond the largest index value allowed by the sizes of the registers or decremented below the smallest index value. Thus, though Spaces 27203, 27205, and 27207 always have the same relative order, their GRF 10354 frames and RCWS registers may be located anywhere in Stack Portion 27201 and RCWS 10358.

2. Microroutine Invocations and Returns

In CS 10110, microroutines may be invoked by other microroutines or by signals from CS 10110 hardware. The methods of invocation aside, microroutine invocations and returns resemble invocations of and returns from procedures written in high-level languages. In the following, the general principles of microroutine invocations and returns are discussed, and thereafter, the specific methods by which microroutines may be invoked in CS 10110. The differences between invocations in monitor mode and

invocations in virtual mode are explained in the detailed discussions of the two modes.

The microroutine which is currently being executed runs on the frame specified by Current Pointer 27215. When an invocation occurs, either because the executing microroutine performs a call, or because a signal which causes invocations has occurred, JP 10114 hardware does three things:

- It stores state information for the invoking microroutine in the RCWS 10358 register associated with the current frame. The state information includes the location at which execution of the invoking microroutine will resume, as well as other state information.
- It increments Current Pointer 27215 and Previous Pointer 27213, thereby providing a frame for the new invocation.
- It begins executing the first instruction of the newly invoked microroutine.

Because the newly-invoked microroutine can access registers of the invoking microroutine's frame, the invoking microroutine can pass "arguments" to the invoked microroutine by placing values in registers in its frame used by the invoked microroutine. However, the invoking microroutine cannot specify which registers contain "arguments" on an invocation, so the invoked microroutine must know which registers of the previous frame are used by the invoking microroutine. Since the only "arguments" which a microroutine has access to are those in the previous frame, a microroutine can pass arguments which it received from its invoker to a microroutine which it invokes only by copying the arguments from its invoker's frame to its own frame, which then becomes the newly-invoked routine's previous frame.

The return is the reverse of the above: Current Pointer 27215 and Previous Pointer 27213 are decremented, thereby "popping off" the finished invocation's frame and returning to the invoker's frame. The invoker then resumes execution at the location specified in the RCWS 10358 register and using the state saved in the RCWS 10358. The saved state includes the value of the Condition Code in MCW1 20290 at the time of the invocation and flags indicating various pending Events. The Condition Code field in MCW1 20290 is set to the saved value, and the pending event flags may cause Events to occur as described in detail below.

3. Means of Invoking Microroutines

In the micromachine, invocations may be produced either by commands in microinstructions or by hardware signals. In the following, invocations produced by commands in microinstructions are termed Calls, while those produced by hardware signals are termed Event invocations and Jams. Invocations are further distinguished from each other by the locations to which they return. Calls and Jams return to the microinstruction following the microinstruction in which the invocation occurs; Event invocations return to that microinstruction, which is then repeated.

In terms of implementation, the different return locations are a consequence of the point in the micromachine cycle at which Calls, Jams, and Event invocations save a return location and transfer control to the called routine. With Calls and Jams, these operations are performed in the M1 cycle; with Event invocations, on the other hand, the Event signal during the M0 cycle causes the M0 cycle to be followed by a MA cycle instead of the M1 cycle, and the operations are performed in the MA cycle. In the M1 cycle, the value in mPC 20276 is incremented; in the MA cycle, it is not. Consequently, the return value saved in RCWS 10358 on a Call or Jam is the incremented value of mPC 20276, while the return value saved on an Event invocation is the unincremented value of mPC 20276. The following discussion will deal first with Calls and Jams, and then with Event invocations.

A Call command in a microinstruction contains a literal value which specifies the offset from the microinstruction containing the Call at which execution is to continue after the Call. When the microinstruction with the Call command is executed in micromachine cycle M1, BRCASE 20278 adds the offset contained in the command to the current value of mPC 20276 in order to obtain the location of the invoked microroutine and sets SITTNAS 20286 to select the location provided by BRCASE 20278 as the location of the next microinstruction. Then the Call command increments mPC 20276 and stores the incremented value of mPC 20276 in the RCWS 10358 register associated with the current frame in SRs 10362 and increments Current Pointer 27215 and Previous Pointer 27213 to provide a new frame in SRs 10362. The Jam works exactly like the Call, except that a hardware signal during micromachine cycle M1 causes the actions associated with the invocation to occur and provides the location of the invoked microroutine directly to SITTNAS 20286.

With Events, Event Logic 20284 causes an invocation to occur during cycle M0 and provides the location of the invoked microroutine via CSADR 20299. Since the Event occurs during cycle M0, the location stored in RCWS 10358 is the unincremented value of mPC 20276, and SITTNAS 20286 selects the location provided by Event Logic 20284 as the location of the next microinstruction. Since the return from the Event causes the microinstruction during which the Event occurred to be re-executed, the microinstruction and the microroutine to which it belongs may be said to be "unaware" of the Event's occurrence. The only difference between the execution of a microinstruction during which an Event occurs and the execution of the same microinstruction without the Event is the length of time required for the execution.

4. Occurrence of Event Invocations (Fig. 273)

As described previously, Event invocations are produced by Event Logic 20284. The location in microcode to which Event Logic 20284 transfers control is determined by the following:

— The operation being commenced by FU 10120. Certain Event invocations may occur only at the beginning of certain FU 10120 operations.
— The state of Event signal lines from hardware and internal registers in Event Logic 20284.
— The state of certain registers visible via MCW1 20290. Some of these registers enable Events and others mask Events. Of the registers which enable Events, some are set by Event signals and others by the microprogram.
— On returns from invocations of microroutines, the settings of certain bits in the RCWS 10358 register belonging to the micromachine frame for the invocation that is being returned to.

Microprograms may use these mechanisms to disable Event signals and to delay an Event invocation from an Event signal for a single microinstruction or an indefinite period, and FU 10120 uses them to automatically delay Event invocations resulting from certain Event signals. Using traditional programming terminology, the mechanisms allow a differential masking of Event signals. An Event signal may be explicitly masked for a single microinstruction, it may be masked for a sequence of microinstructions; it may be automatically masked until a certain operation occurs, or it may be automatically masked for a certain maximum length of time. Event signals which occur while they are masked are not lost. In some cases, the Event signal continues until it is serviced; in others, a register is set to retain the fact that the Event signal occurred. When the Event signal is unmasked, the set register causes the Event signal to reoccur. In some cases, finally, the Event signal is not retained, but recurs when the microinstruction which caused it is repeated.

In the following, the relationship between FU 10120 operations and Event signals is first presented, and then a detailed discussion of the enabling registers in MCW1 20290 and of the bits in RCWS 10358 registers which control Event invocations.

FU 10120 allows Event invocations resulting from Event signals to be inhibited for a single microinstruction; it also delays certain Event invocations for certain Event signals until the first microinstruction of an SIN. Other Event signals occur only at the beginning of an SIN, at the beginning of a Namespace Resolve or Evaluate operation, or at the beginning of a logical memory reference.

Event invocations may be delayed for a single microinstruction by setting a field of the microinstruction itself. Setting this field delays almost all Event invocations, and thereby guarantees that an Event invocation will not occur during the microinstruction's M0 cycle.

Event signals relating to debugging occur at the beginnings of certain micromachine operations. Such Event signals are called Trace Event signals. As will be explained in detail, in the discussion of the debugger, Trace Event signals can occur on the first microinstruction of an SIN, at the beginning of an Evaluate or Resolve operation, at the beginning of a logical memory reference, or at the beginning of a microinstruction. IPM interrupt signals and Interval Timer Overflow Event signals are automatically masked until the beginning of the next SIN or until a maximum amount of time has elapsed, which ever occurs first. The mechanisms involved here are explained in detail in the discussion of interrupt handling in the FU 10120 micromachine.

Turning now to the registers used to mask and enable Event signals, Fig. 273 is a representation of the masking and enabling registers in MCW1 20290 and of the field in RCWS 10358 registers which controls Event invocations. Beginning with the registers in MCW1 20290, there are three registers which control Event invocations: Event Mask Register (EM) 27301, Events Pending Register (EP) 27309, and Trace Enable Register (TE) 27319. Bits in EM 27301 mask certain Event signals as long as they are set; bits in EP Register 27309 record the occurrence of certain Event signals while they are masked; when bits in TE Register 27319 are set, Trace Event signals occur before certain FU 10120 operations.

EM 27301 contains three one bit fields: Asynchronous Mask Field 27303, Monitor Mask Field 27305, and Trace Event Mask Field 27307. As explained in detail in the discussion of FU 10120 hardware, these bits establish a hierarchy of Event masks. If Asynchronous Mask Field 27303 is set, only two Event signals are masked: that resulting from an overflow of EGGTMR 25412 and that resulting from an overflow of EU 10122's stack. If Monitor Mask Field 27305 is set, those Events are masked, and additionally, the FU Stack Overflow Event signal is masked. As will be explained in detail later, when the FU 10120 Stack Overflow Event signal is masked, the FU micromachine is executing in monitor mode. If Trace Event Mask Field 27307 is set, Trace Trap Event signals are masked in addition to the above signals. Each of the fields in EM 27301 may be individually set and cleared by the microprogram.

Four Event signals set fields in EP 27309: the EGGTMR 25412 Runout signal sets ET Field 27311, the INTTMR 25410 Runout signal sets IT Field 27313, the Non-Fatal Memory Error signal sets ME Field 27315, and the Inter-Process Message signal sets IPM Field 27317. Event invocations for all of these Event signals but the Egg Timer Runout signal occur at the beginning of an SIN; in these cases the fields in EP 27309 retain the fact that the Event signal has occurred until that time; the Event invocation for the Egg Timer Runout signal occurs as soon after the signal as the settings of mask bits in EM 27301 allow. The bit in ET Field 27311 retains the fact of the Egg Timer Runout signal until the masking allows the Event invocation to occur. All of the fields in EP 27309 but ME Field 27315 may be reset by microcode. The microroutines invoked by the Events must reset the appropriate fields; otherwise, they will be reinvoked when they

return. ME Field 27315 is automatically reset when the memory error is serviced.

TE Register Field 27319 enables tracing. Each bit in the register enables a kind of Trace Event signal when it is set. Depending on the kind of tracing, the Trace Event signal occurs at the beginning of an SIN, at the beginning of a Resolve or Evaluate operation, at the beginning of a logical memory reference, or at the beginning of a microinstruction. For details, see the following description of debugging.

Turning now to the registers contained in RCWS 10358, each RCWS Register 27322 contains eight fields which control Event signals. The first field is FM Field 27323. FM Field 27323 reflects the value of a register in Event Logic 20284 when the invocation to which RCWS Register 27322 belongs occurs. The register in Event Logic 20284 is set only when the microinstruction currently being executed is the first microinstruction of an SIN. Thus, FM Field 27323 is set only in RCWS Registers 27322 belonging to Event invocations which occur in the M0 cycle of the first microinstruction in the SIN, i.e., at the beginning of the SIN. The value of the register in Event Logic 20284 is saved in FM Field 27323 because several Event invocations may occur at the beginning of a single SIN. The Event invocations occur in order of priority: when the one with the highest priority returns, the fact that FM Field 27323 is set causes the register in Event Logic 20284 to again be set to the state which it has on the first microinstruction of an SIN. The register's state, thus set, causes the next Event invocation which must occur at the beginning of the SIN to take place. After all such invocations are finished, the first microinstruction enters its M1 cycle and resets the register in Event Logic 20284. In its reset state, the register inhibits all Event invocations which may occur only at the beginning of an SIN. It is again set at the beginning of the next SIN.

The remaining fields in RCWS Register 27322 which control Event invocations are the fields in Return Signals Field 27331. These fields allow the information that an Event signal has occurred to be retained through Event invocations until the Event signal's Event invocation takes place. When an invocation occurs, these fields are set by Event Logic 20284. On return from the invocation, the values of the fields are input into Event Logic 20284, thereby producing Event signals. The Event signal with the highest priority results in an Event invocation, and the remaining Event signals set fields in Return Signals Field 27331 belonging to RCWS Register 27322 belonging to the invocation which is being executed when the Event signals occur. Because the fields in Return Signals Field 27330 are input into Event Logic 20284, microcode invoked as a consequence of Event signals which sets one of these fields must reset the field itself. Otherwise, the return from the microcode will simply result in a reinvocation of the microcode.

The seven fields in Return Signals Field 27330 have the following significance:

- When EG Field 27333 is set, an EU 10122 dispatch operation produced an illegal location in EU 10122 microcode EUSITT 20344.
- When NT Field 27335, ST Field 27341, mT Field 27343, or mB Field 27345 is set, a trace signal has occurred. These are explained in detail in the discussion of debugging.
- When ES Field 27337 is set, an EU 10122 Storeback Exception has occurred, i.e., an error occurred when EU 10122 attempted to store the result of an operation in MEM 10112.
- When MRR Field 27339 is set, a condition such as an ATU 10228 miss or a Protection Cache 10234 miss has occurred, and it is necessary to reattempt a memory reference.

d. Virtual Micromachines and the Monitor Micromachine

As previously described, microcode being executed on FU 10120's micromachine can run in either monitor mode or virtual mode. In this portion of the discussion, the distinguishing features and applications of the two modes are explained in detail.

1. Virtual Mode

As previously mentioned, the chief distinction between virtual mode and monitor mode is MIS 10368. The fact that MIS 10368 is of essentially unlimited size has the following consequences for microroutines which execute in virtual mode.

- An invocation of a microroutine executing in virtual mode may have as its consequence further invocations to any depth.
- Any invocation of or return from a microroutine executing in virtual mode may cause a page fault.

The FU micromachine is in virtual mode when all bits in the Event Masks portion of MCW1 20290 are cleared. In this state, no enabled Event signals are masked, and Event invocations may occur in any microinstruction which does not itself mask them.

Because invocations may occur to any depth in virtual mode, microroutines executing in this mode may be recursive. Such recursive microroutines are especially useful for the interpretation of Names. Often, as previously described, the Name Table Entry for a Name will contain Names which resolve to other Names, and the virtual micromachine's limitless stack allows the use of recursive Name Resolution microroutines in such situations. Recursive microroutines may also be used for complex SINs, such as Calls.

Because invocations can occur to any depth, any number of Events may occur while a microroutine is executing in monitor mode. This in turn greatly simplifies Event handling. If an Event signal occurs while an Event with a given priority is being handled and the Event being signalled has a higher priority than the one

being handled, the result is simply the invocation of the new Event's handler. Thus, the order in which the Event handlers finish corresponds exactly to the priorities of their Events: those with the highest finish first.

A page fault may occur on any microinvocation or return executed in virtual mode because an invocation in virtual mode which occurs when there are no more Free Frames 27207 on SRs 10362 causes an Event signal which invokes a microroutine running in monitor mode. The microroutine transfers MIS Frames 27203 from GRF 10354 to Secure Stack 10336 in MEM 10112, and the transfer may cause a page fault. Similarly, when a microreturn takes place from the last frame on MIS Frames 27203 on SRs 10362, an Event signal occurs which invokes a microroutine that transfers additional frames from Secure Stack 10336 to GRF 10354, and this transfer, too, may cause a page fault.

The fact that page faults may occur on microinvocations or microreturns in virtual mode has two important consequences: microroutines which cannot tolerate page faults other than those explicitly generated by the microroutine itself cannot execute in virtual mode, and because unexpected page faults cause execution to become indeterminate, microroutines which must run to completion cannot execute in virtual mode. For example, if the microroutine which handles page faults executed in virtual mode, its invocation could cause a page fault, which would cause the microroutine to be invoked again, which would cause another page fault, and so on through an infinite series of recursions.

2. Monitor Micromachine

As previously described, the essential feature of monitor mode is MOS 10370. In a present embodiment of CS 10110, this stack has a fixed minimum size, and is always contained in GRF Registers 10354. The nature of MOS 10370 has four consequences for microroutines which execute in monitor mode:

- When the micromachine is in monitor mode, the depth of invocations is limited; recursive microroutines therefore cannot be executed in monitor mode, and Event invocations must be limited.
- Invocations of microroutines or returns from microroutines in monitor mode never result in page faults.
- Microroutines executing in monitor mode are guaranteed to run to completion if they do not suspend the Process 610 which they are executing or perform a Call to software.
- When the micromachine is executing in monitor mode, it is guaranteed to return to virtual mode within a reasonable period of time, either because a microroutine executing in monitor mode has run to completion, or because the microroutine has suspended the Process 610 which it is executing, or has made a Call to software. The result in both cases is the execution of a new sequence of SOPs, and thus a return to virtual mode.

In a present embodiment of CS 10110, the FU micromachine is in monitor mode when a combination of masking bits in MCW1 20290 is set which results in the masking of the FU Stack Overflow Event and the Egg Timer Overflow Event. As previously described, these Events are masked if Fields 27303, 27305, or 27307 is set. These Events and the consequences of masking them are explained in detail below.

The event signal for the FU Stack Overflow Event occurs on microinvocations for which there is no frame available in MIS Frames 27203. If the Event signal is not masked, it causes the invocation of a microroutine which moves MIS Frames from MIS Frames 27203 onto a Process 610's Secure Stack 10336. When the FU Stack Overflow Event is masked, all frames in SRs 10362 of GRs 10360 are available for microroutine invocations and microroutine invocations will not result in page faults, but if the capacity of SRs 10362 is exceeded, FU 10120 ceases operation.

The Egg Timer Overflow event signal occurs when Egg TMR 25412 runs out. As will be explained in detail later, Egg TMR 25412 ensures that an Interval Timer Runout, an Inter-processor Message, or a Non-fatal Memory Error will be serviced by JP 10114 within a reasonable amount of time. If an Interval Timer Runout Event signal or an Inter-processor Message Event signal occurs at a time when it is inefficient for the FU micromachine to handle the Event, Egg TMR 25412 begins running. When Egg TMR 25412 runs out, the Event is handled unless the micromachine is in monitor mode. If the Egg TMR 25412 Runout Event signal occurs while the FU micromachine is in monitor mode, i.e., while the Event is masked, the Event signal sets Field 27311 in MCW1 20290. When the FU micromachine reverts to virtual mode, i.e., when all Event Mask bits in MCW1 20290 are cleared, the Egg TMR 25412 Runout Event occurs, and the Interval Timer Runout Inter-processor Message Event handlers are invoked by Event Logic 20284.

e. Interrupt and Fault Handling

1. General Principles

Any computer system must be able to deal with occurrences which disrupt the normal execution of a program. Such occurrences are generally divided into two classes: faults and interrupts. A fault occurs as a consequence of an attempt to execute a machine instruction, and its occurrence is therefore synchronous with the machine instruction. Typical faults are floating point overflow faults and page faults. A floating point overflow fault occurs when a machine instruction attempts to perform a floating point arithmetic operation and the result exceeds the capacity of the CS 10110's floating point hardware, that is EU 10122. A page fault occurs when a machine instruction in a computer system with virtual memory attempts to reference data which is not presently available in the computer system's primary memory, that is MEM

10112. Since faults are synchronous with the execution of machine instructions and in many cases the result of the execution of specific machine instructions, their occurrence is to some extent predictable.

The occurrence of an interrupt is not predictable. An interrupt occurs as a consequence of some action taken by the computer system which has no direct connection with the execution of a machine instruction by the computer system. For example, an I/O interrupt occurs when data transmitted by an I/O device (IOS 10116) reaches the central processing unit (FU 10120), regardless of the machine instruction the central processing unit is currently executing.

In conventional systems, interrupts and faults have been handled as follows: if an interrupt or fault occurs, the computer system recognizes the occurrence before it executes the next machine instruction and executes an interrupt-handling microroutine or Procedure 602 instead of the next machine instruction. If the interrupt or fault cannot be handled by the Process 610 in which it occurs, the interrupt or fault results in a process swap. When the interrupt handling routine is finished, Process 610 which faulted or was interrupted can be returned to the CPU if it was removed and the next machine instruction executed.

While the above method works well with faults, the fact that interrupts are asynchronous causes several problems:

— Machine instructions cannot require an indefinite amount of time to execute, since interrupts cannot be handled until the machine instruction during which they occur is finished.

— It must be possible to remove a Process 610 from the CPU at any time, since the occurrence of an interrupt is not predictable. This requirement greatly increases the difficulty of process management.

The method used for interrupt and fault handling in a present embodiment of CS 10110 is described below.

2. Hardware Interrupt and Fault Handling in CS 10110

In CS 10110, there are two levels of interrupts: those which may created and dealt with completely by software, and those which may created by hardware signals. The former class of interrupts is dealt with in the discussion of Processes 610; the latter, termed hardware interrupts, is discussed below.

In CS 10110, hardware interrupts and faults begin as invocations of microroutines in FU 10120. The invocations may be the result of Event signals or may be made by microprograms. For example, when IOS 10116 places data in MEM 10112 for JP 10114, an inter-processor Message Event signal results, and the signal causes the invocation of Inter-processor Message Interrupt handler microcode. On the other hand, a Page Fault begins as an invocation of Page Fault microcode by LAT microcode. The actions taken by the microcode which begins handling the fault or interrupt depend on whether the fault or interrupt is handled by the Process 610 which was being executed when the fault or Event occurred or by a special KOS Process 610.

In the first case, the Event microcode may perform a Microcode-to-Software Call to a high-level language procedure which handles the Event. An example of an Event handled in this fashion is a floating point overflow: when FU 10120 microcode determines that a floating point overflow has occurred, it invokes microcode which may invoke a floating point overflow procedure provided by the high-level language whose S-Language was being executed when the overflow occurred. In alternate embodiments of CS 10110, the overflow procedure may also be in microcode.

In the second case, the microcode handling the fault or interrupt puts information in tables used by a KOS Process 610 which handles the fault or interrupt and then causes the KOS Process 610 to run at some later time by advancing an Event Counter awaited by the Process 610. Event Counters and the operations on them are explained in detail in a following description of Processes 610. Since the tables and Event Counters manipulated by microcode are always present in MEM 10112, these operations do not cause page faults, and can be performed in monitor mode. For example, when IOS 10116 transmits an IPM Event signal to JP 10114 after IOS 10116 has loaded data into MEM 10112, the Event resulting from the Event signal invokes microcode which examines a queue containing messages from IOS 10116. The messages in the queue contain Event Counter locations, and the microcode which examines the queue advances those Event counters, thereby causing Processes 610 which were waiting for the data returned by the I/O operation to recommence execution.

3. The Monitor Mode, Differential Masking and Hardware Interrupt Handling

FU 10120 micromachine's monitor mode and differential masking facilities allow a method of hardware interrupt handling which overcomes two problems associated with conventional hardware interrupt handling: an interrupt can be handled in a predictable amount of time regardless of the amount of time required to execute an SIN, and if the microcode which handles the interrupt executes in monitor mode, the interrupt may be handled at any time without unpredictable consequences. There are two sources of hardware interrupts in CS 10110: an Inter-Processor Message (IPM) and an Interval Timer 25410 Runout. An IPM occurs when IOS 10116 completes an I/O task for JP 10114 and signals completion of the task via IOJP Bus 10132. An Interval Timer Runout occurs when a preset time at which CS 10110 must take some action is reached. For example, a given Process 610 may have a limit placed on the amount of time it may execute on JP 10114. As is explained in a following description of process synchronization, the virtual processor management system sets Interval Timer 25412 to run out when Process 610 has used all of the time available to it.

Both IPMs and Interval Timer Runouts begin as Event signals. The immediate effect of the Event signal is to set a bit in EP Field 27309 of MCW1. In principle, the set bit can cause invocation of the event microcode for the Event on the next M0 cycle in which the FU 10120 micromachine is in virtual mode. Since microroutines running in monitor mode are guaranteed to return the micromachine to virtual mode within a reasonable length of time, and the Event invocation will occur when this happens, the Event is guaranteed to be serviced in a reasonable period of time. The microroutines invoked by the Events themselves execute in monitor mode, thereby guaranteeing that no page faults will occur while they are executing and that Process 610 which is executing on JP 10114 when the hardware interrupt occurs need not be removed from JP 10114.

While hardware interrupts are serviced in principle as described above, considerations of efficiency require that as many hardware interrupts as possible be serviced when the size of the FU micromachine's stack is at a minimum, i.e., at the beginning of an SIN's execution. This requirement is achieved by means of Egg TMR 25412 and ET Flag 27311 in MCW1 20290. As described above, when an IPM interrupt or an Interval Timer 25410 Runout interrupt occurs, Field 27317 or 27313 respectively is set in MCW1 20290. At the same time, Egg TMR 25412 begins running. If the current SIN's execution ends before Egg TMR 25412 runs out, the set Field in MCW1 20290 causes the Interval Timer Runout or Inter-processor Message Event invocations to occur on the first microinstruction for the next SIN. If, on the other hand, the current SIN's execution does not end before Egg TMR 25412 runs out, the Egg Timer Runout causes an Event signal. The immediate result of this signal is the setting of ET bit 27311 in MCW1 20290, and the setting of ET bit 27311 in turn causes the Interval Timer Runout Event invocation and/or IPM Event invocation to take place on the next M0 cycle to occur while the micromachine is in virtual mode. The above mechanism thus guarantees that most hardware interrupts will be handled at the beginning of an SIN, but that hardware interrupts will always be handled within a certain amount of time regardless of the length of time required to execute an SIN.

g. FU Micromachine and CS 10110 Subsystems

The subsystems of CS 10110, such as the object subsystem, the process subsystem, the S-Interpreter subsystem, and the Name Interpreter subsystem, are implemented all or in part in the micromachine. The description of the micromachine therefore closes with an overview of the relationship between these subsystems and the micromachine. Detailed descriptions of the operation of the subsystems have been presented previously.

The subsystems fall into three main groups: KOS subsystems, the Name Interpreter subsystem, and the S-Interpreter subsystem. The relationship between the three is to some extent hierarchical: the KOS subsystems provide the environment required by the Name Interpreter subsystem, and the Name Interpreter subsystem provides the environment required by the S-Interpreter subsystem. For example, the S-Interpreter subsystem interprets SINs consisting of SOPs and Names; the Name Interpreter subsystem translates Names into logical descriptors, using values called ABPs to calculate the locations contained in the logical descriptors. The KOS subsystems calculate the values of the ABPs, translate Logical Descriptors 27116 into physical MEM 10112 addresses, and check whether a Process 610 has access to an object which it is referencing.

In a present embodiment of CS 10110, the Name Interpreter subsystem and the S-Interpreter subsystem are implemented completely in the micromachine; in other embodiments, they could be implemented in high-level languages or in hardware. The KOS subsystems are implemented in both the micromachine and in high-level language routines. In alternate embodiments of CS 10110, KOS subsystems may be embodied entirely in microcode, or in high-level language routines. Some high-level language routines may execute in any Process 610, while others are executed only by special KOS Processes 610. The KOS subsystems also differ from the others in the manner in which the user has access: with the S-Interpreter subsystem and the Name Interpreter subsystem, the subsystems come into play only when SINs are executed; the subsystems are not directly visible to users of the system. Portions of the KOS subsystems, on the other hand, may be explicitly invoked in high-level language programs. For example, an invocation in a high-level language program may cause KOS to bind a Process 610 to a Virtual Processor 612.

The following will first list the functions performed by the subsystems, and then relate the subsystems to the monitor and virtual micromachine modes and specific micromachine devices. KOS subsystems perform the following functions:

- Virtual memory management;
- Virtual processor management;
- Inter-processor communication;
- Access Control;
- Object management; and,
- Process management.

The Name Interpreter performs the following functions:

- Fetching and parsing SOPs, and
- Interpreting Names.

The S-Interpreter, finally, dispatches SOPs, i.e., locates the FU 10120 and EU 10122 microcode which

executes the operation corresponding to a given SOP for a given S-Language.

Of these subsystems, the S-Interpreter, the Name Interpreter, and the microcode components of the KOS process and object manager subsystems execute on the virtual micromachine; the microcode components of the remaining KOS subsystems execute on the monitor micromachine. As will be seen in the discussions of these subsystems, subsystems which execute on the virtual micromachine may cause Page Faults, and may therefore reference data located anywhere in memory; subsystems which execute on the monitor micromachine may not cause Page Faults, and the data bases which these subsystems manipulate must therefore always be present at known locations in MEM 10112.

The relationship between subsystems and FU 10120 micromachine devices is the following: Microcode for all subsystems uses DESP 20210, Microcode Addressing 27013, and Register Addressing 27011, and may use EU Interface 27007. S-Interpreter microcode uses SOP Decoder 27003, and Name Interpreter Microcode uses Instruction Stream Reader 27001, Parsing Unit 27005, and Name Translation Unit 27015. KOS virtual memory management microcode uses Memory Reference Unit 27017, and Protection Microcode uses Protection Unit 27019.

Having described in detail the structure and operation of CS 10110's major subsystems, MEM 10112, FU 10120, EU 10122, IOS 10116, and DP 10118, and the CS 10110 micromachine, CS 10110 operation will be described in further detail next below. First, operation of CS 10110's Namespace, S-Interpreter, and Pointer Systems will be described. Then, operation of CS 10110 will be described in further detail with respect to CS 10110's Kernel Operating System.

3. Namespace, S-Interpreters, and Pointers (Figs. 301—307, 274)

The preceding chapters have presented an overview of CS 10110, examined its hardware in detail, and explained how the FU 10120 hardware functions as a micromachine which controls the activities of other CS 10110 components. In the remaining portions of the specification, the means are presented by which certain key features of CS 10110 are implemented using the hardware, the micromachine, tables in memory, and high-level language programs. The present chapter presents three of these features: the Pointer Resolution System, Namespace, and the S-interpreters.

The Pointer Resolution System translates pointers, i.e., data items which contain location information, into UID-offset addresses. Namespace has three main functions:

- It locates SINs and fetches them from CS 10110's memory into FU 10120.
- It parses SINs into SOPs and Names.
- It translates Names into Logical Descriptors 27116 or values.

The S-interpreters decode S-operations received from namespace into locations in microcode contained in FUSITT 11012 and EUSITT 20344 and then execute that microcode. If the S-operations require operands, the S-interpreters use Namespace to translate the operands into Logical Descriptors 27116 or values as required by the operations.

Since Namespace depends on the Pointer Resolution System and the S-interpreters depend on Namespace, the discussion of the systems begins with pointers and then deals with namespace and S-interpreters.

A. Pointers and Pointer Resolution (Figs. 301, 302)

A pointer is a data item which represents an address, i.e., in CS 10110, a UID-offset address. CS 10110 has two large classes of pointers: resolved pointers and unresolved pointers. Resolved pointers are pointers whose values may be immediately interpreted as UID-offset addresses; unresolved pointers are pointers whose values must be interpreted by high level language routines or microcode routines to yield UID-offset addresses. The act of interpreting an unresolved pointer is called resolving it. Since the manner in which an unresolved pointer is resolved may be determined by a high-level language routine written by a system user, unresolved pointers provide a means by which users of the system may define their own pointer types.

Both resolved and unresolved pointers have subclasses. The subclasses of resolved pointers are UID pointers and object relative pointers. UID pointers contain a UID and offset, and can thus represent any CS 10110 address; object-relative pointers contain only an offset; the address's UID is assumed to be the same as that of the object containing the object-relative pointer. An object-relative pointer can therefore only represent addresses in the object which contains the pointer.

The subclasses of unresolved pointers are ordinary unresolved pointers and associative pointers. The difference between the two kinds of unresolved pointers is the manner in which they are resolved. Ordinary unresolved pointers are always resolved by high-level language routines, while associative pointers are resolved the first time they are used in a Process 610 and a domain by high-level language routines, but are subsequently resolved by means of a table called the Associated Address Table (AAT). This table is accessible to microcode, and associative pointers may therefore be more quickly resolved than ordinary unresolved pointers.

The following discussion will first explain the formats used by all CS 10110 pointers, and will then explain how pointers are processed in FU 10120.

a. Pointer Formats (Fig. 301)

Figure 301 represents a CS 10110 pointer. The figure has two parts: a representation of General Pointer Format 30101, which gives an overview of the fields which appear in all CS 10110 pointers, and a detailed presentation of Flags and Format Field 30105, which contains the information by which the kinds of CS 10110 pointers are distinguished.

Turning first to General Pointer Format 30101, all CS 10110 pointers contain 128 bits and are divided into three main fields:

- Offset Field 30103 contains the offset portion of a UID-offset address in resolved pointers and in associative pointers; in other unresolved pointers, it may contain an offset from some point in an object or other information as defined by the user.
- Flags and Format Field 30105 contains flags and format codes which distinguish between kinds of pointers. These flags and format codes are explained in detail below.
- UID field 30115 contains a UID in UID pointers and in some associative pointers; in objectrelative pointers, and other associative pointers, its meaning is undefined, and in ordinary unresolved pointers, it may contain information as defined by the user.

Flags and Format Field 30105 contains four subfields:

- Fields 30107 and 30111 are reserved and must be set to 0.
- NR Field 30109 indicates whether a pointer is resolved or unresolved. In resolved pointers, the field is set to 0, and in unresolved pointers, it is set to 1.
- Format Code Field 30113 indicates the kind of resolved or unresolved pointers. Format codes for the present embodiment are explained below.

The values of Format Code Field 30113 may range from 0 to 31. If Format Code Field 30113 has the value 0, the pointer is a null pointer, i.e., a pointer which neither directly nor indirectly indicates an address. The meanings of the other format codes depend on the value of NR Field 30109:

NR Field Value	Format Code Value	Meaning
0	1	UID pointer
0	2	Object-relative pointer
0	all other codes	Illegal
1	1	UID associative pointer
1	2	Object-relative associative pointer
1	all other codes	Ordinary unresolved pointer

As indicated by the above table, the present embodiment has two kinds of associative pointer, UID associative pointers and object-relative associative pointers. Like a UID pointer, a UID associative pointer contains a UID and an offset, and like an object-relative pointer, an object-relative associative pointer contains an offset and takes the value of the UID from the object to which it belongs. However, as will be explained in detail later, the UID and offset which the associative pointers contain or represent are not used as addresses. Instead, the UID and offset are used as tags to locate entries in the AAT, which associates an associative pointer with a resolved pointer.

b. Pointers in FU 10120 (Fig. 302)

When a pointer is used as an address in FU 10120, the address information in the pointer must be translated into a Logical Descriptor 27116 consisting of an AON, an offset, and a length field of 0; when a Logical Descriptor 27116 in FU 10120 is used to form a pointer value in memory, the AON must be converted back to a UID. The first conversion is termed pointer-to-descriptor conversion, and the second descriptor-to-pointer conversion. Both conversions are accomplished by microcodes executing in FU 10120.

What is involved in the translation depends on the kind of pointer: if the pointer is a UID pointer, the UID must be translated into an AON; if the pointer is an object-relative pointer, the AON required to fetch the pointer is the pointer's AON, so no translation is necessary. If the pointer is an unresolved pointer, it must first be translated into a resolved pointer and then into a Logical Descriptor 27116. If the pointer is associative, the translation to a resolved pointer may be performed by means of the ATT.

In the present embodiment, when other FU 10120 microcode calls pointer-to-descriptor microcode, the calling microcode passes Logical Descriptor 27116 for the location of the pointer which is to be translated as an argument to the pointer-to-description translation microcode. The pointer-to-descriptor microcode returns a Logical Descriptor 27116 produced from the value of the pointer at the location specified by

Logical Descriptor 27116 which the pointer-to-descriptor microcode received as an argument.

The pointer-to-descriptor microcode first uses Logical Descriptor 27116 given it as an argument to fetch the value of the pointer's Offset Field 30103 from memory. It then saves Logical Descriptor 27116's offset in the output register belonging to OFFALU 20242 and places the value of the pointer's Offset Field 30103 in the offset field of Logical Descriptor 27116 which it received as an argument. The pointer-to-descriptor microcode then saves Logical Descriptor 27116 indicating the pointer's location by storing Logical Descriptor 27116's AON and offset (obtained from OFFALU 20242) in a register in the GRF 10354 frame being used by the invocation of the pointer-to-descriptor microcode. Next, the microcode adds 40 to the offset stored in OFFALU 20242, thereby obtaining the address of NR Field 30109, and uses the address to fetch and read NR Field 30109 and Format Code Field 30113. The course of further processing is determined by the values of these fields. If NR Field 30109 indicates a resolved pointer, there are four cases, as determined by the value of Format Code Field 30113:

- Format code field = 0: The pointer is a null pointer.
- Format code field = 1: The pointer is a UID pointer.
- Format code field = 2: The pointer is an intra-object pointer.
- Any other value of the format code field: The pointer is invalid.

In the first case, the microcode sets all fields of the argument to 0; in the second, it fetches the value of UID Field 30115 from memory and invokes LAR microcode (explained in the discussion of objects), which translates the UID to the AON associated with it. The AON is then loaded into the argument's AON field. In the third case, the AON of Logical Descriptor 27116 for the pointer's location and the pointer's AON are the same, so the argument already contains the translated pointer. In the fourth case, the microcode performs a call to a pointer fault-handling Procedure 602 which handles invalid pointer faults, passing saved Logical Descriptor 27116 for the pointer as an argument. Procedure 602 which handles the fault must return a resolved pointer to the microcode, which then converts it to a Logical Descriptor 27116 as described above.

c. Descriptor to Pointer Conversion

Descriptor to pointer conversion is the reverse of pointer to descriptor conversion with resolved pointers. The operation must be performed whenever a resolved pointer is moved from an FU 10120 register into MEM 10112. The operation takes two arguments: a Logical Descriptor 27116 which specifies the address to which the pointer is to be written, and a Logical Descriptor 27116 whose AON and offset fields specify the location contained in the pointer. There are two cases: intra-object pointers and UID pointers. Both kinds of pointers have values in Offset Field 30103, so the descriptor-to-pointer microcode first writes the second argument's offset to location specified by the first argument's Logical Descriptor 27116. The next step is to determine whether the pointer is an intra-object pointer or a UID pointer. To do so, the microcode compares the arguments' AONs. If they are the same, the pointer points to a location in the object which contains it, and is therefore an intra-object pointer. Since UID Field 30115 of an intra-object pointer is meaningless, the only step remaining for intra-object pointers is to set Flags and Format Field 30105 to the binary representation of 2, which sets all bits but bit 46 to 0, and thereby identifies the pointer as a resolved intra-object pointer.

With UID pointers, the descriptor-to-pointer microcode sets Flags and Format Field 30105 to 1, thereby identifying the pointer as a resolved UID pointer, and calls a KOS LAR microroutine (explained in detail in the discussion of objects) which converts the first argument's AON to a UID and places the result UID in the current frame. When the KOS AON to UID conversion microroutine returns, the descriptor-to-pointer microcode writes the UID to the converted pointer's UID Field 30115.

B. Namespace and the S-Interpreters (Figs. 303—307)

Namespace and the S-Interpreter both interpret information contained in Procedure Objects 608. Consequently, the discussion of these components of CS 10110 begins with an overview of those parts of Procedure Object 606 relevant to Namespace and the S-Interpreters, and then explains Namespace and the S-interpreters in detail.

a. Procedure Object 606 Overview (Fig. 303)

Figure 303 represents those portions of Procedure Object 608. Fig. 303 expands information contained in Fig. 103; Fields which appear in both Figures have the number of Fig. 103. Portions of Procedure Object 608 which are not discussed here are dealt with later in the discussion of Calls and Returns. The most important part of a Procedure Object 608 for these systems is Procedure Environment Descriptor (PED) 30303. A Procedure 602's PED 30303 contains the information required by Namespace and the S-interpreter to locate and parse Procedure 602's code and interpret its Names. A number of Procedures 602 in a Procedure Object 608 may share a PED 30303. As will be seen in the discussion of Calls, the fact that a Procedure 602 shares a PED 30303 with the Procedure 602 that invokes it affects the manner in which the Call is executed.

The fields of PED 30303 which are important to the present discussion are three fields in Header 30304: K Field 30305, LN Field 30307, and SIP Field 30309, and three of the remaining fields: NTP Field 30311, SDPP Field 30313, and PBP Field 30315.

— K Field 30305 indicates whether the Names in the SINs of Procedures 602 which share PED 30303 have 8, 12, or 16 bits.
— LN Field 30307 contains the Name which has the largest index of any in Procedure 602's Name Table 10350.
— SIP Field 30309 is a UID pointer to the object which contains the S-interpreter for Procedure 602's S-Language.
— NTP Field 30311 is an object-relative pointer to the beginning of Procedure 602's Name Table 10350.
— SDPP Field 30313 is a pointer which is resolved to the location of static data used by Procedures 602 to which PED 30303 belongs when one of Procedures 602 is invoked by a given Process 610. The resolved pointer corresponding to SDPP 30313 is the SDP ABP.
— PBP Field 30315 contains the PBP ABP for invocations of Procedures 602 to which PED 30303 belongs. The PBP ABP is used to calculate locations inside Procedure Object 608.

Other areas of interest in Procedure Object 608 are Literals 30301 and Static Data Prototype (SDPR) 30317. Literals 30301 contains literal values, i.e., values in Procedure 602 which are known at compile time and will not change during program execution. SDPR 30317 may contain any of the following: pointers to external routines and to static data contained in other objects, information required to create static data for a Procedure 602, and in some cases, the static data itself. Pointers in SDPR 30317 may be either resolved or non-resolved.

In the present embodiment, Binder Area 30323 is also important. Binder Area 30323 contains information which allows unresolved pointers contained in Procedure Object 608 to be resolved. Unresolved pointers other than SDPP 30313 in Procedure Object 608 all contain locations in Binder Area 30323, and the specified location contains the information required to resolve the pointer.

Fig. 303 contains arrows showing the locations in Procedure Object 608 pointed to by NTP Field 30311, SDPP Field 30313, and PBP Field 30315. NTP Field 30311 points to the beginning of Name Tables 10350, and thus a Name's Name Table Entry can be located by adding the Name's value to NTP Field 30311. PBP Field 30315 points to the beginning of Literals 30301, and consequently, the locations of Literals and the locations of SINs may be expressed as offsets from the value of PBP Field 30315. SDPP Field 30313 points to the beginning of SDPR 30317. As will be explained in detail in the discussion of Calls, when a procedure 602 has static data, the SDP ABP is derived from SDPP Field 30313.

b. Namespace

The Namespace component of CS 10110 locates SINs belonging to a procedure and fetches them from memory to FU 10120, parses SINs into SOPs and Names, and performs Resolve and Evaluation operations on Names. The Resolve operation translates a Name into a Logical Descriptor 27116 for the data represented by the Name, while the Evaluation operation obtains the data itself. The Evaluation operation does so by performing a Resolve operation and then using the resulting Logical Descriptor 27116 to fetch the data. Since the Evaluation and Resolve operations are the most complicated, the discussion begins with them.

1. Name Resolution and Evaluation

Name Resolution and Evaluation translate Names into Logical Descriptors 27116 by means of information contained in the Names' NTEs, and the NTEs define locations in terms of Architectural Base Registers. Consequently, the following discussion will first describe Name Table Entries and Architectural Base Pointers and then the means by which Namespace translates the information contained in the Name Table Entries and Architectural Base Pointers into Logical Descriptors 27116.

2. The Name Table (Fig. 304)

As previously mentioned, Name Tables 10350 are contained in Procedure Objects 608. Name Tables 10350 contain the information required to translate Names into Logical Descriptors 27116 for the operands represented by the Names. Each Name has as its value the number of a Name Table Entry. A Name's Name Table Entry is located by multiplying the Name's value by the size of a short Name Table Entry and adding the product to the value in NTP Field 30311 of PED 30303 belonging to Procedure 602 which contains the SIN.

The Name Table Entry contains length and type information for the data item specified by the Name, and represents the data item's location as a displacement from a known location, termed the base. The base may be a location specified by an ABP, a location specified by another Name, or a location specified by a pointer. In the latter case, the pointer's location may be specified in terms of an ABP or as a Name.

Fig. 304 is a detailed representation of a Name Table Entry (NTE) 30401. There are two kinds of NTEs 30401: Short NTEs 30403 and Long NTEs 30405. Short NTEs 30403 contain 64 bits; Long NTEs 30405 contain 128 bits. Names that represent scaler data items whose displacements may be expressed in 16 bits have Short NTEs 30403; Names that represent scaler data items whose displacements require more than 16 bits and Names that represent array elements have Long NTEs 30405.

A Short NTE 30403 has four main fields, each 16 bits in length:

— Flags and Format Field 30407 contains flags and format information which specify how Namespace is to interpret NTE 30401.

— Base Field 30425 indicates the base to which the displacement is to be added to obtain the location of the data represented by the Name. Base Field 30425 may represent the location in four ways: by means of an ABP by means of a Name, by means of a pointer located by means of an ABP, and by means of a pointer located by means of a Name.
— Length Field 30435 represents the length of the data. The length may be a literal value or a Name. If it is a Name, the Name resolves to a location which contains the data item's length.
— Displacement Field 30437 contains the displacement of the beginning of the data from the base specified in Field 30425. The displacement is a signed integer value.

Long NTEs 30405 have four additional fields, each 16 bits long: Two of the fields, Index Name Field 30441 and IES Field 30445 are used only in NTEs 30401 for Names that represent arrays.

— Displacement Extension Field 30439 is used in all Long NTEs 30405. If the displacement value in Field 30437 has less than 16 bits, Displacement Extension Field 30439 contains sign bits, i.e., the bits in the field are set to 0 when the displacement is positive and 1 when the displacement is negative. When the displacement value has more than 16 bits, Displacement Extension Field 30439 contains the most significant bits of the displacement value as well as sign bits.
— Index Name Field 30441 contains a Name that represents a value used to index an element of an array.
— Field 30443 is reserved.

IES Field 30445 contains a Name or Literal that specifies the size of an element in an array. The value represented by this field is used together with the value represented by Index Name Field 30441 to locate an element of an array.

As may be seen from the above, the following fields may contain names: Base Field 30425, Length Field 30435, Index Name Field 30441, and IES Field 30445.

Two fields in NTE 30401 require further consideration: Flags and Format Field 30407 and Base Field 30425. Flags and Format Field 30407 has three subfields: Flags Field 30408, FM Field 30421, and Type Field 30423. Turning first to Flags Field 30408, the six flags in the field indicate how Namespace is to interpret NTE 30401. The flags have the following meanings when they are set:

— Long NTE Flag 30409: NTE 30401 is a Long NTE 30405.
— Length is a Name Flag 30411: Length Field 30435 contains a Name.
— Base is a Name Flag 30413: Base Field 30425 contains a Name instead of the number of an ABP.
— Base Indirect Flag 30415: Base Field 30425 represents a pointer, and the location represented by NTE 30401 is to be calculated by obtaining the pointer's value and adding the value contained in Displacement Field 30437 and Displacement Extension Field 30439 to the pointer's offset.
— Array Flag 30417: NTE 30401 represents an array.
— IES is a Name Flag 30419: IES Field 30445 contains a Name that represents the IES value.

Several of these flags may be set in a given NTE 30401. For example, an entry for an array element that was referenced via a pointer to the array which in turn was represented by a Name, and whose IES value was represented by a Name, would have Flags 30409, 30413, 30415, 30417, and 30419 set.

FM Field 30421 indicates how the data represented by the Name is to be formated when it is fetched from memory. The value of FM Field 30421 is placed in FIU Field 27107 of Logical Descriptor 27116 produced from NTE 30401. The two bits allow for four possibilities:

Setting	Meaning
00	right justify, zero fill
01	right justify, sign fill
10	left justify, zero fill
11	left justify, ASCII space fill

The four bits in Type Field 30423 are used by compilers for language-specific type information. The value of Type Field 30423 is placed in Type Field 27109 of Logical Descriptor 27116 produced from NTE 30401.

Base Field 30425 may have either Base is an ABP Format 30427 or Base is a Name Format 30432. The manner in which Base Field 30425 is interpreted depends on the setting of Base is a Name Flag 30413 and Base Indirect Flag 30415. There are four possibilities:

Field Settings

Base is a Name	Base Indirect	Meaning
0	0	ABP Format locates base directly.
0	1	ABP Format locates a pointer which is the base.
1	0	Base is Name Format locates base when Name is resolved.
1	1	Base is Name Format locates a pointer when Name is resolve and the pointer is the base.

As indicated by the above table, Base Field 30425 is interpreted as having Base is ABP Format 30427 when Base is a Name Flag 30411 is not set. In Base is ABP Format 30427, Base Field 30425 has two subfields: ABP Field 30429 and Pointer Locator Field 30431. The latter field has meaning only when Base Indirect Flag 30415 is set. ABP Field 30429 is a two-bit code which indicates the ABP. The settings and their meanings are the following:

Setting	APB
00	FP
01	Unused
10	SDP
11	PBP

The ABPs are discussed below. When Base Indirect Flag 30415 is set to 1 and Base is a Name Flag 30413 is set to 0, the remaining 14 bits of the Base Field in ABP Format are interpreted as Pointer Locator Field 30413. When so interpreted, Pointer Locator Field 30413 contains a signed integer, which, when multiplied by 128, gives the displacement of a pointer from the ABP specified in ABP Field 30429. The value of this pointer is then the base to which the displacement is added.

Base Field 30425 is interpreted as having Base is a Name Format 30432 when Base is a Name Flag 30413 is set to 1. In Base is a Name Format 30432, Base Field 30425 contains a Name. If Base Indirect Flag 30415 is not set, the Name is resolved to obtain the Base. If Base Indirect Flag 30415 is set, the name is evaluated to obtain a pointer value, and that pointer value is the Base.

3. Architectural Base Pointers (Figs. 305, 306)

If Base is a Name Flag 30413 belonging to a NTE 30401 is not set, Base Field 30425 specifies one of the three ABPs in CS 10110:

- PBP specifies a location in Procedure Object 608 to which displacements may be added to obtain the locations of Literals and SINs.
- SDP specifies a location in a Static Data Block for an invocation of a Procedure 602 to which displacements may be added to obtain the locations of static data and linkage pointers to Procedures 602 contained in other Procedure Objects 608 and static data.
- FP specifies a location in the MAS frame belonging to Procedure 602's current invocation to which displacements may be added to obtain the location of local data and linkage pointers to arguments.

Each time a Process 610 invokes a Procedure 602, Call microcode saves the current values of the ABPs on Secure Stack 10336, calculates the values of the ABPs for the new invocation, and places the resulting Logical Descriptors 27116 in FU 10120 registers, where they are accessible to Namespace microcode.

Call microcode calculates the ABPs as follows: PBP is obtained directly from PBP Field 30315 in PED 30303 belonging to the Procedure 602 being executed. All that is required to make it into a Logical Descriptor 27116 is the addition of the AON for Procedure Object 608's UID.

SDP is obtained by performing a pointer-to-descriptor translation on SDPP Field 30313. FP, finally, is provided by the portion of Call microcode which creates the new MAS 502 frame for the invocation. As is described in detail in the discussion of Call, the Call microcode copies linkage pointers to the invocation's actual arguments onto MAS 502, sets FP to point to the location following the last actual argument, and then allocates storage for the invocation's local data. Positive displacements from FP thus specify locations

in the local data, while negative offsets specify linkage pointers.

a.a. Resolving and Evaluating Names (Fig. 305)

The primary operations performed by Namespace are resolving names and evaluating them. A Name has been resolved when Namespace has used the ABPs and information contained in the Name's NTE 30401 to produce a Logical Descriptor 27116 for the Name; a name has been evaluated when Namespace has resolved the Name, presented the resulting Logical Descriptor 27116 for the Name to memory, and obtained the value of the data represented by the Name from memory.

The resolve operation has three parts, which may be performed in any order:

— Obtaining the Base from Base Field 30425 of the Name's NTE 30401.
— Obtaining the displacement.
— Obtaining the length from Length Field 30435.

Obtaining the length is the simplest of the operations: if Length in a Name Flag 30411 is set, the length is the value obtained by evaluating the Name contained in Length Field 30435; otherwise, Length Field 30435 contains a literal value and the length is that literal's value.

There are four ways in which the Base may be calculated. Which is used depends on the settings of Base is a Name Flag 30413 and Base Indirect Flag 30415:

— Both Flags 0: the ABP specified in ABP Field 30429 is the Base.
— Base is a Name Flag 30413 0 and Base Indirect Flag 30415 1: The Base is the location contained in the pointer specified by ABP Field 30429 and pointer Locator Field 30431.
— Base is a Name Flag 30413 1 and Base Indirect Flag 30415 0: The Base is the location obtained by resolving the Name in Base Field 30425.
— Both Flags 1: The Base is the location obtained by evaluating the Name in Base Field 30425.

The manner in which Namespace calculates the displacement depends on whether NTE 30401 represents a scaler data item or an array data item. In the first case, Namespace adds the value contained in Displacement Field 30437 and Displacement Extension Field 30439 to the location obtained for the Base; in the second case, Namespace evaluates Index Name Field 30441 and IES Field 30445, multiplies the resulting values together, and adds the product to the value in Displacement Field 30437 in order to obtain the displacement.

If any field of a NTE 30401 contains a Name, Namespace obtains the value or location represented by the Name by performing a Resolve or Evaluation operation on it as required. As mentioned in the discussion of NTEs 30401, flags in Flags Field 30408 indicate which fields of an NTE 30401 contain Names. Since the NTE 30401 for a Name used in another NTE 30401 may itself contain Names, Namespace performs the Resolve and Evaluation operations recursively.

b.b. Implementation of Name Evaluation and Name Resolve in CS 10110

In the present embodiment, the Name Evaluation and Resolve operations are carried out by FU 10120 microcode Eval and Resolve commands. Both commands require two pieces of information: a register in the current frame of SR portion 10362 of GRF 10354 for receiving Logical Descriptor 27116 produced by the operation, and the source of the Name which is to be resolved or evaluated. Both Resolve and Eval may choose between three sources: Parser 20264, Name Trap 20254, and the low-order 16 bits of the output register for OFFALU 20242. Resolve may specify current frame registers 0, 1, or 2 for Logical Descriptor 27116, and Eval may specify current frame registers 0 or 1. At the end of the Resolve operation, Logical Descriptor 27116 for the data represented by the Name is in the specified SR 10362 register and at the end of the Evaluation operation, Logical Descriptor 27116 is in the specified SR 10362 register and the data's value has been transferred via MOD Bus 10114 to EU 10122's OPB 20322.

The execution of both Resolve and Eval commands always begin with the presentation of the Name to Name Cache 10226. The Name presented to Name Cache 10226 is latched into Name Trap 20254, where it is available for subsequent use by Name Resolve microcode.

If there is an entry for the Name in Name Cache 10226, a name cache hit occurs. For Names with NTEs 30401 fulfilling three conditions, the Name Cache 10226 entry for the Name is a Logical Descriptor 27116 for the data item represented by the Name. The conditions are the following:

— NTE 30401 contains no Names.
— Length Field of NTE 30401 specifies a length of less than 256 bits.
— If Base is Indirect Flag 30415 is set, Pointer Displacement Field 30431 must have a negative value, indicating that the base is a linkage pointer.

Logical Descriptor 27116 can be encached in this case because neither the location nor the length of the data represented by the Name can change during the life of an invocation of Procedure 602 to which the Name belongs. If the Name Cache 10226 entry for the Name is a Logical Descriptor 27116, the hit causes Name Cache 10226 to place Logical Descriptor 27116 in the specified SR 10362 register. In all other cases, the Name Cache 10226 entry for the Name does not contain a Logical Descriptor 27116, and a hit causes Name Cache 10226 to emit a JAM signal. The JAM signal invokes microcode which uses information stored in Name Cache 10226 to construct Logical Descriptor 27116 for the data item represented by the Name. JAMS are explained in detail below.

If there is no entry for the Name in Name Cache 10226, a Name Cache Miss occurs, and Name Cache 10226 emits a cache miss JAM signal. The Name Resolve microroutine invoked by the cache miss JAM

signal constructs an entry in Name Cache 10226 from the Name's NTE 30401, using FU 10120's DESP 20210 to perform the necessary calculations. When it is finished, the cache miss microcode leaves a Logical Descriptor 27116 for the Name in the specified SR 10362 register and returns.

The Resolve operation is over when Logical Descriptor 27116 has been placed in the specified GRF 10354 register; the Evaluation operation continues by presenting Logical Descriptor 27116 to Memory Reference Unit 27017, which reads the data represented by Logical Descriptor 27116 from memory and places it on OPB 20322. The memory reference may result in Protection Cache 10234 misses and ATU 10228 misses, as well as protection faults and page faults, but these are handled by means of event signals and are therefore invisible to the Evaluation operation.

Name Cache 10226 produces 15 different JAM signals. The signal produced by a JAM depends on the following: whether the operation is a Resolve or an Eval, which register Logical Descriptor 27116 is to be placed in, whether a miss occurred, and in the case of a hit, which register in the Name Cache 10226 entry for the Name was loaded last. From the point of view of the behavior of the microcode invoked by the JAM, the last two factors are the most important. Their relation to the microcode is explained in detail below.

In the present embodiment, all entries in Name Cache 10226 are invalidated when a Procedure 602 calls another Procedure 602. The invalidation is required because Calls always change the value of FP and may also change the values of SDP and PBP, thereby changing the meaning of NTEs 30401 using displacements from ABPs. Entries for Names in invoked Procedure 602 are created and loaded into Name Cache 10226 when the Names are evaluated or resolved and a cache miss occurs.

The following discussion will first present Name Cache 10226 as it appears to the microprogrammer and then explain in detail how Name Cache 10226 is used to evaluate and resolve Names, how it is loaded, and how it is flushed.

c.c. Name Cache 10226 Entries (Fig. 306)

The structure and the physical behavior of Name Cache 10226 was presented in the discussion of FU 10120 hardware; here, the logical structure of Name Cache 10226 entries as they appear to the microprogrammer is presented. To the microprogrammer, Name Cache 10226 appears as a device which, when presented a Name on NAME Bus 20224, always provides the microprogrammer with a Name Cache 10226 entry for the Name consisting of four registers. The microprogrammer may read from or write to any one of the four registers. When the microprogrammer writes to the four registers, the action taken by Name Cache 10226 when a hit occurs on the Name associated with the four registers depends on which of the registers has most recently been loaded. The means by which Name Cache 10226 associates a Name with the four registers, and the means by which Name Cache 10226 provides registers when it is full are invisible to the microprogrammer.

Fig. 306 illustrates Name Cache Entry 30601 for a Name. The four Registers 30602 in Name Cache Entry 30601 are numbered 0 through 3, and each Register 30602 has an AON, offset, and length field like those in GRF 10354 registers, except that some flag bits in GRF 10354 register AON fields are not included in Register 30602 fields, and the length field in Register 30602 is 8 bits long. As is the case with GRF 10354 registers, the microprogrammer can read or write individual fields of Register 30602 or entire Register 30602. Name Cache Entry 30601 is connected via DB 27021 to DESP 20210, and consequently, the contents of a GRF 10354 register may be obtained from or transferred to a Register 30602 or viceversa. When the contents of a Register 30602 have been transfered to a GRF 10354 register, the contents may be processed using OFFALU 20242 and other arithmetic-logical devices in DESP 20210.

d.d. Name Cache 10226 Hits

When a Name is presented to Name Cache 10226 and Name Cache 10226 has a Name Cache Entry 30601 containing information about the Name, a name cache hit occurs. On a hit, Name Cache 10226 hardware always loads the contents of Register 30602 0 of the Name's Name Cache Entry 30601 into the GRF 10354 register specified in the Resolve or Eval microcommand. In addition, a hit may result in the invocation of microcode via a JAM:

- The JAM may invoke special microcode for resolving Names of array elements whose NTEs 30401 allow certain hardware accelerations of index calculations.
- The JAM may invoke general name resolution microcode which produces a Logical Descriptor 27116 from the contents of Name Cache Entry 30601.

Whether the hit produces a JAM, and the kind of JAM it produces, are determined by the last Register 30602 to be loaded when Name Cache Entry 30601 was created by Name Cache Miss microcode. If Register 30602 0 was the last to be loaded, no JAM occurs; if Register 30602 1 was loaded last, the JAM for special array Name resolution occurs; if Register 30602 2 or 3 was loaded last, the JAM for general Name resolution occurs.

As may be inferred from the above, Name Cache 10226 hardware defines the manner in which Name Cache Entries 30601 are loaded for the first two cases. In the first case, Name Cache Register 30602 0 must contain Logical Descriptor 27116 for the Name's data. As already mentioned, the Name's NTE 30401 must therefore describe data whose location and length does not change during an invocation and whose length is less than 256 bits. Name Cache 10226 hardware also determines the form of Name Cache Entries 30601 for encachable arrays. An encachable array NTE 30401 is an array NTE 30401 which fills the following

conditions:

- The only Name contained in array NTE 30401 is in Index Name Field 30441.
- NTE 30401 for the index Name fills the conditions for scaler NTEs 30401 for which Logical Descriptors 27116 may be encached.
- The value in IES Field 30445 is no greater than 128 and a power of 2.
- Array NTE 30401 otherwise fills the conditions for scaler NTEs 30401 for which Logical Descriptors 27116 may be encached.

In the present embodiment, the encachable array entry uses registers 0, 1, and 2 of Name Cache Entry 30601 for the name:

Register	Contents		
	AON	OFFSET	LENGTH
0	Logical Descriptor 27116 for the index Name		
1	0	IES power of 2	unused
2	Logical Descriptor 27116 for the array		

When a hit for this type of entry occurs, the resulting JAM signal does two things: it invokes encachable array resolve microcode and it causes the index Name's Logical Descriptor 27116 to be presented to Memory Reference Unit 27017 for a read operation which returns the value of the data represented by the index Name to an accumulator in OFFALU 20242. The encachable array resolve microroutine then uses the Name that caused the JAM, latched into Name Trap 20254, to locate Register 30602 2 of Name Cache Entry 30601 for the Name, writes the contents of Register 30602 2 into the GRF register specified by the Resolve or Eval microcommand, obtains the product of the IES value and the index value by shifting the index value left the number of times specified by the IES exponent in Register 30602 1, adds the result to the offset field of the GRF 10354 register containing the array's Logical Descriptor 27116, thus obtaining Logical Descriptor 27116 for the desired array element, and returns.

For the other cases, the manner in which Name Cache Entries 30601 are loaded and processed to obtain Logical Descriptors 27116 is determined by the microprogrammer. The JAM signal which results if a Name Cache Entry 30601 is neither a Logical Descriptor 27116 nor an encachable array entry merely invokes a microroutine. The microroutine uses the Name latched into Name Trap 20254 to locate the Name's Name Cache Entry 30601 and then reads tag values in Name Cache Entry 30601 to determine how the information in Name Cache Entry 30601 is to be translated into a Logical Descriptor 27116. The contents of Name Cache Entries 30601 for the other cases have two general forms: one for NTEs 30401 with Base is Indirect Flag 30415 set, and one for NTEs in which it is not set. The first general form looks like this:

Register	Contents		
	AON	OFFSET	LENGTH
0	ABP AON	tag/length	unused
1	0	index name/IES	unused
2	0	unused	unused
3	0	data displacement from loc. specified by pointer	unused

Register 30602 0 contains the AON of the ABP. Register 30602 0's offset field contains two items: the tag, which contains Flags Field 30408 of NTE 30401 along with other information, and which determines how Name Resolve microcode interprets the contents of Name Cache Entry 30601, and a value or Name for the length of the data item. Register 30602 1 is used only if the Name represents a data item in an array. It then contains the Name from Index Field 30441 and the Name or value from IES Field 30445. The offset field of Register 30602 3 contains the sum of the offset indicated by NTE 30401's ABP and of the displacement indicated by NTE 30401.

The second format, used for NTEs 30401 whose bases are obtained from pointers or by resolving a Name, looks like this:

Registers	Contents		
	AON	OFFSET	LENGTH
0	0	tag/length	unused
1	0	index name/IES	unused
2	0	FM and type bits/ base field	unused
3	0	data displacement from loc. specified by pointer or name	unused

In this form, the location of the Base must be obtained either by evaluating a pointer or resolving a Name. Hence, there is no field specifying the Base's AON. Otherwise, Registers 30602 0 and 1 have the same contents as in the previous format. In Register 30602 2, the offset field contains Name Table Entry 30401's FM Field 30421 and Type Field 30423 and Base Field 30425. The Offset Field of Register 30602 2 contains the value of Name Table Entry 30401 Displacement Fields 30437 and 30439.

As in Name Table Entries 30401, the index must be represented by a Name, and length, IES, and Base may be represented by Names. If a field of Name Cache Entry 30601 contains a Name, a flag in the tag indicates that fact, and Name Resolve microcode performs an Eval or Resolve operation on it as required to obtain the value or location represented by the name.

The microcode which resolves Name Cache Entries 30601 of the types just described uses the general algorithms described in the discussion of Name Table Entries 30401, and is therefore not discussed further here.

e.e. Name Cache 10226 Misses

When a Name is presented to Name Cache 10226 and there is no Name Cache Entry 30601 for the Name, a name cache miss occurs. On a miss Name Cache 10226 hardware emits a JAM signal which invokes name cache miss microcode. The microcode obtains the Name which caused the miss from Name Trap 20254 and locates the Name's NTE 30401 by adding the Name to the value of NTP 30311 from PED 30303 for Procedure 602 being executed. As will be explained in detail later, when a Procedure 602 is called, the Call microcode places the AON and offset specifying the NTP's location in a register in GR's 10360. Using the information contained in the Name's NTE 30401, the Cache Miss microcode resolves the Name and constructs a Name Cache Entry 30601 for it. As described above, the microcode determines the method by which it resolves the Name and the form of the Name's Name Cache Entry 30601 by reading Flags Field 30408 in the Name's NTE 30401. Since the descriptions of the Resolve operation, the micromachine, Name Cache 10226, and the formats of Name Cache Entries 30601 are sufficient to allow those skilled in the art to understand the operations performed by Cache Miss microcode, no further description of the microcode is provided.

f.f. Flushing Name Cache 10226

As described in the discussion of Name Cache 10226 hardware, hardware means, namely VALS 24068, exist which allow Name Cache Entries 30601 to be invalidated. Name Cache Entries 30601 may be invalidated singly, or all entries in Name Cache 10226 may be invalidated by means of a single microcommand. The latter operation is termed name cache flushing. In the present embodiment, Name Cache 10226 must be flushed when Process 610 whose Virtual Processor 612 is bound to JP 10114 executes a Call or a Return and whenever Virtual Processor 612 NO is unbound from JP 10114. Flushing is required on Call and Return because Calls and Returns change the values of the ABPs and other pointers needed to resolve Names. At a minimum, a Call produces a new MAS Frame 10412, and a Return returns to a previous Frame 10412, thereby changing the value of FP. If the called Procedure 602 has a different PED 30303 from that of the calling Procedure 602, the Call or Return may also change PBP, SDP, and NTP. Flushing is required when a Virtual Processor 612 is unbound from JP 10114 because Virtual Processor 612 which is next bound to JP 10114 is bound to a different Process 610, and therefore cannot use any information belonging to Process 610 bound to the Previous Virtual Processor 612.

g.g. Fetching the I-Stream

As explained in the discussion of FU 10120 hardware, SINs are fetched from memory by Prefetcher 20260. PREF 20260 contains a Logical Descriptor 27116 for a location in Code 10344 belonging to Procedure 602 which is currently being executed. On any MO cycle, PREF 20260 can place Logical Descriptor 27116 on DB 27021, cause Memory Reference Unit 27017 to fetch 32 bits at the location specified by Logical

Descriptor 27116, and write them into INSTB 20262. When INSTB 20262 is full, PREF 20260 stops fetching SINs until Namespace parsing operations, described below, have processed part of the contents of INSTB 20262, thereby creating space for more SINs.

The fetching operation is automatic, and requires intervention from Namespace only when a SIN causes a branch, i.e., causes the next SIN to be executed to be some other SIN than the one immediately following the current SIN. On a branch, Namespace must load PREF 20260 with the location of the next SIN to be executed and cause PREF 20260 to begin fetching SINs at that location. The operation which does this is specified by the load-prefetch-for-branch microcommand. The microcommand specifies a source for a Logical Descriptor 27116 and transfers that Logical Descriptor 27116 via DB 27021 to PREF 20260. After PREF 20260 has thus been loaded, it begins fetching SINs at the specified location. Since any SINs still in INSTB 20262 have been rendered meaningless by the branch operation, the first SINs loaded into INSTB 20262 are simply written over INSTB 20262's prior contents. Fig. 274 contains an example of the use of the load-prefetch-for-branch microcommand.

h.h. Parsing the I-Stream

The I-stream as fetched from MEM 10112 and stored in INSTB 20262 is a sequence of SOPs and Names. As already mentioned, the I-stream has a fixed format: in the present embodiment, SOPs are always 8 bits long, and Names may be 8, 12, or 16 bits long. The length of Names used in a given procedure is fixed, and is indicated by the value in K Field 30305 in the Procedure 602's PED 30303. The Namespace parsing operations obtain the SOPs and Names from the I-stream and place them on NAME Bus 20224. The SOPs are transferred via this bus to the devices in SOP Decoder 27003, while the Names are transferred to Name Trap 20254 and Name Cache 10226 for Resolve and Evaluation operations as described above. As the parsing operations obtain SOPs and Names, they also update the three program counters CPC 20270, EPC 20274, and IPC 20272. The values in these three counters are offsets from PBP which point to locations in Code 10344 belonging to Procedure 602 being executed. CPC 20270 points to the I-stream syllable currently being parsed, so it is updated on every parsing operation. EPC 20274 points to the beginning of the last SIN executed by JP 10114, and IPC 20272 points to the beginning of the current SIN, so these program counters are changed only at the beginning of the execution of an SIN, i.e., when a SOP is parsed.

As described in the discussion of FU 10120 hardware, in the current implementation, parsing consists physically of reading 8 or 16 bits of data from a location in INSTB 20262 identified by a pointer for INSTB 20262 which is accessible only to the hardware. As data is read, the hardware increments the pointer by the number of bits read, wrapping around and returning to the beginning of INSTB 20262 if it reaches the end. At the same time that the hardware increments the pointer, it increments CPC 20270 by the same number of bits. As previously mentioned, CPC 20270 contains the offset from PBP of the SOP or Name being currently parsed, thus coordinating the reading of INSTB 20262 with the reading of Procedure 602's Code 10344.

The number of bits read depends on whether Parser 20264 is reading an SOP or a Name, and in the latter case, by the syllable size specified for the Name. The syllable size is contained in CSSR 24112. On a Call to a Procedure 602 which has a different PED 30303 from that of the calling procedure, the Call microcode loads the value contained in K Field 30305 into CSSR 24112.

Namespace's parsing operations are performed by separate microcommands for parsing SOPs and Names. There is a single microcommand for parsing S-operations: parse-op-stage. The microcommand obtains the next eight bits from INSTB 20262, places the bits onto NAME Bus 20224, and latches them into LOPCODE Register 24212. It also updates EPC 20274 and IPC 20272 as required at the beginning of an SIN: EPC 20274 is set to IPC 20272's former value, and IPC 20272 is set to CPC 20270's value. At the end of the operation, CPC 20270 is incremented by 8. Since the parsing of an SOP always occurs as the first operation in the interpretation of an SIN, the parse-op-stage command is generally combined with a dispatch fetch command. As will be explained below, the latter command interprets the S-operation as an address in FDISP 24218, and FDISP 24218 in turn produces an address in FUSITT 11012. The latter address is the location of the beginning of the SIN microcode for the SIN.

There are two microcommands for parsing Names:
parse_k_load_epc and parse_k_dispatch_ebox. Both commands obtain a number of bits from INSTB 20262 and place them on NAME Bus 20214. With both microcommands, the syllable size, K, stored in CSSR 24112, determines the number of bits obtained from INSTB 20262. Both commands also increment CPC by the value stored in CSSR 24112. In addition, parse_k_load_epc sets EPC to IPC's value, while parse_k_dispatch_ebox also dispatches EU 10122, i.e., interprets the SOP saved in LOPCODE 24210 as an address in EDISP 24222, which in turn contains an address in EU EUSITT 20344. The EU EUSITT 20344 address is passed via EUDIS Bus 20206 to COMQ 20342 in EU 10122.

c. The S-Interpreters (Fig. 307)

CS 10110 does not assign fixed meanings to SOPs. While all SOPs are 8 bits long, a given 8 bit SOP may have one meaning in one S-Language and a completely different meaning in another S-Language. The semantics of an S-Language's S-operations are determined completely by the S-interpreter for the S-Language. Thus, in order to correctly interpret an S-operation, CS 10110 must know what S-interpreter it is to use. The S-interpreter is identified by a UID pointer with offset 0 in SIP Field 30309 of PED 30303 for

Procedure 602 that CS 10110 is currently executing. In the present embodiment, the UID is the UID of a microcode object which contains FU 10120 microcode. When loaded into FUSITT 11012, the microcode interprets SOPs as defined by the S-Language to which the SOP belongs. In other embodiments, the UID may be the UID of a Procedure Object 608 containing Procedures 602 which interpret the S-Language's SOPs, and in still others, the S-interpreter may be contained in a PROM and the S-interpreter UID may not specify an object, but may serve solely to identify the S-interpreter.

When a Procedure 602 executes an SIN on JP 10114, CS 10110 must translate the value of SIP Pointer 30309 for Procedure 602 and the S-instruction's SOP into a location in the microcode or high-level language code which makes up the S-interpreter. The location obtained by the translation is the beginning of the microcode or high-level language code which implements the SIN. The translation of an SOP together with SIP Pointer 30309 into a location in the S-interpreter is termed dispatching. Dispatching in the present embodiment involves two primary components: a table in memory which translates the value of SIP Pointer 30309 into a small integer called the Dialect Number, and S-operation Decoder Portion 27003 of the FU 10120 micromachine. The following discussion will first present the table and explain how an SIP Pointer 30309 is translated into a Dialect Number, and then explain how the Dialect Number and the SOP together are translated into locations in FUSITT 11012 and EUSITT 20344.

1. Translating SIP into a Dialect Number (Fig. 307)

In the present embodiment, all S-interpreters in CS 10110 are loaded into FUSITT 11012 when CS 10110 begins operation and each S-interpreter is always placed in the same location. Which S-interpreter is used to interpret an S-Language is determined by a value stored in dialect register RDIAL 24212. Consequently, in the present embodiment, a Call to a Procedure 602 whose S-interpreter differs from that of the calling Procedure 602 must translate the UID pointer contained in SIP Field 30309 into a Dialect Number.

Fig. 307 represents the table and microcode which performs this translation in the present embodiment. S-interpreter Translation Table (STT) 30701 is a table which is indexed by small AONs. Each STT Entry (STTE) 30703 has two fields: an AON Field 30705 and a Dialect Number Field 30709. Dialect Number Field 30709 contains the Dialect Number for the S-interpreter object whose AON is in AON Field 30705.

When CS 10110 begins operation, each S-interpreter object is wired active and assigned an AON small enough to serve as an index in STT 30701. By convention, a given S-interpreter object is always assigned the same AON and the same Dialect Number. The AON is placed in AON Field 30705 of STTE 30703 indexed by the AON, and the Dialect Number is placed in Dialect Number Field 30709. Since the S-interpreter objects are wired active, these AONs will never be reassigned to other objects.

On a Call which requires a new S-interpreter, Call microcode obtains the new SIP from SIP Field 30309, calls KOS LAR microcode to translate its UID to its AON, uses the AON to locate the S-interpreter's STTE 30703, and places the value of Dialect Number Field 30709 into RDIAL 21242.

Other embodiments may allow S-interpreters to be loaded into FUSITT 11012 at times other than system initialization, and allow S-interpreters to occupy different locations in FUSITT 11012 at different times. In these embodiments, STT 30701 may be implemented in a manner similar to the implementations of AST 10914 or MHT 10716 in the present embodiment.

2. Dispatching

Dispatching is accomplished by Dispatch Files 27004. These files translate the values provided by RDIAL 24212 and the SOP of the S-instruction being executed into the location of microcode for the SIN specified by the S-operation in the S-interpreter specified by the value of RDIAL 24212. The present embodiment has three dispatch files: FDISP 24218, FALG 24220, and EDISP 24222. FDISP 24218 and FALG 24220 translate S-operations into locations of microcode which executes on FU 10120; EDISP 24222 translates S-operations into locations of microcode which executes on EU 10122. The difference between FDISP 24218 and FALG 24220 is one of speed: FDISP 24218 can translate an SOP in the same microinstruction which performs a parse_op_stage command to load the SOP into LOPCODE 24210. FALG 24220 must perform the translation on a cycle following the one in which the SOP is loaded into LOPCODE 24210. Typically, the location of the first portion of the microcode to execute an S-operation is contained in an FDISP 24218 register, the location of portions executed later is contained in an FALG 24220 register, and the location of microcode for the S-operation which executes on EU 10122 is contained in EDISP 24222.

In the present embodiment, the registers accomplish the translation from S-operation to microcode location as follows: As mentioned in the discussion of FU 10120 hardware, each Dispatch File contains 1024 registers. Each register may contain an address in an S-interpreter. As will be seen in detail later, the address may be an address in an S-interpreter's object, or it may be the address in FUSITT 11012 or EUSITT 20344 of a copy of microcode stored at an S-interpreter address. The registers in the Dispatch Files may be divided into sets of 128 or 256 registers. Each set of registers translates the SOPs for a single S-Language into locations in microcode. Which set of registers is used to interpret a given S-operation is decided by the value of RDIAL 24212; which register in the set is used is determined by the value of the S-operation. The value contained in the specified register is then the location of microcode which executes the S-instruction specified by the S-operation in the S-Language specified by RDIAL 24212.

Logically, the register addressed by the concatenated value in turn contains a 15 bit address which is

the location in the S-interpreter of the first microinstruction of microcode used to execute the S-instruction specified by the S-operation in the S-Language specified by the contents of RDIAL 24212. In the present embodiment, the microcode referred to by the address may have been loaded into FUSITT 11012 and EUSITT 20344 or it may be available only in memory. Addresses of microcode located in FUSITT 11012 and EUSITT 20344 are only eight bits long. Consequently, if a Dispatch File 27004 contains an address which requires more bits than that, the microcode specified by the address is in memory. As described in the discussion of FU 10112 hardware, addresses larger than 8 bits produce an Event Signal, and microcode invoked by the event signal fetches the microinstruction at the specified address in the S-interpreter from memory and loads it into location 0 of FUSITT 11012. The event microcode then returns, and the microinstruction at location 0 is executed. If the next microinstruction also has an address larger than 8 bits, the event signal occurs again and the process described above is repeated.

As previously mentioned, FDISP 24218 is faster than FALG 24220. The reason for the difference in speed is that FDISP registers contain only 6 bits for addressing the S-interpreter. The present embodiment assumes that all microcode addressed via FDISP 24218 is contained in FUSITT 11012. It concatenates 2 zero bits with the six bits in the FDISP 24218 register to produce an 8 bit address for FUSITT 11012. FDISP 24218 registers can thus contain the location of every fourth FUSITT 11012 register between FUSITT register 256 and FUSITT register 448. The microcode loaded into these locations in FUSITT 11012 is microcode for operations which are performed at the start of the SIN by many different SINs. For example, all SINs which perform operations on 2 operands and assign the result to a location specified by a third operand must parse and evaluate the first two operands and parse and resolve the third operand. Only after these operations are done are SINs-specific operations performed. In the present embodiment, the microcode which parses, resolves, and evaluates the operands is contained in a part of FUSITT 11012 which is addressable by FDISP 24218.

As previously mentioned, in the present embodiment, FUSITT 11012 and EUSITT 20344 may be loaded only when CS 10110 is initialized. The microcode loaded into FUSITT 11012 and EUSITT 20344 is produced by the microbinder from the microcode for the various SINs. To achieve efficient use of FUSITT 11012 and EUSITT 20344, microcode for operations shared by various S-interpreters appears only once in FUSITT 11012 and EUSITT 20344. While the SINs in different S-Languages which share the microcode have different registers in FDISP 24218, FALG 24220, or EDISP 24222 as the case may be, the registers for each of the S-instructions contain the same location in FUSITT 11012 or EUSITT 20344.

4. The Kernel Operating System

A. Introduction

Many of the unique properties of CS 10110 are produced by the manipulation of tables in MEM 10112 and Secondary Storage 10124 by programs executing on JP 10114. These programs and tables together make up the Kernel Operating System (KOS). Having described CS 10110's components and the means by which they cooperate to execute computer programs, this specification now presents a detailed account of KOS and of the properties of CS 10110 which it produces. The discussion begins with a general introduction to operating systems, then presents an overview of CS 10110's operating systems, an overview of the KOS, and detailed discussions of the implementation of objects, access control, and Processes 610.

a. Operating Systems (Fig. 401)

In CS 10110, as in other computer systems, the operating system has two functions:

— It controls the use of CS 10110 resources such as JP 10114, MEM 10112, and devices in IOS 10116 by programs being executed on CS 10110.

— It defines how CS 10110 resources appear to users of CS 10110.

The second function is a consequence of the first: By controlling the manner in which executing programs use system resources, the operating system in fact determines how the system appears to its users. Figure 401 is a schematic representation of the relationship between User 40101, Operating System 40102, and System Resources 40103. When User 40101 wishes to use a System Resource 40103, User 40101 requests the use of System Resource 40103 from Operating System 40102, and Operating System 40102 in turn commands CS 10110 to provide the requested Resources 40103. For example, when a user program wishes to use a peripheral device, it does not deal with the device directly, but instead calls the Operating System 40102 procedure 602 that controls the device. While Operating System 40102 must take into account the device's complicated physical properties, the user program that requested the device need know nothing about the physical properties, but must only know what information the Operating System 40102 Procedure 602 requires to perform the operation requested by the user program. For example, while the peripheral device may require that a precise pattern of data be presented to it, the Operating System 40102 procedure 602 may only require the data itself from the user program, and may format the data as required by the peripheral device. The Operating System 40102 Procedure 602 that controls the peripheral device thus transforms a complicated physical interface to the device into a much simpler logical interface.

1. Resources Controlled by Operating Systems (Fig. 402)

Operating Systems 40102 control two kinds of resources: physical resources and virtual resources. The physical resources in the present embodiment of CS 10110 are JP 10114, IOS 10116 and the peripheral

devices associated with IOS 10116, MEM 10112, and Secondary Storage 10124. Virtual resources are resources that the operating system itself defines for users of CS 10110. As was explained above, in controlling how CS 10110's resources are used, Operating System 40102 defines how CS 10110 appears to the users. Instead of the physical resources controlled by Operating System 40102, the user sees a far simpler set of virtual resources. The logical I/O device interface that Operating System 40102 gives the user of a physical I/O device is such a virtual resource. Often, an Operating System 40102 will define sets of virtual resources and multiplex the physical resources among these virtual resources. For instance, Operating System 40102 may define a set of Virtual Processors 612 that correspond to a smaller group of physical processors, and a set of virtual memories that correspond to a smaller group of physical resources. When a user executes a program, it runs on a Virtual Processor 612 and uses virtual memory. It seems to the user of the virtual processor and the virtual memory that he has sole access to a physical processor and physical memory, but in fact, Operating System 40102 is multiplexing the physical processors and memories among the Virtual Processors 612 and virtual memories.

Operating System 40102, too, uses virtual resources. For instance, the memory management portion of an Operating System 40102 may use I/O devices; when it does so, it uses the virtual I/O devices defined by the portion of the Operating System 40102 that manages the I/O devices. One part of Operating System 40102 may also redefine virtual resources defined by other parts of Operating System 40102. For instance, one part of Operating System 40102 may define a set of primitive virtual I/O devices and another part may use these primitive virtual I/O devices to define a set of high-level user-oriented I/O devices. Operating System 40102 thus turns the physical CS 10110 into a hierarchy of virtual resources. How a user of CS 10110 perceives CS 10110 depends entirely on the level at which he is dealing with the virtual resources.

The entity that uses the resources defined by Operating System 40102 is the process. A Process 610 may be defined as the activity resulting from the execution of a program with its data by a sequential processor. Whenever a user requests the execution of a program on CS 10110, Operating System 40102 creates a Process 610 which then executes the Procedures 602 making up the user's program. In physical terms, a process 610 is a set of data bases in memory that contain the current state of the program execution that the process represents. Operating System 40102 causes Process 610 to execute the program by giving Process 610 access to the virtual resources which it requires to execute the program, by giving the virtual resources access to those parts of Process 610's state which they require to perform their operations, and by giving these virtual resources access to the physical resources. The temporary relationship of one resource to another or of a Process 610 to a resource is called a binding. When a Process 610 has access to a given Virtual Processor 612 and Virtual processor 612 has access to process 610's state, process 610 is bound to Virtual Processor 612, and when Virtual Processor 612 has access to JP 10114 and Virtual Processor 612's state is loaded into JP 10114 registers, Virtual processor 612 is bound to JP 10114, and JP 10114 can execute SINs contained in Procedures 602 in the program being executed by Process 610 bound to Virtual Processor 612. Binding and unbinding may occur many times in the course of the execution of a program by a Process 610. For instance, If a Process 610 executes a reference to data and the data is not present in MEM 10112, then Operating System 40102 unbinds Process 610's Virtual Processor 612 from JP 10114 until the data is available in MEM 10112. If the data is not available for an extended period of time, or if the user for whom Process 610 is executing the program wishes to stop the execution of the program for a while, Operating System 40102 may unbind process 610 from its Virtual Processor 612. Virtual Processor 612 is then available for use by other Processes 610.

As mentioned above, the binding process involves giving a first resource access to a second resource, and using the first resource's state in the second resource. To permit binding and unbinding, Operating System 40102 maintains data bases that contain the current state of each resource and each Process 610. State may be defined as the information that the operating system must have to use the resource or execute the Process 610. The state of a line printer, for instance, may be variables that indicate whether the line printer is busy, free, off line, or out of order. A Process 610's state is more involved, since it must contain enough information to allow Operating System 40102 to bind Process 610 to a Virtual Processor 612, execute Process 610 for a while, unbind Process 610, and then rebind it and continue execution where it was halted. A process 610's state thus includes all of the data used by Process 610 up to the time that it was unbound from a Virtual Processor 612, along with information indicating whether Process 610 is ready to begin executing again.

Figure 402 shows the relationship between Processes 610, virtual, and physical resources in an operating system. The figure shows a multi-process Operating System 40102, that is, one that can multiplex CS 10110 resources among several Processes 610. The Processes 610 thus appear to be executing concurrently. The solid arrows in Figure 402 indicate bindings between virtual resources or between virtual and physical resources. Each Process 610 is created by Operating System 40102 to execute a user program. The program consists of Procedures 602, and Process 610 executes Procedures 602 in the order prescribed by the program. Processes 610 are created and managed by a component of Operating System 40102 called the Process Manager. Process Manager 40203 executes a Process 610 by binding it to a Virtual Processor 612. There may be more Processes 610 than there are Virtual Processors 612. In this case, Operating System 40102 multiplexes Virtual Processors 612 among Processes 610.

Virtual Processors 612 are created and made available by another component of Operating System 40102, Virtual Processor Manager 40205. Virtual Processor Manager 40205 also multiplexes JP 10114

among Virtual Processors 612. If a Virtual Processor 612 is ready to run, Virtual Processor Manager 40205 binds it to JP 10114. When Virtual Processor 612 can run no longer, or when another Virtual Processor 612 requires JP 10114, Virtual Processor Manager 40205 unbinds running Virtual Processor 612 from JP 10114 and binds another Virtual Processor 612 to it.

Virtual Processors 612 use virtual memory and I/O resources to perform memory access and input-output. Virtual Memory 40206 is created and managed by Virtual Memory Manager 40207, and Virtual I/O Devices 40208 are created and managed by Virtual I/O Manager 40209. Like Virtual Processor Manager 40205, Components 40207 and 40209 of Operating System 40102 multiplex physical resources among the virtual resources. As described above, one set of virtual resources may use another set. One way in which this can happen is indicated by the broken arrows in Figure 402. These arrows show a binding between Virtual Memory 40206 and Virtual I/O Device 40208. This binding occurs when Virtual Memory 40206 must handle a reference to data contained on a peripheral device such as a disk drive. To the user of Virtual Memory 40206, all data appears to be available in MEM 10110. In fact, however, the data is stored on peripheral devices such as disk drives, and copied into MEM 10112 when required. When a Process 610 references data that has not been copied into MEM 10112, Virtual Memory 40206 must use IOS 10116 to copy the data into MEM 10112. In order to do this, it uses a Virtual I/O Device 40208 provided by Virtual I/O Manager 40209.

b. The Operating System in CS 10110

For the sake of clarity, Operating System 40102 has been described as though it existed outside of CS 10110. In fact, however, Operating System 40102 itself uses the resources it controls. In the present embodiment, parts of Operating System 40102 are embodied in JP 10114 hardware devices, parts are embodied in microcode which executes on JP 10114, and parts are embodied in Procedures 602. These Procedures 602 are sometimes called by Processes 610 executing user programs, and sometimes by special Operating System Processes 610 which do nothing but execute operations for Operating System 40102.

The manner in which the components of Operating System 40102 interact may be illustrated by the way in which CS 10110 handles a page fault, i.e., a reference to data which is not available in MEM 10110. The first indication that there may be a page fault is an ATU Miss Event Signal. This Event Signal is generated by ATU 10228 in FU 10120 when there is no entry in ATU 10228 for a Logical Descriptor 27116 used in a read or write operation. The Event Signal invokes Operating System 40102 microcode, which examines a table in MEM 10112 in order to find whether the data described by Logical Descriptor 27116 has a copy in MEM 10112. If the table indicates that there is no copy, Operating System 40102 microcode communicates the fact of the page fault to an Operating System 40102 Virtual Memory Manager process 610 and removes Virtual Processor 612 bound to the Process 610 which was executing when the page fault occurred from JP 10114. Some time later, Virtual Memory Manager Process 610 is bound to JP 10114. Procedures 602 executed by Virtual Memory Manager Process 610 then initiate the I/O operations required to locate the desired data in Secondary Storage 10124 and copy it into MEM 10112. When the data is available in MEM 10112, Operating System 40102 allows Virtual Processor 612 bound to Process 610 which was executing when the page fault occurred to return to JP 10114. Virtual Processor 612 repeats the memory reference which caused the page fault, and since the data is now in MEM 10112, the reference succeeds and execution of Process 610 continues.

c. Extended Operating System and the Kernel Operating System (Fig. 403)

In CS 10110, Operating System 40102 is made up of two component operating systems, the Extended Operating System (EOS) and the Kernel Operating System (KOS). The KOS has direct access to the physical resources. It defines a set of primitive virtual resources and multiplexes the physical resources among the primitive virtual resources. The EOS has access to the primitive virtual resources defined by KOS, but not to the physical resources. The EOS defines a set of user-level virtual resources and multiplexes the primitive virtual resources defined by KOS among the user level virtual resources. For example, KOS provides EOS with Processes 610 and Virtual processors 612 and binds Virtual Processors 612 to JP 10114, but EOS decides when a Process 610 is to be created and when a process 610 is to be bound to a Virtual processor 612.

Figure 403 shows the relationship between a user Process 610, EOS, KOS, and the physical resources in CS 10110. Figure 403 shows three levels of interface between executing user Process 610 and JP 10114. The highest level of interface is Procedure Level 40302. At this level, Process 610 interacts with CS 10110 by calling Procedures 602 as specified by the program Process 610 is executing. The calls may be either calls to User Procedures 40306 or calls to EOS Procedures 40307. When Process 610 is executing a procedure 602, Process 610 produces a stream of SINs. The stream contains two kinds of SINs, S-language SINs 40310 and KOS SINs 40311. Both kinds of SINs interact with CS 10110 at the next level of interface, SIN-level Interface 40309. SINs 40310 and 40311 are interpreted by Microcode 40312 and 40313, and Microinstructions 40315 interact with CS 10110 at the lowest level of interface, JP 10114 Interface 40316. As already explained in the discussion of the FU 10120 micromachine, certain conditions in JP 10114 result in

Event Signals 40314 which invoke microroutines in S-interpreter Microcode 40312 or KOS Microcode 40313. Only Procedure-Level Interface 40302 and SIN-level Interface 40309 are visible to users. Procedure-level Interface 40309 appears as calls in user Procedures 602 or as statements in user Procedures 602 which compilers translate into calls to EOS procedures 602. SIN-level Interface 40309 appears as the Name Tables 10335 and SINs in Procedure Objects 608 generated by compilers.

As Figure 403 indicates, EOS exists only at Procedural Level 40302, while KOS exists at Procedural Level 40302, and SIN Level 40304, and within the microcode beneath SIN Level 40309. The only portion of the operating system that is directly available to user Processes 610 is EOS Procedures 40307. EOS Procedures 40307 may in turn call KOS procedures 40308. In many cases, an EOS Procedure 40307 will contain nothing more than the call to a KOS Procedure 40308.

User Procedures 40306, EOS Procedures 40307, and KOS Procedures 40308 all contain S-language SINs 40310. In addition, KOS Procedures 40308 only may contain special KOS SINs 40311. Special KOS SINs 40311 control functions that are not available to EOS Procedures 40307 or User Procedures 40306, and KOS SINs 40311 may therefore not appear in Procedures 40306 or 40307. S-language SINs 40310 are interpreted by S-interpreter Microcode 40312, while KOS SINs 40311 are interpreted by KOS Microcode 40313. KOS Microcode 40313 may also be called by S-interpreter Microcode 40313. Depending on the hardware conditions that cause Event Signals 40314, Signals 40314 may cause the execution of either S-interpreter Microcode 40312 or KOS Microcode 40313.

Figure 403 shows the system as it is executing a user Process 610. There are in addition special Processes 610 reserved for KOS and EOS use. These Processes 610 work like user Processes 610, but carry out operating system functions such as process management and virtual memory management. With one exception, EOS Processes 610 call EOS Procedures 40307 and KOS Procedures 40308, while KOS Processes 610 call only KOS Procedures 40308. The exception is the beginning of Process 610 execution: KOS performs the KOS-level functions required to begin executing a Process 610 and then calls EOS. EOS performs the required EOS level functions and then calls the first User Procedure 40306 in the program Process 610 is executing.

A description of how KOS handles page faults can serve to show how the parts of the system at the JP 10114—, SIN—, and procedure Levels work together. A page fault occurs when a Process 610 references a data item that has no copy in MEM 10112. The page fault begins as an Event Signal from ATU 10228. The Event Signal invokes a microroutine in KOS Microcode 40313. If the microroutine confirms that the referenced data item is not in MEM 10112, it records the fact of the page fault in some KOS tables in MEM 10112 and calls another KOS microroutine that unbinds Virtual Processor 612 bound to Process 610 that caused the page fault from JP 10114 and allows another Process 610's Virtual Processor 612 to run. Some time after the page fault, a special operating system Process 610, the Virtual Memory Manager Process 610, runs and executes KOS Procedures 40309. Virtual Memory Manager Process 610 initiates the I/O operation that reads the data from Secondary Storage 10124 into MEM 10112. When IOS 10116 has finished the operation, Process 610 that caused the page fault can run again and Virtual Memory Manager Process 610 performs an operation which causes Process 610's Virtual Processor 612 to again be bound to JP 10114. When Process 610 resumes execution, it again attempts to reference the data. The data is now in MEM 10112 and consequently, the page fault does not recur.

The division of Operating System 40102 into two hierarchically-related operating systems is characteristic for CS 10110. Several advantages are gained by such a division:

- Each of the two operating systems is simpler than a single operating system would be. EOS can concern itself mainly with resource allocation policy and high-level virtual resources, while KOS can concern itself with low-level virtual resources and hardware control.
- Because each operating system is simpler, it is easier to verify that each system's components are performing correctly, and the two systems are therefore more dependable than a single system.
- Dividing Operating System 40102 makes it easier to implement different embodiments of CS 10110. Only the interface provided by EOS is visible to the user, and consequently, the user interface to the system can be changed without altering KOS. In fact, a single CS 10110 may have a number of EOSs, and thereby present different interfaces to different users. Similarly, changes in the hardware affect the implementation of the KOS, but not the interface that KOS provides EOS. A given EOS can therefore run on more than one embodiment of CS 10110.
- A divided operating system is more secure than a single operating system. Physical access to JP 10114 is provided solely by KOS, and consequently, KOS can ensure that users manipulate only those resources to which they have access rights.

All CSs 10110 will have the virtual resources defined by KOS, while the resources defined by EOS will vary from one CS 10110 to another and even within a single CS 10110. Consequently, the remainder of the discussion will concern itself with KOS.

The relationship between the KOS and the rest of CS 10110 is governed by four principles:

- Only the KOS has access to the resources it controls. User calls to EOS may result in EOS calls to KOS, and S-language SINs may result in invocations of KOS microcode routines, but neither EOS nor user programs may directly manipulate resources controlled by KOS.
- The KOS is passive. It responds to calls from the EOS, to microcode invocations, and to Event Signals, but it initiates no action on its own.

— The KOS is invisible to all system users but the EOS. KOS does not affect the logical behavior of a Process 610 and is noticeable to users only with regard to the speed with which a Process 610 executes on CS 10110.

As discussed above, KOS manages both physical and virtual resources. The physical resources and some of the virtual resources are visible only within KOS; others of the virtual resources are provided to EOS. Each virtual resource has two main parts: a set of data bases that contain the virtual resource's state, and a set of routines that manipulate the virtual resource. The set of routines for a virtual resource are termed the resource's manager. The routines may be KOS Procedures 40308, or they may be KOS Microcode 40313. As mentioned, in some cases, KOS uses separate Processes 610 to manage the resources.

For the purposes of this specification, the resources managed by KOS fall into two main groups: those associated with objects, and those associated with Processes 610. In the following, first those resources associated with objects, and then those associated with Processes 610 are discussed.

B. Objects and Object Management (Fig. 404)

The virtual resources termed objects are defined by KOS and manipulated by EOS and KOS. Objects as seen by EOS have five properties:

— A single UID that identifies the object throughout the object's life and specifies what Logical Allocation Unit (LAU) the object belongs to.
— A set of attributes that describe the object and limit access to it.
— Bit-addressable contents. I the present embodiment, the contents may range from 0 to (2**32) — 1 bits in length. Any bit in the contents may be addressed by an offset.
— Objects may be created.
— Objects may be destroyed.

All of the data and Procedures 602 in a CS 10110 are contained in objects. Any process 610 executing on a CS 10110 may use a UID-off set address to attempt to access data or Procedures 602 in certain objects on any CS 10110 accessible to the CS 10110 on which Process 610 is executing. The objects which may be thus accessed by any Process 610 are those having UIDs which are guaranteed unique for all present and future CS 10110. Objects with such unique UIDs thus form a single address space which is at least potentially accessible to any process 610 executing on any CS 10110. As will be explained in detail later, whether a Process 610 can in fact access an object in this single address space depends on whether Process 610 has access rights to the object. Other objects, whose UIDs are not unique, may be accessed only by Processes 610 executing on CSs 10110 or groups of CSs 10110 for which the non-unique UID is in fact unique. No two objects accessible to a CS 10110 at a given time may have identical UIDs.

The following discussion of objects will first deal with objects as they are seen directly by EOS and indirectly by user programs, and then deal with objects as they appear to KOS.

Figure 404 illustrates how objects appear to EOS. The object has three parts: the UID 40401, the Attributes 40404, and the Contents, 40406. The object's contents reside in a Logical Allocation Unit (LAU), 40405. UID 40401 has two parts: a LAU Identifier (LAUID) 40402 that indicates what LAU 40405 the object is on, and the Object Serial Number (OSN) 40403, which specifies the object in LAU 40405.

The EOS can create an object on a LAU 40405, and given the object's UID 40401, can destroy the object. In addition, EOS can read and change an object's Attributes 40404. Any Process 610 executing on a CS 10110 may reference information in an object by specifying the object's UID 40401 and the bit in the object at which the information begins. At the highest level, addresses in CS 10110 thus consist of a UID 40401 specifying an object and an offset specifying the number of bits into the object at which the information begins. As will be explained in detail below, KOS translates such UID-offset addresses into intermediate forms called AON-offset addresses for use in JP 10114 and into page number-displacement addresses for use in referencing information which has been copied into MEM 10112.

The physical implementation and manipulation of objects is restricted solely to KOS. For instance, objects and their attributes are in fact stored in Secondary Storage 10124. When a program references a portion of an object, KOS copies that portion of the object from Secondary Storage 10124 into MEM 10112, and if the portion in MEM 10112 is changed, updates the copy of the object in Secondary Storage 10124. EOS and user programs cannot control the location of an object in Secondary Storage 10124 or the location of the copy of a portion of an object in MEM 10112, and therefore can access the object only by means of KOS.

While EOS cannot control the physical implementation of an object, it can provide KOS with information that allows KOS to manage objects more effectively. Such information is termed hints. For instance, KOS generally copies a portion of an object into MEM 10112 only if a Process 610 references information in the object. However, EOS schedules Process 610 execution, and therefore can predict that certain objects will be required in the near future. EOS can pass this information on to KOS, and KOS can use the information to decide what portions of objects to copy into MEM 10112.

a. Objects and User Programs (fig. 405)

As stated above, user programs manipulate objects, but the objects are generally not directly visible to user programs. Instead, user programs use symbols such as variable names or other references to refer to

data stored in objects or file names to refer to the objects themselves. The discussion of Namespace has already illustrated how CS 10110 compilers translate variable names appearing in statements in Procedures 602 into Names, i.e., indexes of NTEs 30401, how Name Resolve microcode resolves NTE 30401 into Logical Descriptors 27116, and how ATU 10228 translates Logical Descriptors 27116 into locations in MEM 10112 containing copies of the portions of the objects in which the data represented by the variables resides.

The translation of filenames to UIDs 40401 is accomplished by EOS. EOS maintains a filename translation table which establishes a relationship between a system filename called a pathname and the UID 40401 of the object containing the file's data, and thereby associates the pathname with the object. A Pathname is a sequence of ASCII characters which identifies a file to a user of CS 10110. Each pathname in a given CS 10110 must be unique. Figure 405 shows the filename translation table. Referring to that figure, when a user gives pathname 40501 to the EOS, EOS uses Filename Translation Table 40503 to translate pathname 40501 into UID 40401 for object 40504 containing the file. An object in CS 10110 may thus be identified in two ways: by means of its UID 40401 or by means of a Pathname 40501. While an object has only a single UID 40401 throughout its life, the object may have many Pathnames 40501. All that is required to change an object's pathname 40501 is the substitution of one Pathname 40501 for another in the object's Entry 40502 in Filename Translation Table 40503. One consequence of the fact that an object may have different Pathnames 40501 during its life is that when a program uses a Pathname 40501 to identify an object, a user of CS 10110 may make the program process a different object simply by giving the object which formerly had Pathname 40501 which appears in the program a new Pathname 40501 and giving the next object to be processed the Pathname 40501 which appears in the program.

In the present embodiment, an object may contain only a single file, and consequently, a Pathname 40501 always refers to an entire object. In other embodiments, a Pathname 40501 may refer to a portion of an object, and in such embodiments, Filename Translation Table 40503 will associate a Pathname 40501 with a UID-offset address specifying the beginning of the file.

b. UIDs 40401 (Fig. 406)

UIDs 40401 may identify objects and other entities in CS 10110. Any entity identified by a UID 40401 has only a single UID throughout its life. Figure 406 is a detailed representation of a CS 10110 UID 40401. UID 40401 is 80 bits long, and has two fields. Field 40402, 32 bits long, is the Logical Allocation Unit Identifier (LAUID). It specifies LAU 40405 containing the object. LAUID 40402 is further subdivided into two subfields: LAU Group Number (LAUGN) 40607 and LAU Serial Number (LAUSN) 40605. LAUGN 40607 specifies a group of LAUs 40405, and LAUSN 40605 specifies a LAU 40405 in that group. Purchasers of CS 10110 may obtain LAUGNs 40607 from the manufacturer. The manufacturer guarantees that he will assign LAUGN 40607 given the purchaser to no other CS 10110, and thus these LAUGNs 40607 may be used to form UIDs 40401 which will be unique for all CSs 10110. Field 40604, 48 bits long, is the Object Serial Number (OSN). It specifies the object in LAU 40405.

UIDs 40401 are generated by KOS Procedures 602.

There are two such procedures 602, one which generates UIDs 40401 which identify objects, and another which generates UIDs 40401 which identify other entities in CS 10110. The former Procedure 602 is called Generate Object UID, and the latter Generate Non-object UID. The Generate Object UID Procedure 602 is called only by the KOS Create Object Procedure 602. Create Object Procedure 602 provides Generate Object UID Procedure 602 with a LAUID 40402, and Generate Object UID Procedure 602 returns a UID 40401 for the object. In the present embodiment, UID 40401 is formed by taking the current value of the architectural clock, contained in a location in MEM 10112, forming an OSN 40403 from the architectural clock's current value, and concatenating OSN 40403 to LAUID 40402.

Generate Non-object UID Procedure 602 may be invoked by EOS to provide a UID 40401 which does not specify an object. Non-object UIDs 40401 may be used in CS 10110 wherever a unique label is required. For example, as will be explained in detail later, all Virtual processors 612 which are available to CS 10110 have non-object UIDs 40401. All such non-object UIDs 40401 have a single LAUSN 40607, and thus, EOS need only provide a LAUGN 40605 as an argument. Generate Non-object UID Procedure 602 concatenates LAUGN 40605 with the special LAUSN 40607, and LAUID 40402 thus produced with an OSN 40403 obtained from the architectural clock. In other embodiments, OSNs 40403 for both object and non-object UIDs 40401 may be generated by other means, such as counters.

CS 10110 also has a special UID 40401 called the Null UID 40401. The Null UID 40401 contains nothing but 0 bits, and is used in situations which require a UID value which cannot represent an entity in CS 10110.

c. Object Attributes

What a program can do with an object is determined by the object's Attributes 40404. There are two kinds of Attributes 40404: Object Attributes and Control Attributes. Object Attributes describe the object's contents; Control Attributes control access to the object. Objects may have Attributes 40404 even though they have no Contents 40406, and in some cases, objects may even exist solely for their Attributes 40404.

For the purposes of this discussion, there are two kinds of Object Attributes: the Size Attribute and the Type Attributes.

An object's Size Attribute indicates the number of bits that the object currently contains. On each

reference to an object's Contents 40406, KOS checks to make sure that the data accessed does not extend beyond the end of the object. If it does, the reference is aborted.

The Type Attributes indicate what kind of information the object contains and how that information may be used. There are three categories of Type Attributes: the Primitive Type Attributes, the Extended Type Attribute, and the Domain of Execution attribute. An object's Primitive Type Attribute indicates whether the object is a data object, a Procedure Object 608, an Extended Type Manager, or an S-interpreter. As their names imply, data objects contain data and Procedure Objects 608 contain Procedures 602. Extended Type Managers (ETMs) are a special type of Procedure Object 608 whose Procedures 608 may perform operations solely on objects called Extended Type Objects. Extended Type Objects (ETOs) are objects which have an Extended Type Attribute in addition to their Primitive Type Attribute; for details, see the discussion of the Extended Type Attribute below. S-interpreters are objects that contain interpreters for S-languages. In the present embodiment, the interpreters consist of dispatch tables and microcode, but in other embodiments, the interpreters may themselves be written in high-level languages. Like the Length Attribute, the Primitive Type Attributes allow KOS to ensure that a program is using an object correctly. For instance, when the KOS executes a call for a Procedure 602 it checks whether the object specified by the call is a Procedure Object 608. If it is not, the call fails.

d. Attributes and Access Control

The remaining Object Attributes and the Control Attributes are all part of CS 10110's Access Control System. The Access Control System is discussed in detail later; here, it is dealt with only to the extent required for the discussion of objects. In CS 10110, an access of an object occurs when a Process 610 fetches SINs contained in a Procedure Object 608, reads data from an object, writes data to an object, or in some cases, when Process 610 transfers control to a Procedure 602. The Access Control System checks whether a Process 610 has the right to perform the access it is attempting. There are two kinds of access in CS 10110, Primitive Access and Extended Access. Primitive Access is access which the Access Control System checks on every reference to an object by a Process 610; Extended Access is access that is checked only on user request. Primitive access checks are performed on every object; extended access checks may be performed only on ETOs, and may be performed only by Procedures 602 contained in ETMs.

The means by which the Access Control System checks a Process 610's access to an object are Process 610's subject and the object's Access Control Lists (ACLs). Each Process 610 has a subject made up of four UIDs 40401. These UIDs 40401 specify the following:

- The user for whom Process 610 was created. This UID 40401 is termed the principal component of the subject.
- Process 610 itself. This UID 40401 is termed the process component.
- The domain in which Process 610 is currently executing. This UID 40401 is termed the domain component.
- A user-defined subgroup of subjects. This UID 40401 is termed the tag component.

A domain is a group of objects which may potentially be accessed by any Process 610 which is executing a Procedure 602 in one of a group of Procedure Objects 608 or ETMs. Each Procedure Object 608 or ETM has a Domain of Execution (DOE) Attribute. This attribute is a UID 40401, and while a Process 610 is executing a Procedure 602 in that Procedure Object 608 or ETM, the DOE attribute UID 40401 is the domain component in Process 610's subject. The DOE attribute thus defines a group of objects which may be accessed by a Process 610 executing Procedures 602 from Procedure Object 608. The group of objects is called Procedure Object 608's domain. As may be seen from the above definition, a subject's domain component may change on any call to or return from a Procedure 602. The tag component may change whenever the user desires. The principal component and the process component, on the other hand, do not change for the life of Process 610.

The ACLs which make up the other half of the Access Control System are attributes of objects. Each ACL consists of a series of Entries (ACLE), and each ACLE has two parts: a Subject Template and a set of Access Privileges. The Subject Template defines a group of subjects, and the set of Access Privileges define the kinds of access that subjects belonging to the group have to the object. To check whether an access to an object is legal, the KOS examines the ACLs. It allows access only if it finds an ACLE whose Subject Template matches the current subject of Process 610 which wishes to make the access and whose set of Access Privileges includes the kind of access desired by Process 610. For example, a Procedure Object 608 may have an ACL with two entries: one whose Subject Template allows any subject access, and whose set of Access Privileges allows only Execute Access, and another whose Subject Template allows only a single subject access and whose set of Access Privileges allows Read, Write, and Execute Access. Such an ACL allows any user of CS 10110 to execute the Procedures 602 in Procedure Object 608, but only a specified Process 610 belonging to a specified user and executing a specified group of Procedures 602 may examine or modify the Procedures 602 in the Procedure Object 608.

There are two kinds of ACLs. All objects have Primitive Access Control Lists (PACLs); ETOs may in addition have Extended Access Control Lists (EACLs). The subject portion of the ACLE is the same in all ACLs; the two kinds of list differ in the kinds of access they control. The access controlled by the PACL is defined by KOS and is checked by KOS on every attempt to gain such access; the access controlled by the EACL is defined by the user and is checked only when the user requests KOS to do so.

e. Implementation of Objects

1. Introduction (Fig. 407, 408)

The user of a CS 10110 need only concern himself with objects as they have just been described. In order for a Process 610 to reference an object, the object's LAU 40405 must be accessible from CS 10110 upon which Process 610 is running, Process 610 must know the object's UID 40401, and Process 610's current subject must have the right to access the object in the desired manner. Process 610 need know neither how the object's Contents 40406 and Attributes 40404 are stored on CS 10110's physical devices nor the methods CS 10110 uses to make the object's Contents 40406 and Attributes 40404 available to Process 610.

The KOS, on the other hand, must implement objects on the physical devices that make up CS 10110. In so doing, it must take into account two sets of physical limitations:

— In logical terms, all CSs 10110 have a single logical memory, but the physical implementation of memory in the system is hierarchical: a given CS 10110 has rapid access to a relatively small MEM 10112, much slower access to a relatively large amount of slow Secondary Storage 10124, and very slow access to LAUs 40405 on other accessible CSs 10110.

— UIDs 40401, and even more, subjects, are too large to be handled efficiently on JP 10114's internal data paths and in JP 10114's registers.

The means by which the KOS overcomes these physical limitations will vary from embodiment to embodiment. Here, there are presented first an overview and then a detailed discussion of the means used in the present embodiment.

The physical limitations of the memory are overcome by means of a Virtual Memory system. The Virtual Memory System creates a one-level logical memory by automatically bringing copies of those portions of objects required by executing Processes 610 into MEM 10112 and automatically copying altered portions of objects from MEM 10112 back to Secondary Storage 10124. Objects thus reside primarily in Secondary Storage 10124, but copies of portions of them are made available in MEM 10112 when a Process 610 makes a reference to them. Besides bringing portions of objects into MEM 10112, when required, the Virtual Memory System keeps track of where in MEM 10112 the portions are located, and when a Process 610 references a portion of an object that is in MEM 10112, the Virtual Memory System translates the reference into a physical location in MEM 10112.

JP 10114's need for smaller object identifiers and subject identifiers is satisfied by the use of internal identifiers called Active Object Numbers (AONs) and Active Subject Numbers (ASNs) inside JP 10114. Each time a UID 40401 is moved from MEM 10112 into JP 10114's registers, it is translated into an AON, and the reverse translation takes place each time an AON is moved from a JP 10114's registers to MEM 10112. Similarly, the current subjects of Processes 610 which are bound to Virtual Processors 612 are translated from four UIDs 40401 into small integer ASNs, and when Virtual Processor 612 is bound to JP 10114, the ASN for the subject belonging to Virtual Processor 612's process 610 is placed in a JP 10114 register. The translations from UID 40401 to AON and vice-versa, and from subject to ASN are performed by KOS.

When KOS translates UIDs 40401 to AONs and vice-versa, it uses AOT 10712. An AOT 10712 Entry (AOTE) for an object contains the object's UID 40401, and the AOTE's index in AOT 10712 is that object's AON. Thus, given an object's AON, KOS can use AOT 10712 to determine the object's UID 40401, and given an obect's UID 40401, KOS can use AOT 10712 to determine the object's AON. If the object has not been referenced recently, there may be no AOTE for the object, and thus no AON for the object's UID 40401. Objects that have no AONs are called inactive objects. If an attempt to convert a UID 40401 to an AON reveals that the object is inactive, an Inactive Object Fault results and KOS must activate the object, that is, it must assign the object an AON and make an AOTE for it.

KOS uses AST 10914 to translate subjects into ASN's. When a Process 610's subject changes, AST 10914 provides Process 610 with the new subject's ASN. A subject may presently have no ASN associated with it. Such subjects are termed inactive subjects. If a subject is inactive, an attempt to translate the subject to an ASN causes KOS to activate the subject, that is, to assign the subject an ASN and make an entry for the subject in AST 10914.

In order to achieve efficient execution of programs by Processes 610, KOS accelerates information that is frequently used by executing processes 610. There are two stages of acceleration:

— Tables that contain the information are wired into MEM 10112, that is, the Virtual Memory System never uses MEM 10112 space reserved for the tables for other purposes.

— Special hardware devices in JP 10114 contain portions of the information in the tables.

MHT 10716, AOT 10712, and AST 10914 are examples of the first stage of acceleration. As previously mentioned, these tables are always present in MEM 10112. Address Translation Unit (ATU) 10228 is an example of the second stage. As previously explained, ATU 10228 is a hardware cache that contains copies of the most recently used MHT 10716 entries. Like MHT 10716, it translates AON offset addresses into the MEM 10112 locations that contain copies of the data that the UID-offset address corresponding to the AON-offset address refers to ATU 10228 is maintained by KOS Logical Address Translation (LAT) microcode.

Figure 407 shows the relationship between ATU 10228, MEM 10112, MHT 10716, and KOS LAT microcode 40704. When JP 10114 makes a memory reference, it passes AON-offset Address 40705 to ATU 10228. If ATU 10228 contains a copy of MHT 10716's entry for Address 40705, it immediately produces the corresponding MEM 10112 Address 40706 and transmits the address to MEM 10112. If there is no copy,

ATU 10228 produces an ATU Miss Event Signal which invokes LAT microcode 40704 in JP 10114. LAT microcode 40704 obtains the MHT entry that corresponds to the AON-offset address from MHT 10716, places the entry in ATU 10228, and returns. JP 10114 then repeats the reference. This time, there is an entry for the reference, and ATU 10228 translates the AON address into the address of the copy of the data contained in MEM 10112.

The relationship between KOS table, hardware cache, and microcode just described is typical for the present embodiment of CS 10110. The table (in this case, MHT 10716), is the primary source of information and is maintained by the Virtual Memory Manager Process, while the cache accelerates portions of the table and is maintained by KOS microcode that is invoked by event signals from the cache.

AOT 10712, AST 10914, and MHT 10716 share another characteristic that is typical of the present embodiment of CS 10110: the tables are constructed in such a fashion that the table entry that performs the desired translation is located by means of a hash function and a hash table. The hash function translates the large UID 40401, subject, or AON into a small integer. This integer is the index of an entry in the hash table. The contents of the hash table entry is an index into AOT 10712, AST 10914, or MHT 10716, as the case may be, and these tables are maintained in such a fashion that the entry corresponding to the index provided by the hash table is either the entry that can perform the desired translation or contains information that allows KOS to find the desired entry. The entries in the tables furthermore contain the values they translate. Consequently, KOS can hash the value, find the entry, and then check whether the entry is the one for the hashed value. If it is not, KOS can quickly go from the entry located by the hash table to the correct entry.

Figure 408 shows how hashing works in AST 10914 in the present embodiment. In the present embodiment, Subject 40801, i.e., the principal, process, and domain components of the current subject, are input into Hash Function 40802. Hash Function 40802 produces the index of an entry in ASTHT 10710. ASTHT Entry 40504 in turn contains the index of an Entry (ASTE) 40806 in AST 10914 These ASTE 40806 indexes are ASNs. ASTE 40806 contains the principal, process, and domain components of some subject and a link field pointing to ASTE 40806'. ASTE 40806' has 0 in its link field, which indicates that it is the last link in the chain of ASTES begining with ASTE 40806. If the hashing of a subject yields ASTE 40806, KOS compares the subject in ASTE 40806 with the hashed subject; if they are identical, ASTE 40806's index in AST 10914 is the subject's ASN. If they are not identical, KOS uses the link in ASTE 40806 to find ASTE 40806'. It compares the subject in ASTE 40806' with the hashed subject; if they are identical, ASTE 40806''s AST index is the subject's ASN; otherwise, ASTE 40806' is the last entry in the chain, and consequently, there is no ASTE 40806 and no ASN for the hashed subject.

In the following, we will discuss the implementation of objects in the present embodiment in detail, beginning with the implementation of objects in Secondary Storage 10124 and proceeding then to CS 10110's Active Object Management System, the Access Control System, and the Virtual Memory System.

2. Objects in Secondary Storage 10124 (Figs. 409, 410)

As described above, objects are collected into LAUs 40405. The objects belonging to a LAU 40405 are stored in Secondary Storage 10124. Each LAU 40405 contains an object whose contents are a table called the Logical Allocation Unit Directory (LAUD). As its name implies, the LAUD is a directory of the objects in LAU 40405. Each object in LAU 40405, including the object containing the LAUD, has an entry in the LAUD. Figure 409 shows the relationship between Secondary Storage 10124, LAU 40405, the LAUD, and objects. LAU 40405 resides on a number of Storage Devices 40904. LAUD Object 40902' in LAU 40405 contains LAUD 40903. Two LAUDEs 40906 are shown. One contains the attributes of LAUD Object 40902 and the location of its contents, and the other contains the attributes of LAUD Object 40902' containing LAUD 40903 and the location of its contents.

KOS uses a table called the Active LAU Table (ALAUT) to locate the LAUD belonging to LAU 40405. Figure 410 illustrates the relationship between ALAUT 41001, ALAUT Entries 41002, LAUs 40405, and LAUD Objects 40902'. Each LAU 40405 accessible to CS 10110 has an Entry (ALAUTE) 41002 in ALAUT 41001. ALAUTE 41002 for LAU 40405 includes LAU 40405's LAUID 40402 and UID 40401 of LAU 40705's LAUD Object 40902'. Hence, given an object's UID 40401, KOS can use UID 40401's LAUID 40402 to locate ALAUTE 41002 for the object's LAU 40405, and can use ALAUTE 41002 to locate LAU 40405's LAUD 40903. Once LAUD 40903 has been found, OSN portion 40402 of the object's UID 40401 provides the proper LAUDE 40906, and LAUDE 40906 contains object's attributes and the location of its contents.

LAUD 40903 and the Procedures 602 that manipulate it belong to a part of KOS termed the Inactive Object Manager. The following discussion of the Inactive Object Manager will begin with the manner in which an object's contents are represented on Secondary Storage 10124, will then discuss LAUD 40903 in detail, and conclude by discussing the operations performed by Inactive Object Manager Procedures 602.

a.a. Representation of an Object's Contents on Secondary Storage 10124

In general, the manner in which an object's contents are represented on Secondary Storage 10124 depends completely on the Secondary Storage 10124. If a LAU 40405 is made up of disks, then the object's contents will be stored in disk blocks. As long as KOS can locate the object's contents, it makes no difference whether the storage is contiguous or non-contiguous.

In the present embodiment, the objects' contents are stored in files created by the Data General

Advance Operating System (AOS) procedures executing on IOS 10116 These procedures manage files that contain objects' contents for KOS. In future CSs 10110, the representation of an object's contents on Secondary Storage 10124 will be managed by a portion of KOS.

b.b. LAUD 40903 (fig. 411, 412)

Figure 411 is a conceptual illustration of LAUD 40903. LAUD 40903 has three parts: LAUD Header 41102, Master Directory 41105, and LAUD Entries (LAUDEs) 40906. LAUD Header 41102 and Master Directory 41105 occupy fixed locations in LAUD 40903, and can therefore always be located from the UID 40401 of LAUD 40903 given in ALAUT 41001. The locations of LAUDEs 40906 are not fixed, but the entry for an individual object can be located from Master Directory 41105.

Turning first to LAUD Header 41102, LAUD Header 41102 contains LAUID 40402 belonging to LAU 40405 to which LAUD 40903 belongs and OSN 40403 of LAUD 40903. As will be explained in greater detail below, KOS can use OSN 40403 to find LAUDE 40906 for LAUD 40903.

Turning now to Master Directory 41105, Master Directory 41105 translates an object's OSN 40403 into the location of the object's LAUDE 40906. Master Directory 41105 contains one Entry 41108 for each object in LAU 40505. Each Entry has two fields: OSN Field 41106 and Offset Field 41107. OSN Field 41106 contains OSN 40403 for the object to which Entry 41108 belongs; Offset Field 41107 contains the offset of the object's LAUDE 40906 in LAUD 40903. KOS orders Entries 41108 by increasing OSN 40403, and can therefore use binary search means to find Entry 41108 containing a given OSN 40403. Once Entry 41108 has been located, Entry 41108's Offset Field 41107, combined with LAUD 40903's OSN 40403, yields the UID offset address of the object's LAUDE 40906.

Once KOS knows the location of LAUDE 40906 it can determine an object's Attributes 40404 and the location of its Contents 40406. Figure 411 gives only an overview of LAUDE 40906's general structure. LAUDE 40906 has three components: a group of fields of fixed size 41109 that are present in every LAUDE 40906, and two variable sized components, one, 41139, containing entries belonging to the object's PACL, and another, 41141, containing the object's EACL.

As the preceding descriptions of the LAUD's components imply, the number of LAUDEs 40906 and Master Directory Entries 41108 varies with the number of objects in LAU 40405. Furthermore, the amount of space required for an object's EACL and PACL varies from object to object. KOS deals with this problem by including Free Space 41123 in each LAUD 40903. When an object is created, or when an object's ACLs are expanded, the Inactive Object Manager expands LAUD 40903 only if there is no available Free Space 41123; if there is Free Space 41123, the Inactive Object Manager takes the necessary space from Free Space 41123; when an object is deleted or an object's ACLs shortened, the Inactive Object Manager returns the unneeded space to Free Space 41123.

Figure 412 is a detailed representation of a single LAUDE 40906. Figure 412 presents those fields of LAUDE 40906 which are common to all embodiments of CS 10110; fields which may vary from embodiment to embodiment are ignored. Starting at the top of Figure 412, Structure Version Field 41209 contains information by which KOS can determine which version of LAUDE 40906 it is dealing with. Size Field 41211 contains the Size Attribute of the object to which LAUDE 40906 belongs. The Size Attribute specifies the number of bits currently contained in the object. Lock Field 41213 is a KOS lock. As will be explained in detail in the discussion of Processes 610, Lock Field 41213 allows only one Process 610 to read or write LAUDE 40906 at a time, and therefore keeps one Process 610 from altering LAUDE 40906 while another Process 610 is reading LAUDE 40906. File Identifier 41215 contains a system identifier for the file which contains the Contents 40406 of the object to which LAUDE 40906 belongs. The form of File Identifier 41215 may vary from embodiment to embodiment; in the present embodiment, it is an AOS system file identifier. UID Field 41217 contains UID 40401 belonging to LAUDE 40906's object. Primitive Type Field 41219 contains a value which specifies the object's Primitive Type. The object may be a data object, a Procedure Object 608, an ETM, or an S-interpreter object. AON Field 41221 contains a valid value only when LAUDE 40906's object is active, i.e., has an entry in AOT 10712. AON Field 41221 then contains the object's AON. If the object is an ETO, Extended Type Attribute Field 41223 contains the UID 40401 of the ETO's ETM. Otherwise, it contains a Null UID 40401. Similarly, if the object is a Procedure Object 608 or an ETM, Domain of Execution Attribute Field 41225 contains the object's Domain of Execution Attribute.

The remaining parts of LAUDE 40906 belong to the Access Control System and will be explained in detail in that discussion. Attribute Version Number Field 41227 contains a value indicating which version of ACLEs this LAUDE 40906 contains, PACL Size Field 41229 and EACL Size Field 41231 contain the sizes of the respective ACLs, PACL Offset Field 41233 and EACL Offset Field 41235 contain the offsets in LAUD 40903 of additional PACLEs 41139 and EACLEs 41141, and fixed PACLEs 41237 contains the portion of the PACL which is always included in LAUDE 40906.

3. Active Objects (fig. 413)

An active object is an object whose UID 40401 has an AON associated with it. In the present embodiment, each CS 10110 has a set of AONs' KOS associates these AONs with UIDs 40401 in such fashion that at any given moment, an AON in a CS 10110 represents a single UID 40401. Inside FU 10120, AONs are used to represent UIDs CS 10110. In the present embodiment, the AON is represented by 14 bits. A 112-bit UID-offset address (80 bits for UID 40401 and 32 for the offset) is thus represented inside FU 10120

by a 46-bit AON-offset address (14 bits for the AON and 32 bits for the offset).

A CS 10110 has far fewer AONs than there are UIDs 40401. KOS multiplexes a CS 10110's AONs among those objects that are being referenced by CS 10110 and therefore require AONs as well as UIDs 40401. While a given AON represents only a single UID 40401 at any given time, at different times, a UID 40401 may have different AONs associated with it.

Figure 413 provides a conceptual representation of the relationship between AONs and UIDs 40401. Each CS 10110 has potential access to 2**80 UIDs 40401. Some of these UIDs, however, represent entities other than objects, and others are never associated with any entity. Each CS 10110 also has a set of AONs 41303 available to it. In the present embodiment, this set may have up to 2**14 values. Since the AONS are only used internally, each CS 10110 may have the same set of AONs 41303. Any AON 41304 in set of AONs 41303 may be associated with a single UID 40401 in set of object UIDs 41301. At different times, an AON 41304 may be associated with different UIDs 40401.

As mentioned above, KOS associates AONs 41304 with UIDs 40401. It does so by means of AOT 10712. Each AOT entry (AOTE) 41306 in AOT 10712 associates a UID 40401 with an AON 41304. AON 41304 is the index of AOTE 41306 which contains UID 40401. Until AOTE 41306 is changed, the AON 41304 which is the index of AOTE 41306 containing UID 40401 represents UID 40401. AOT 10712 also allows UIDs 40401 to be translated into AONs 41303 and vice-versa. Figure 413 illustrates the process for UID-offset Address 41308 and AON-offset Address 41309. AOTE 41306 associates AON 41304 in AON-offset Address 41309 with UID 40401 in UID-offset Address 41308, and Addresses 41308 and 41309 have the same Offset 41307. Consequently, AON-offset Address 41309 represents UID-offset Address 41308 inside JP 10114. Since both addresses use the same Offset, Address 41309 can be translated into address 41308 by translating Address 41309's AON 41304 into Address 41308's UID 40401, and Address 41308 can be translated into Address 41309 by the reverse process. In both cases, the translation is performed by finding the proper AOTE 41306.

The process by which an object becomes active is called object activation. A UID-offset Address 41308 cannot be translated into an AON-offset Address 41309 unless the object to which UID 40401 of UID-offset Address 41308 belongs is active. If a Process 610 attempts to perform such a translation using a UID 40401 belonging to an inactive object, an Inactive Object Fault occurs. KOS handles the fault by removing Process 610 that attempted the translation from JP 10114 until a special KOS Process called the Object Manager Process has activated the object. After the object has been activated, Process 610 may return to JP 10114 and complete the UID 40401 to AON 41304 translation.

The portion of KOS that manages active objects is called the Active Object Manager (AOM). Parts of the AOM are Procedures 602, and parts of it are microcode routines. The high-level language components of the AOM may be invoked only by KOS processes 610. KOS Active Object Manager Process 610 performs most of the functions involved in active object management.

a.a. UID 40401 to AON 41304 Translation

Generally speaking, in CS 10110, addresses stored in MEM 10112 and Secondary Memory 10124 are stored as UID offset addresses. The only form of address that FU 10120 can translate into a location in MEM 10112 is the AON-offset form. Consequently, each time an address is loaded from MEM 10112 into a FU 10120 register, the address must be translated from a UID-offset address to an AON-offset address. The reverse translation must be performed each time an address is moved from a FU 10120 register back into memory.

Such translations may occur at any time. For example, a running Virtual Processor 612 performs such a translation when the Process 610 being executed by Virtual Processor 612 carries out an indirect memory reference. An indirect memory reference is a reference which first fetches a pointer, that is, a data item whose value is the address of another data item, and then uses the address contained in the pointer to fetch the data itself. In CS 10110, pointers represent UID-offset addresses. Virtual Processor 612 performs the indirect memory reference by fetching the pointer from MEM 10112, placing it in FU 10120 registers, translating UID 40401 represented by the pointer into AON 41304 associated with it, and using the resulting AON-offset address to access the data at the location specified by the address.

Most such translations, however, occur when Virtual Processor 612 state is saved or restored. For instance, when one Process 610's Virtual Processor 612 is removed from JP 10114 and another Process 610's Virtual Processor 612 is bound to JP 10114, the state of Virtual Processor 612 being removed from JP 10114 is stored in memory, and the state of Virtual Processor 612 being bound to JP 10114 is moved into JP 10114's registers. Because only UID-offset addresses may be stored in memory, all of the AON-offset addresses in the state of Virtual Processor 612 which is being removed from JP 10114 must be translated into UID-offset addresses. Similarly, all of the UID-offset addresses in the state of Virtual Processor 612 being bound to JP 10114 must be translated into AON-offset addresses before they can be loaded into FU 10120 registers.

C. The Access Control System

As mentioned in the introduction to objects, each time a process 610 accesses data or SINs in an object, the KOS Access Control System checks whether Process 610's current subject has the right to perform the kind of access that Process 610 is attempting. If Process 610's current subject does not have the proper access, the Access Control System aborts the memory operation which Process 610 was attempting to

carry out. The following discussion presents details of the implementation of the Access Control System, beginning with subjects, then proceeding to subject templates, and finally to the means used by KOS to accelerate access checking.

a. Subjects

A Process 610's subject is part of process 610's state and is contained along with other state belonging to Process 610 in an object called a Process Object. Process Objects are dealt with at length in the detailed discussion of Processes 610 which follows the discussion of objects. While a subject has, as mentioned above, four components, the principal component, the process component, the domain component, and the tag component, the Access Control System in the present embodiment of CS 10110 assigns values to only the first three components and ignores the tag component when checking access.

In the present embodiment, the UIDs 40401 which make up the components of a Process 610's subject are the UIDs 40401 of objects containing information about the entities represented by the UIDs 40401. The principal component's UID 40401 represents an object called the Principal Object. The Principal Object contains information about the user for whom Process 610 was created. For example, the information might concern what access rights the user had to the resources of CS 10110, or it might contain records of his use of CS 10110. The process component's UID 40401 represents the Process Object, while the domain component's UID 40401 represents an object called the Domain Object. The Domain Object contains information which must be accessible to any Process 610 whose subject has the Domain Object's UID 40401 as its domain component. Other embodiments of CS 10110 will use the tag component of the subject. In these embodiments, the tag component's UID 40401 is the UID 40401 of a Tag Object containing at least such information as a list of the subjects which make up the group of subjects represented by the tag component's UID.

b. Domains

As stated above, the subject's domain component is the domain of execution attribute belonging to the Procedure Object 608 or ETM whose code is being executed when the access request is made. The domain component of the subject thus gives Process 610 to which the subject belongs potential access to the group of objects whose ACLs have ACLEs with subject templates containing domain components that match the DOE attribute. This group of objects is the domain defined by the Procedure Object 608 or ETM's DOE attribute. When a Process 610 executes a Procedure 602 from a Procedure Object 608 or ETM with a given DOE attribute, Process 610 is said to be executing in the domain defined by that DOE attribute. As may be inferred from the above, different Procedure Objects 608 or ETMs may have the same DOE attribute, and objects may have ACLEs which make them members of many different domains.

In establishing a relationship between a group of Procedure Objects 608 and another group of objects, a domain allows a programmer using CS 10110 to ensure that a given object is read, executed, or modified only by a certain set of Procedures 602. Domains may thus be used to construct protected subsystems in CS 10110. One example of such a protected subsystem is KOS itself: the objects in CS 10110 which contain KOS tables all have ACLs whose domain template components match only the DOE which represents the KOS domain. The only Procedure Objects 608 and ETMs which have this DOE are those which contain KOS Procedures 602, and consequently, only KOS Procedures 602 may manipulate KOS tables.

Since an object may belong to more than one domain, a programmer may use domains to establish hierarchies of access. For example, if some of the objects in a first domain belong both to the first domain and a second domain, and the second domain's objects all also belong to the first domain, then Procedures 602 contained in Procedure Objects 608 whose DOEs define the first domain may access any object in the first domain, including those which also belong to the second domain, while those from Procedure Objects 608 whose DOEs define the second domain may access only those objects in the second domain.

c. Access Control Lists

As previously mentioned, the Access Control System compares the subject belonging to Process 610 making an access to an object and the kind of access Process 610 desires to make with the object's ACLs to determine whether the access is legal. The following discussion of the ACLs will first deal with Subject Templates, since they are common to all ACLs, and then with PACLs and EACLs.

1. Subject Templates (Fig. 416)

Figure 416 shows Subject Templates, PACL Entries (PACLEs), and EACL Entries (EACLEs). Turning first to the Subject Templates, Subject Template 41601 consists of four components, Principal Template 41606, Process Template 41607, Domain Template 41609, and Tag Template 41611. Each template has two fields, Flavor Field 41603, and UID Field 41605. Flavor Field 41603 indicates the way in which the template to which it belongs is to match the corresponding component of the subject for Process 610 attempting the access. Flavor Field 41603 may have one of three values: match any, match one, match group. If Flavor Field 41603 has the value match any, any subject component UID 40401 matches the template, and the Access Control System does not examine UID Field 41605. If Flavor Field 41603 has the value match one, then the corresponding subject component must have the same UID 40401 as the one contained in UID Field 41605. If Flavor Field 41603 has the value match group, finally, then UID Field 41605 contains a UID 40401 of an

object containing information about the group of subject components which the given subject component may match.

2. Primitive Access Control Lists (PACLs)

PACLs are made up of PACLEs 41613 as illustrated in Figure 416. Each PACLE 41613 has two parts: a subject template 41601 and an Access Mode Bits Field 41615. The values in Access Mode Bits Field 41615 define 11 kinds of access. The eleven kinds fall into two groups: Primitive Data Access and Primitive Non-data Access. Primitive Data Access controls what the subject may do with the object's Contents 40406; Primitive Non-data Access controls what the subject may do with the object's Attributes 40404.

There are three kinds of Primitive Data Access: Read Access, Write Access, and Execute Access. If a subject has Read Access, it can examine the data contained in the object; if the subject has Write Access, it can alter the data contained in the object; if it has Execute Access, it can treat the data in the object as a Procedure 602 and attempt to execute it. A subject may have none of these kinds of access, or any combination of the kinds. On every reference to an object, the KOS checks whether the subject performing the reference has the required Primitive Data Access.

Primitive Non-data Access to an object is required only to set or read an object's Attributes 40404, and is checked only when these operations are performed. The kinds of Non-data Access correspond to the kinds of Attributes 40404:

Attributes	Kind of Access
Object Attributes	get object attributes set object attributes
Primitive Control	get primitive control attributes
Attributes	set primitive control attributes
Extended Control Attributes	get extended control attributes set extended control attributes
ETM Access	use as ETM create ETO

The access rights for object attributes allow a subject to get and set the object attributes described previously. The access rights for primitive and extended control attributes allow a subject to get and set an object's PACL and EACL respectively.

An object may have any number of PACLEs 41613 in its PACL. The first five PACLEs 41613 in an object's PACL are contained in fixed PACLE Field 41237 of LAUDE 40906 for the object; the remainder are stored in LAUD 40903 at the location specified in PACL Offset Field 41233 of LAUDE 40906.

3. APAM 10918 and Protection Cache 10234 (Fig. 421)

Primitive non-data access rights are checked only when users invoke KOS routines that require such access rights, and extended access rights are checked only when users request such checks. Primitive data access rights, on the other hand, are checked every time a Virtual Processor 612 makes a memory reference while executing a Process 610. The KOS implementation of primitive data access right checking therefore emphasizes speed and efficiency. There are two parts to the implementation: APAM 10918 in MEM 10112, and Protection Cache 10234 in JP 10114. APAM 10918 is in a location in MEM 10112 known to KOS microcode. APAM 10918 contains primitive data access information copied from PACLEs 41613 which belong to active objects and whose Subject Template 41601 matches an active subject. Protection Cache 10234, in turn, contain copies of the information in APAM 10918 for the active subject of Process 610 whose Virtual Processor 612 is currently bound to JP 10114 and active objects referenced by Process 610. A primitive data access check in CS 10110 begins with Protection Cache 10234, and if the information is not contained in Protection Cache 10234, proceeds to APAM 10918, and if it is not there, finally, to the object's PACL. The discussion which follows begins with APAM 10918.

Figure 421 shows APAM 10918. APAM 10918 is organized as a two-dimensional array. The array's row indexes are AONs 41304, and its column indexes are ASNs. There is a row for each AON 41304 in CS 10110, and a column for each ASN. In Figure 421, only a single row and column are shown. Any primitive data access information in APAM 10918 for the object represented by AON 41304 j is contained in Row 42104, while Column 42105 contains any primitive data access information in APAM 10918 for the subject

represented by ASN k. APAM Entry (APAME) 42106 is at the intersection of Row 42104 and Column 42105, and thus contains the primitive data access information from that PACLE 41613 belonging to the object represented by AON 41304 j whose Subject Template 41601 matches the subject represented by ASN k.

An expanded view of APAME 42106 is presented beneath the epresentation of APAM 10918. APAME 42106 contains four 1-bit fields. The bits represent the kinds of primitive data access that the subject represented by APAME 42106's column index has to the object represented by APAME 42106's row index.

- Field 42107 is the Valid Bit. If the Valid Bit is set, APAME 42106 contains whatever primitive data access information is available for the subject represented by the column and the object represented by the row. The remaining fields in APAME 42106 are meaningful only if Valid Bit 42107 is set.
- Field 42109 is the Execute Bit. If it is set, APAME 42106's subject has Execute Access to APAME 42106's object.
- Field 42111 is the Read Bit. If it is set, APAME 42106's subject has Read Access to APAME 42106's object.
- Field 42113 is the Write Bit. If it is set, APAME 42106's subject has Write Access to APAME 42106's object.

Any combination of bits in Fields 42109 through 42113 may be set. If all of these fields are set to 0, APAME 42106 indicates that the subject it represents has no access to the object it represents.

KOS sets APAME 42106 for an ASN and an AON 41304 the first time the subject represented by the ASN references the object represented by AON 41304. Until APAME 42106 is set, Valid Bit 42107 is set to 0. When APAME 42106 is set, Valid Bit 42107 is set to 1 and Fields 42109 through 42113 are set according to the primitive data access information in the object's PACLE 41613 whose Subject Template 41601 matches the subject. When an object is deactivated, Valid Bits 42107 in all APAMEs 42106 in the row belonging to the object's AON 41304 are set to 0; similarly, when a subject is deactivated, Valid Bits 42107 in all APAMEs 42106 in the column belonging to the subject's ASN are set to 0.

4. Protection Cache 10234 and Protection Checking (Fig. 422)

The final stage in the acceleration of protection information is Protection Cache 10234 in JP 10114. The details of the way in which Protection Cache 10234 functions are presented in the discussion of the hardware; here, there are discussed the manner in which Protection Cache 10234 performs access checks, the relationship between protection Cache 10234, APAM 10918, and AOT 10712, and the manner in which KOS protection cache microcode maintains Protection Cache 10234.

Figure 422 is a block diagram of Protection Cache 10234, AOTE 10712, APAM 10918, and KOS Microcode 42207 which maintains Protection Cache 10234. Each time JP 10114 makes a memory reference using a Logical Descriptor 27116, it simultaneously presents Logical Descriptor 27116 and a Signal 42208 indicating the kind of memory operation to Protection Cache 10234 and ATU 10228. Entries 42215 in Protection Cache 10234 contain primitive data access and length information for objects previously referenced by the current subject of Process 610 whose Virtual Processor 612 is currently bound to JP 10114. On every memory reference, Protection Cache 10234 emits a Valid/invalid Signal 42205 to MEM 10112. If Protection Cache 10234 contains no Entry 42215 for AON 41304 contained in Logical Descriptor 27116's AON field 27111, if Entry 42215 indicates that the subject does not have the type of access required by process 610, or if the sum of Logical Descriptor 27116's OFF field 27113 and LEN field 27115 exceed the object's current size, Protection Cache 10234 emits an Invalid Signal 42205. This signal causes MEM 10112 to abort the memory reference. Otherwise, Protection Cache 10234 emits a Valid Signal 42205 and MEM 10112 executes the memory reference.

When Protection Cache 10234 emits an Invalid Signal 42205, it latches Logical Descriptor 27116 used to make the reference into Descriptor Trap 20256, the memory command into Command Trap 27018, and if it was a write operation, the data into Data Trap 20258, and at the same time emits one of two Event Signals to KOS microcode. Illegal Access Event Signal 42208 occurs when Process 610 making the reference does not have the proper access rights or the data referenced extends beyond the end of the object. Illegal Access Event Signal 42208 invokes KOS microcode 42215 which performs a Microcode to Software Call 42217 (described in the discussion of Calls) to KOS Access Control System Procedures 602 and passes the contents of Descriptor Trap 20256, Command Trap 27018, the ASN of Process 610 (contained in a register MGR's 10360), and if necessary, Data Trap 20258 to these Procedures 602. These procedures 602 inform EOS of the protection violation, and EOS can then remedy it.

Cache Miss Event Signal 42206 occurs when there is no Entry 42215 for AON 41304 in protection Cache 10234. Cache Miss Event Signal 42206 invokes KOS Protection Cache Miss Microcode 42207, which constructs missing Protection Cache Entry 42215 from information obtained from AOT 10712 and APAM 10918. If APAM 10918 contains no entry for the current subject's ASN and the AON of the object being referenced, protection Cache Miss Microcode 42207 performs a Microcode-to-software Call to KOS Access Control System Procedures 602 which go to LAUDE 40906 for the object and copy the required primitive data access information from the PACLE 41613 belonging to the object whose Subject Template 41601 matches the subject attempting the reference into APAM 10918. The KOS Access Control System Procedures 602 then return to Cache Miss Microcode 42207, which itself returns. Since Cache Miss Microcode 41107 was invoked by an Event Signal, the return causes JP 10114 to reexecute the memory reference which caused the protection cache miss. If protection Cache 10234 was loaded as a result of the

last protection cache miss, the miss does not recur; if Protection Cache 10234 was not loaded because the required information was not in APAM 10918, the miss recurs, but since the information was placed in APAM 10918 as a result of the previous miss, Cache Miss Microcode 42207 can now construct an Entry 42215 in Protection Cache 10234. When Cache Miss Microcode 42207 returns, the memory reference is again attempted, but this time Protection Cache 10234 contains the information and the miss does not recur.

Cache Miss Microcode 42207 creates a new Protection Cache Entry 42215 and loads it into Protection Cache 10234 as follows: Using AON 41304 from Logical Descriptor 27116 latched into Descriptor Trap 20256 when the memory reference which caused the miss was executed and the current subject's ASN, contained in GR's 10360, Cache Miss Microcode locates APAME 42106 for the subject represented by the ASN and the object represented by AON 41304 and copies the contents of APAME 42106 into a JP 10114 register which may serve as a source for JPD Bus 10142. It also uses AON 41304 to locate AOTE 41306 for the object and copies the contents of Size Field 41519 into another JP 10114 register which is a source for JPD Bus 10142. It then uses three special microcommands, executed in successive microinstructions, to load Protection Cache Entry 42215. The first microcommand loads Protection Cache Entry 42215's TS 24010 with AON 41304 of Logical Descriptor 27116 latched into Descriptor Trap 20256; the second loads the object's size into Protection Cache 10234's EXTENT field, and the third loads the contents of APAME 42106 in the same fashion.

Another microcommand invalidates all Entries 42215 in Protection Cache 10234. This operation, called flushing, is performed when an object is deactivated or when the current subject changes. The current subject changes whenever a Virtual Processor 612 is unbound from JP 10114, and whenever a Process 610 performs a call to or a return from a Procedure 602 executing in a domain different from that in which the calling Procedure 602 or the Procedure 602 being returned to executes in. In the cases of the Call and the unbinding of Virtual Processor 612, the cache flush is performed by KOS Call and dispatching microcode; in the case of object deactivation, it is performed by a KOS procedure using a special KOS SIN which invokes Cache Flush Microcode.

D. Processes

1. Synchronization of Processes 610 and Virtual Processors 612

Since Processes 610 and the Virtual Processors 612 to which they are bound may execute concurrently on CS 10110, KOS must provide means for synchronizing Processes 610 which depend on each other. For example, if process 610 A cannot proceed until Process 610 B has performed some operation, there must be a mechanism for suspending A's execution until B is finished. Generally speaking, four kinds of synchronization are necessary:

- One Process 610 must be able to halt and wait for another Process 610 to finish a task before it proceeds.
- One Process 610 must be able to send another Process 610 a message and wait for a reply before it proceeds.
- When processes 610 share a data base, one Process 610 must be able to exclude other Processes 610 from the data base until the first Process 610 is finised using the data base.
- One Process 610 must be able to interrupt another Process 610, i.e., asynchronously cause the second Process 610 to perform some action.

KOS has internal mechanisms for each kind of synchronization, and in addition supplies synchronization mechanisms to EOS. KOS uses the internal mechanisms to synchronize Virtual Processors 612 and KOS Processes 610, while EOS uses the mechanisms supplied by KOS to synchronize all other Processes 610. The internal mechanisms are the following:

- Event counters, Await Entries, and Await Tables. As will be explained in detail below, Event Counters and Await Entries allow one Process 610 to halt and wait for another Process 610 to complete an operation. Event counters and Await Entries are also used to implement process interrupts. Await Entries are organized into Await Tables.
- Message Queues. Message Queues allow one Process 610 to send a message to another and wait for a reply. Message Queues are implemented with Event Counters and queue data structures.
- Locks. Locks allow one Process 610 to exclude other Processes 610 from a data base or a segment of code. Locks are implemented with Event Counters and devices called Sequencers.

KOS makes Event Counters, Await Entries, and Message Queues available to EOS. It does not provide Locks, but it does provide Sequencers, so that EOS can construct its own Locks. The following discussion will define and explain the logical properties of Event Counters, Await Entries, Message Queues, Sequencers, and Locks. Their implementation in the present embodiment will be described along with the implementation of Processes 610 and Virtual Processors 612.

a. Event Counters 44801, Await Entries 44804, and Await Tables (Fig. 448, 449)

Event Counters, Await Entries, and Await Tables are the fundamental components of the KOS Synchronization System. Figure 448 illustrates Event Counters and Await Entries in the present embodiment. Figure 449 gives a simplified representation of Process Event Table 44705, the present embodiment's Await Tables. Turning first to Figure 448, Event Counter 44801 is an area of memory which

contains a value that may only be increased. In one of the present embodiment, Event Counters 44801 for KOS systems which may not page fault are always present in MEM 10112; other Event Counters 44801 are stored in Secondary Storage 10124 unless a Process 610 has referenced them and thereby caused the VMM System to load them into MEM 10112. The value contained in an Event Counter 44801 is termed an Event Counter Value 44802. In the present embodiment, EventCounter 44801 contains 64 bits of data, of which 60 make up Event Counter Value 44802. Event Counter 44801 may be referred to either as a variable or by means of a 128-bit UID pointer which contains Event Counter 44801's location. The UID pointer is termed an Event Counter Name 44803.

Await Entry 44804 is a component of entries in Await Tables. In the present embodiment, there are two Await Tables: Process Event Table 44705 and Virtual Processor Await Table (VPAT) 45401. VPAT 45401 is always present in MEM 10112. As already mentioned, Figure 449 illustrates PET 44705. Both PET 44705 and UPAT 45401 will be described in detail later. Each Await Entry 44804 contains an Event Counter Name 44803, an Event Counter Value 44802, and a Back Link 44805 which identifies a Process 610 or a Virtual Processor 612. Await Entry 44804 thus establishes a relationship between an Event Counter 44801, an Event Counter Value 44802, and a Process 610 or Virtual processor 612.

Turning now to Figure 449, in the present embodiment, all Await Entries 44804 for user Processes 610 are contained in PET 44705. PET 44705 also contains other information. Figure 449 presents only those parts of PET 44705 which illustrate Await Entries 44804. PET 44705 is structured to allow rapid location of Await Entries 44804 belonging to a specific Event Counter 44801. PET entries (PETEs) 44909 contain links which allow them to be combined into lists in PETE 44705. There are four kinds of lists in PET 44705:

- Event counter lists: these lists link all PETEs 44909 for Event Counters 44801 whose Event Counter Names 44803 hash to a single value.
- Await lists: These lists link all PETEs 44909 for Event Counters 44801 which a given Process 610 is awaiting.
- Interrupt lists: These lists link all PETEs 44909 for Event Counters 44801 which will cause an interrupt to occur for a given Process 610.
- The Free list: PETEs 44909 which are not being used in one of the above lists are on a free list.

Each PETE 44909 which is on an await list or an interrupt List is also on an event counter list.

Turning first to the event counter lists, all PETEs 44909 on a given event counter list contain Event Counter Names 44803 which hash to a single value. The value is produced by Hash Function 44901, and then used as an index in PET Hash Table (PETHT) 44903. That entry in PETHT 44903 contains the index in PET 44705 of that PETE 44909 which is the head of the event counter list. PETE List 44904 represents one such event counter list. Thus, given an Event Counter Name 44803, KOS can quickly find all Await Entries 44804 belonging to Event Counter 44801.

In the present embodiment, the implementation of Event Counters 44801 and tables with Await Entries 44804 involves both Processes 610 and Virtual Processors 612 to which Processes 610 are bound. As will be explained later, a large number of Event Counters 44801 and Await Entries 44804 belonging to Processes 610 are multiplexed onto a small number of Event Counters 44801 and Await Entries 44804 belonging to the Processes' Virtual Processors 612. Await entries 44804 for Event Counters 44801 belonging to Virtual Processors 612 are contained in VPAT 45401.

b. Synchronization with Event Counters 44801 and Await Entries 44804

The simplest form of Process 610 synchronization provided by KOS uses only Event Counters 44801 and Await Entries 44804. Coordination takes place like this: A Process 610 A requests KOS to perform an Await Operation, i.e., to establish one or more Await Entries 44804 and to suspend Process 610 A until one of the Await Entries is satisfied. In requesting the Await Operation, Process 610 A defines what Event Counters 44801 it is awaiting and what Event Counter Values 44802 these Event Counters 44801 must have for their Await Entries 44804 to be satisfied. After KOS establishes Await Entries 44804, it suspends Process 610 A. While process 610 A is suspended, other Processes 610 request KOS to perform Advance Operations on the Event Counters 44801 specified in Process 610 A's Await Entries 44804. Each time a Process 610 requests an Advance Operation on an Event Counter 44801, KOS increments Event Counter 44801 and checks Event Counter 44801's Await Entries 44804. Eventually, one Event Counter 44801 satisfies one of Process 610 A's Await Entries 44804, i.e., reaches a value equal to or greater than the Event Counter Value 44802 specified in its Await Entry 44804 for process 610 A. At this point, KOS allows process 610 A to resume execution. As process 610 A resumes execution, it deletes all of its Await Entries 44804.

E. Virtual Processors 612 (fig. 453)

As previously stated, a Virtual processor 612 may be logically defined as the means by which a Process 610 gains access to JP 10114. In physical terms, a Virtual Processor is an area of MEM 10112 which contains the information that the KOS microcode which binds Virtual Processors 612 to JP 10114 and unbinds them from JP 10114 requires to perform the binding and unbinding operations. Figure 453 shows a Virtual Processor 612. The area of MEM 10112 belonging to a Virtual Processor 612 is Virtual processor 612's Virtual Processor State Block (VPSB) 614. Each Virtual Processor 612 in a CS 10110 has a VPSB 614. Together, the VPSBs 614 make up VPSB Array 45301. Within the Virtual Processor management system, each Virtual Processor 612 is known by its VP Number 45304, which is the index of the Virtual Processor

612's VPSB 614 in VPSB Array 45301. Virtual Processors 612 are managed by means of lists contained in Micro VP Lists (MVPL) 45309. Each Virtual processor 612 has an Entry (MVPLE) 45321 in MVPL 45309, and as Virtual Processor 612 changes state, virtual processor management microcode moves it from one list to another in MVPL 45309.

VPSB 614 contains two kinds of information:

information from Process Object 901 belonging to Process 610 which is bound to VPSB 614's Virtual Processor 612, and information used by the Virtual Processor Management System to manage Virtual Processor 612. The most important information from Process Object 901 is the following:

— Process 610's principal and process UIDs 40401.
— AONs 41304 for Process 610's Stack Objects 44703. (VPSB 614 uses AONs 41304 because KOS guarantees that AONs 41304 belonging to Stack Objects 44703 will not change as long as a Process 610 is bound to a Virtual Processor 612.)

Given AON 41304 of Process 610's SS object 10336, the Virtual Processor Management System can locate that portion of Process 610's state which is moved into registers belonging to JP 10114 when process 610's Virtual Processor 612 is bound to JP 10114. Similarly, when Virtual Processor 612 is unbound from JP 10114, the virtual processor management system can move the contents of JP 10114 registers into the proper location in SS Object 10336.

a. Virtual Processor Managment (Fig. 453)

EOS can perform six operations on Virtual Processors 612:

— Request VP allows EOS to request a Virtual Processor 612 from KOS.
— Release VP allows EOS to return a Virtual Processor 612 to KOS.
— Bind binds a Process 610 to a Virtual Processor 612.
— Unbind unbinds a process 610 from a Virtual Processor 612.
— Run allows KOS to bind Process 610's Virtual Processor 612 to JP 10114.
— Stop prevents KOS from binding process 610's Virtual Processor 612 to JP 10114.

As can be seen from the above list of operations, EOS has no direct influence over the actual binding of a Virtual Processor 612 to JP 10114. This operation is performed by a component of KOS microcode called the Dispatcher. Dispatcher microcode is executed whenever one of four things happens:

— Process 610 whose Virtual Processor 612 is currently bound to JP 10114 executes an Await Operation.
— Process 610 whose Virtual Processor 612 is currently bound to JP 10114 executes an Advance Operation which satisfies an Await Entry 44801 for some other Process 610.
— Either Interval Timer 25410 or Egg Timer 25412 overflows, causing an Event Signal which invokes Dispatcher microcode.
— IOJP Bus 10132 is activated, causing an Event Signal which invokes Dispatcher microcode. IOS 10116 activates IOJP bus 10132 when it loads data into MEM 10112 for JP 10114.

When Dispatcher microcode is invoked by one of these events, it examines lists in MVPL 45309 to determine which Virtual Processor 612 is to run next. For the purposes of the present discussion, only two lists are important: the running list and the eligible list. In the present embodiment, the running list, headed by Running List Head 45321, contains only a single MVPLE 45321, that representing Virtual Processor 612 currently bound to JP 10114. In embodiments with multiple JPs 10114, the running list may have more than one MVPLE 45321. The eligible list, headed by Eligible List Head 45313, contains MVPLEs 45321 representing those Virtual Processors 612 which may be bound to JP 10114. MVPLEs 45321 on the eligible list are ordered by priorities assigned Processes 610 by EOS. Whenever KOS Dispatcher microcode is invoked, it compares the priority of Process 610 whose Virtual Processor 612's MVPLE 45321 is on the running list with the priority of Process 610 whose Virtual Processor 612's MVPLE 45321 is at the head of the eligible list. If the latter Process 610 has a higher priority, KOS Dispatcher microcode places MVPLE 45321 belonging to the former Process 610's Virtual Processor 612 on the eligible list and MVPLE 45321 belonging to the latter Process 610's Virtual Processor 612 onto the running list. Dispatcher microcode then swaps Processes 610 by moving state in JP 10114 belonging to the former Process 610 onto the former Process 610's SS object 10336 and moving JP 10114 state belonging to the latter Process 610 from the latter Process 610's SS object 10336 into JP 10114.

b. Virtual Processors 612 and Synchronization (Fig. 454)

When a synchronization operation is performed on a Process 610, one of the consequences of the operation is a synchronization operation on a Virtual Processor 612. For example, an Advance Operation which satisfies an Await Entry 44804 for a Process 610 causes an Advance Operation which satisfies a second Await Entry 44804 for Process 610's Virtual Processor 612. Similarly, a synchronization operation performed on a Virtual Processor 612 may have a synchronization operation on Virtual Processor 612's Process 610 as a consequence. For example, if a Virtual Processor 612 performs an operation involving file I/O, Virtual Processor 612's Process 610 must await the completion of the I/O operation.

Figure 454 illustrates the means by which process level synchronization operations result in virtual processor-level synchronization operations and vice-versa. The discussion first describes the components which transmit process-level synchronization operations to Virtual Processors 612 and the manner in which these components operate. Then it describes the components which transmit virtual processor-level

synchronization operations to Processes 610 and the operation of these components.

The first set of components is made up of VPSBA 45301 and VPAT 45401. VPSBA 45301 is shown here with two VPSBs 614: one belonging to a Virtual Processor 612 bound to a user Process 610 and one belonging to a Virtual Processor 612 bound to the KOS Process Manager process 610. VPAT 45401 is a virtual processor-level table of Await Entries 44804. Each Await Entry 44804 is contained in a VPAT Entry (VPATE) 45403. Each Virtual Processor 612 bound to a Process 610 has a VPAT Chunk 45402 of four VPATEs 45403 in VPAT 45401, and can thus await up to four Event Counters 44801 at any given time. The location of a Virtual processor 612's VPAT Chunk 45402 is kept in Virtual Processor 612's VPSB 614. When an Advance Operation satisfies any of the Await Entries 44804 belonging to a Virtual Processor 612, all in Virtual Processor 612's VAT Chunk 45402's Await Entries 44804 are deleted. As in PET 44705, VPATES 45403 containing Await Entries 44804 which are awaiting a given Event Counter 44801 are linked together in a list.

VPATEs 45403 for Virtual Processors 612 bound to user Processes 610 may contain Await Entries 44804 for user Process 610's Private Event Counter 45405. Private Event Counter 45405 is contained in Process 610's Process Object 901. It is advanced each time an Await Entry 44804 in a PETE 44909 on a PET List belonging to Process 610 is satisfied.

The components operate as follows: When KOS performs an Await Operation on Process 610, it makes Await Entries 44804 in both PET 44705 and VPAT 45401 and puts Process 610's VP 612 on the suspended list in MVPL 45309. As previously described, an Await Entry 44804 in PET 44705 awaits an Event Counter 44801 specified in the Await Operation which created Await Entry 44804. Await Entry 44804 in VPAT 45401 awaits Process 610's Private Event Counter 45405. Each time an Await Entry 44804 belonging to Process 610 in PET 44705 is satisfied, Process 610's Private Event Counter 45405 is advanced. The advance of Private Event Counter 45405 satisfies Await Entry 44801 for Process 610's Virtual processor 612 in VPAT 45401, and consequently, KOS deletes Virtual Processor 612's VPATEs 45403 and moves Virtual Processor 612's MVPLE 45321 in MVPL 45309 from the suspended list to the eligible list.

The components which allow a Virtual Processor 612 to transmit a synchronization operation to a process 610 are the following: Outward Signals Object (OSO) 45409, Multiplexed Outward Signals Event Counter 45407, and PET 44705. OSO 45409 contains Event Counters 44801 which KOS FU 10120 microcode advances when it performs operations which user Processes 610 are awaiting. Event Counters 44801 in OSO 45409 are awaited by Await Entries 44804 in PET 44705. Each time KOS FU 10120 microcode advances an Event Counter 44801 in OSO 45409, it also advances Multiplexed Outward Signals Event Counter 45407. It is awaited by an Await Entry 44804 in VPAT 45401 belonging to Virtual Processor 612 bound to KOS Process Manager Process 610. When Virtual Processor 62 bound to KOS Process Manager Process 610 is again bound to JP 10114, KOS Process Manager Process 610 examines all PETEs 44909 belonging to the Event Counters 44801 in OSO 45423. If an advance of an Event Counter 44801 in OSO 44801 satisfied a PETE 44909 Process 610, that Process 610's Private Event Counter 45405 is advanced as previously described, and Process 610 may again execute.

A user I/0 operation illustrates how the components work together. Each user I/0 channel has an Event Counter 44801 in OSO 45409. When a Process 610 performs a user I/O operation on a channel, the EOS I/O routine establish an Await Entry 44804 in the PET 44705 list belonging to Process 610 for the channel's Event Counter 44801 in OSO 45409. When the I/O operation is complete, IOS 10116 places a message to JP 10114 in an area of MEM 10112 and activates IOJP Bus 10132. The activation of IOJP Bus 10132 causes an Event Signal which invokes KOS microcode. The microcode examines the message from IOS 10116 to determine which channel is involved, and then advances Event Counter 44801 for that channel in OSO 45409 and Multiplexed Outward Signals Event Counter 45407. The latter advance satisfies an Await Entry 44804 for Process Manager Process 610's Virtual Processor 612 in VPAT 45401, and Process Manager Process 610 begins executing. Process Manager Process 610 examines OSO 45409 to determine which Event Counters 44801 in OSO 45409 have been advanced since the last time process manager Process 610 executed, and when it finds such an Event Counter 44801, it examines the Event Counter Chain in PET 44705 for that Event Counter 44801. If it finds that the advance satisfied any Await Entries 44804 in the Event Counter Chain, it advances Private Event Counter 45405 belonging to Process 610 specified in Await Entry 44804, thereby causing that Process 610 to resume execution as previously described.

F. Process 610 Stack Manipulation

This section of the specification for CS 10110 describes the manner in which Process 610's MAS 502 and SS 504 are manipulated. As previously mentioned, in CS 10110, a Process 610's MAS 502 and SS 504 are contained in several objects. In the present embodiment, there are five objects, one for each domain's portion of the Macro Stack (MAS) (MAS Objects 10328 through 10324) and one for the Secure Stack (SS) (SS Object 10336). In other embodiments, a Process 610's MAS 502 may contain objects for user-defined domains as well. Though a Process 610's MAS 502 and SS 504 are contained in many objects, they function as a single logical stack. The division into several objects is a consequence of two things: the domain component of the protection system, which requires that an object referenced by a Procedure 602 have Procedure 602's domain of execution, and the need for a location inaccessible to user programs for micromachine state and state which may be manipulated only by KOS.

Stack manipulation takes place under the following circumstances:

— When a procedure 602 is invoked or a Return SIN is executed. Procedure 602 invocations are

performed by means of a Call SIN. Call causes a transfer of control to the first SIN in the invoked Procedure 602 and the Return SIN causes a transfer of control back to the SIN in the invoking Procedure 602 which follows the Call SIN.

— When a non-local Go To SIN is executed. The non-local Go To causes a transfer of control to an arbitrary position in some Procedure 602 which was previously invoked by Process 610 and whose invocation has not yet ended.
— When a condition arises, i.e., an execution of a statement in a program puts the executive Process 610 into a state which requires the execution of a previously established Handler Procedure 602.
— When a Process 610 is interrupted, i.e., when an Interrupt Entry 45718 for Process 610 is satisfied.

Most of the mechanisms involved in stack manipulation are used in Call and Return; these operations are therefore dealt with in detail and the other operations only as they differ from Call and Return. The discussion first introduces Call and Return, then explains the stacks in detail, and finally analyzes Call and Return and the other operations in detail.

1. Introduction to Call and Return

As a Process 610 executes a program, it executes Call and Return SINs. A Call SIN begins an invocation of a procedure 602, and a Return SIN ends the invocation. Generally speaking, a Call SIN does the following:

— It saves the state of Process 610's execution of Procedure 602 which contains the Call SIN. Included in this state is the information required to continue Procedure 602's execution after the Call SIN is finished. This portion of the state is termed calling Procedure 602's Macrostate.
— It creates the state which Process 610 requires to begin execution called Procedure 602.
— It transfers control to the first SIN in the called Procedure 602's code.

The Return SIN does the opposite: it releases the state of called Procedure 602, restores the saved state of calling Procedure 602, and transfers control to the SIN in the calling Procedure 602 following the Call SIN. An invocation of a Procedure 602 lasts from the execution of the Call SIN which transfers control to the Procedure 602 to the execution of the Return SIN which transfers control back to Procedure 602 which contained the Call SIN. The state belonging to a given invocation of a Procedure 602 by a Process 610 is called Procedure 602's invocation state.

While Calls and Returns may be implemented in many different fashions, it is advantageous to implement them using stacks. When a Call creates invocation state for a Procedure 602, that invocation state is added to the top of Process 610's stack. The area of a stack which contains the invocation state of a Procedure 602 is called a frame. Since a called Procedure 602 may call another procedure 602, and that another, a stack may have any number of frames, each frame containing the invocation state resulting from the invocation of a Procedure 602 by Process 610, and each frame lasting as long as the invocation it represents. When called Procedure 602 returns to its caller, the frame upon which it executes is released and the caller resumes execution on its frame. Procedure 602 being currently executed by a Process 610 thus always runs on the top frame of Process 610's MAS 502.

Calls and Returns in CS 10110 behave logically like those in other computer systems using stacks to preserve process 610 state. When a Process 610 executes a Call SIN, the SIN saves as Macrostate the current values of the ABPs, the location of the SIN at which the execution of calling Procedure 602 is to continue, and information such as a pointer to calling Procedure 602's Name Table 10350 and UID 40401 belonging to the S-interpreter object which contains the S-interpreter for Procedure 602's S-language. The Call SIN then creates a stack frame for called Procedure 602, obtains the proper ABP values, the location of called Procedure 602's Name Table 10350 and UID 40401 belonging to its S-interpreter object, and begins executing newly-invoked Procedure 602 on the newly-created stack frame. The Return SIN deletes the stack frame obtains the ABP values and name interpreter information from the Macrostate saved during the Call SIN and then transfers control to the SIN at which execution of calling Procedure 602 is to continue.

However the manner in which Call and Return are implemented is deeply affected by CS 10110's Access Control System Broadly speaking there are two classes of Calls and Returns in CS 0110: those which are mediated by KOS and those which are not. In the following discussion, the former class of Calls and Returns are termed Mediated Calls and Returns, and the latter are called Neighborhood Calls and Returns. Most Calls and Returns executed by CS 10110 are Neighborhood Calls and Returns; Mediated Calls and Returns are typically executed when a user procedure 602 calls EOS Procedures 602 and these in turn call KOS Procedures 602. The Mediated Call makes CS 10110 facilities available to user Processes 610 while protecting these CS 10110 facilities from misuse and therefore generally serves the same purpose as system calls in the present art. As will be seen in the ensuing discussion, Mediated Call requires more CS 10110 overhead than Neighborhood Call but the extra overhead is less than that generally required by system calls in the present art.

Mediated Calls and Returns involve S-interpreter, Namespace, and KOS microcode. S-interpreter and Namespace microcode interpret the Names involved in the call and only modifies those portions of Macrostate accessible to the S-interpreter. The remaining Macrostate is modified by KOS microroutines invoked in the course of the Call SIN. A Mediated Call may be made to any Procedure 602 contained in an object to which Process 610's subject has Execute Access at the time the invocation occurs. Mediated Calls and Returns must be made in the following situations:

— When called Procedure 602 has a different Procedure Environment Descriptor (PED) 30303 from that used by calling Procedure 602. Such Calls are termed Cross-PED Calls.
— When called Procedure 602 is in a different Procedure Object 608 from calling Procedure 602. Such Calls are termed Cross-Procedure Object Calls.
— When called Procedure 602's Procedure Object 608 has a different Domain of Execution (DOE) Attribute from that of calling Procedure 602's Procedure Object 608, and therefore must place its Invocation State on a different MAS object from that used by calling Procedure 602. Such Calls are termed Cross-Domain Calls.

In all of the above Calls, the information required to complete the Call is not available to the S-interpreter and consequently. KOS mediation is required to complete the Call. Neighborhood Calls and Returns only modify two components of Macrostate: the pointer to the current SIN and the FP ABP. Both of these components are available to the S-interpreter as long as called Procedure 602 has the same PED 30303 i.e., uses the same Name Tabe 10350 and S-interpreter or the calling Procedure 602 and has Names with the same syllable size as calling Procedure 602. The Call and Return SINs are specific to each S-language, but they resemble each other in their general behavior. The following discussion will deal exclusively with this general behavior and will concentrate on Mediated Calls and Returns. The discussion first describes MAS 502 and SS 504 belonging to a Process 610 and those parts of Procedure Object 608 involved in Calls and Returns, and then describes the implementation of Calls and Returns.

2. Macro Stacks (MAS) 502 (Fig. 467)

Figure 467 gives an overview of an object belonging to a Process 610's MAS 502. The description of this Figure will be followed by descriptions of other Figures containing detailed representations of portions of MAS objects.

At a minimum MAS Object 46703 comprises KOS MAS Header 10410 together with Unused Storage 46727 reserved for the other elements comprising MAS Object 46703. If Process 610 has not yet returned from an invocation of a Procedure 602 contained in a Procedure Object 608 whose DOE is that required for access to MAS Object 46703. MAS object 46703 further comprises a Stack Base 46703 and at least one MAS Frame 46709.

Each MAS Frame 46709 represents one mediated invocation of a procedure 602 contained in a Procedure Objece 608 with the DOE attribute required by MAS 46703, and may in addition represent neighborhood invocations of Procedures 602 which share that Procedure 602's Procedure Object 608. The topmost MAS Frame 46709 represents the most recent group of invocations of Procedures 602 with the DOE attribute required by MAS Object 46703 and the bottom MAS Frame 46709 the earliest group of invocations from which Process 610 has not yet returned. Frames for invocations of Procedures 602 with other domains of execution are contained in other MAS Objects 46703. As will be explained in detail below MAS Frames 46709 in different MAS objects 46703 are linked by pointers.

MAS Domain Stack Base 46703 has two main parts: KOS MAS Header 10410 which contains information used by KOS microcode which manipulates MAS Object 46703, and Perdomain Information 46707, which contains Information about 46703's domain and static information, i.e., information which lasts longer than an invocation used by Procedures 602 with MAS Frames 46709 on MAS Object 46703. MAS Frame 46709 also has two main parts, a KOS Frame Header 10414 which contains information used by KOS to manipulate Frame 46709 and S-interpreter Portion 46713 which contains information available to the S-interpreter when it executes the group of Procedures 602 whose invocations are represented by Frame 46709.

When making Calls and Returns, the S-interpreter and KOS microcode use a group of pointers to locations in MAS Object 46703. These pointers comprise the following:
— MAS Object UID 46715 the UID 40401 of AS Object 46703.
— First Frame Offset (FFO) 46719 which locates the beginning of KOS Frame Header 10414 belonging to the first MAS Frame 46709 in MAS Object 46703.
— Frame Header Pointer (FHP) 46702 which locates the beginning of the topmost KOS Frame Header 10414 in MAS Object 46703.
— Stack Top Offset (STO) 46704 a 32-bit offset from Stack UID 46715 which marks the first bit in Unused Storage 46727.

As will be seen presently all of these pointers are contained in fields in KOS MAS Header 46705.

a.a. MAS Base 10410 (Fig. 468)

Figure 468 is a detailed representation of MAS Domain Stack Base 10410 Turning first to the detailed representation of KOS MAS Header 46705 contained therein, there are the following fields:
— Format Information Field 46801 containing information about the format of KOS MAS Header 46705.
— Flags Field 46803. Of these flags, only one is of interest to the present discussion: Domain Active Flag 46804. This flag is set to TRUE when Process 610 to which MAS Object 46703 belongs is executing the Invocation of Procedure 602 whose invocation record makes up the topmost MAS Frame 46709 contained in MAS Object 46703 to which KOS MAS Header 46705 belongs.
— PFO Field 46805: All MAS Headers 46705 and Frame Headers 46709 have fields containing offsets locating the previous and following headers in MAS Object 46703. In a Stack Header 46705 there is no

previous header and this field is set to 0.

- FFO Field 46805: The field locating the following header In a Stack Header 46705 this field contains FFO 46719 since the next header is the first Frame Header in MAS Object 46703.
- STO Field 46807: the field containing STO offset 46704.
- Process ID Field 46809: UID 40401 belonging to Process Object 901 for Process 610 to which MAS Object 46703 belongs.
- Domain Environment Information pointer Field 46811: The pointer contained in the field locates an area which contains domain-specific information. In the present embodiment. the area is part of MAS Stack Base 10410; however, in other embodiments, it may be contained in a separate object.
- Signaller Pointer Field 46813: The pointer contained in the field locates a Procedure 602 which KOS invokes when a Process 610's execution causes a condition to arise while it is executing in the domain to which MAS object 46703 belongs.
- AAT Pointer Field 30211: The pointer in Field 30211 locates AAT 30201 for MAS Object 46703. AAT 30201 is described in detail in Chapter 3.
- Frame Label Sequencer Field 46819: This field contains a Sequencer 45102. Sequencer 45102 is used to generate labels used to locate MAS Frames 46709 when a non-local GOTO is executed.

Turning now to the detailed representation of Domain Environment Information 46821 located by Domain Environment Information Pointer Field 46811 there are the following fields:

- KOS Format Information Field 46823.
- Flags Field 46825 containing the following flags:
  - Pending Interrupt Flag 46827, set to TRUE when Process 610 has an interrupt pending for the domain to which MAS Object 46703 belongs.
  - Domain Dead Flag 46829, set to TRUE when Process 610 can no longer execute Procedures 602 with domains of execution equal to that to which MAS Object 46703 belongs.
  - Invoke Verify on Entry Flag 46833 and Invoke Verify on Exit Flag 46835. The former flag is set to TRUE when KOS is to invoke a Procedure 602 which checks the domain's data bases before a Procedure 602 is allowed to execute on the domain's MAS Object 46703; the latter is set to TRUE when KOS is to invoke such a Procedure 602 on exit from a Procedure 602 with the domain as its DOE.
  - Default Handler Non-null Flag 46835 is set to TRUE when there is a default clean-up handler for the domain. Clean-up handlers are described later.
  - Interrupt Mask Field 46839 determines what interrupts set for Process 610 in MAS object 46703's domain will be honored.
  - Domain UID Field 46841 contains UID 40401 for the domain to which MAS Object 46703 belongs.
  - Fields 46843 through 46849 are pointers to Procedures 602 or tables of pointers to Procedures 602. The Procedures 602 so located handle situations which arise as MASs 502 are manipulated. The use of these fields will become clear as the operations which require their use are explained.

b.b. Per-domain-Data Area 46853 (Fig. 468)

Per-domain Data Area 46853 contains data which cannot be kept in MAS Frames 46709 belonging to invocations of Procedures 602 executing in MAS Object 46703's domain, but which must be available to these invocations Per-Domain Data Area 46853 has two components: Storage Area 46854 and AAT 30201. Storage Area 46854 contains static data used by Procedures 602 with invocations on MAS Object 46703 and data used by S-interpreters which are used by such procedures 602. Associated Address Table (AAT) 30201 is used to locate data in Storage Area 46854. A detailed discussion of AAT 30201 is contained in Chapter 3.

Two kinds of data is stored in Storage Area 46854: static data and S-interpreter data.

Static data is stored in Static Data Block 46863. Static Data Block 46863 comprises two parts: Linkage Pointers 46865 and Static Data Storage 46867 Linkage Pointers 46865 are pointers to static data not contained in Static Data Storage 46867 for example, data which lasts longer than Process 610 and pointers to External Procedures 602 which the Procedure 602 to which Static Data Storage 46867 belongs invokes. Static Data Storage 46867 contains storage for static data used by the Procedure 602 which does not last longer than Process 610 executing the Procedure 602.

S-interpreter data is data required by S-interpreters used by Procedures 602 executing on MAS object 46703.

The S-interpreter data is stored in S-interpreter Environment Block (SEB) 46864 which, like Static Data Block 46864 is located via AAT 30201: The contents of SEB 46864 depend on the S-interpreter.

c.c. MAS Frame 46709 Detail (fig. 469)

Figure 469 represents a typical frame in MAS Object 46703. Each MAS Frame 46709 contains a Mediated Frame 46947 produced by a Mediated Call of a Procedure 602 contained in a Procedure Object 608 whose DOE attribute is the one required for execution on MAS object 46703. Mediated Frame 46947 may be followed by Neighborhood Frames 46945 produced by Neighborhood Calls of Procedures 602. Mediated Frame 46947 has two parts, a KOS Frame Header 10414 which is manipulated by KOS microcode, and an S-interpreter portion which is manipulated by S-interpreter and Namespace microcode. Neighborhood Frames 46945 have no KOS Frame Headers 10414. As will become clear upon closer

examination of Figure 469. Mediated Frames 46947 in the present embodiment contain no Macrostate. In the present embodiment, Macrostate for these frames is kept on SS Object 10336; however in other embodiments, Macrostate may be stored in Mediated Frames 46947. Neighborhood Frames 46945 contain those portions of the macrostate which may be manipulated by Neighborhood Call; the location of this macrostate depends on the Neighborhood Call SIN.

Turning now to KOS Frame Header 10414. there are the following fields:

- KOS Format Information Field 46901 containing information about MAS Frame 46709's format.
- Flags Field 46902. This field contains the following flags:
  - Result of Cross-domain Call Flag 46903. This Flag is TRUE if MAS Frame 46709 which precedes this MAS Frame 46709 is in another MAS Object 46703.
  - Is Signaller Flag 46905. This flag is TRUE if this MAS Frame 46709 was created by the invocation of a Signaller Procedure 602.
  - Do Not Return Flag 46907: This flag is TRUE if Process 610 is not to return to the invocation for which this MAS Frame 46709 was created.
  - Flags 46909 through 46915 indicate whether various lists used in condition handling and non-local GOTOs are present in the MAS Frame 46709.
- Previous Frame Offset Field 46917. Next Frame Offset Field 46919. and Frame Top Offset Field 46921 are offsets which give the location where Header 10414 for the previous MAS Frame 46709 in MAS Object 46703 begins. the location where the header for the next MAS Frame 46709 in MAS Object 46703 begins, and the location of the first bit beyond the top of MAS Frame 46709 respectively.
- Fields 46923 through 46927 are offsets which locate lists in S-interpreter portion 46713 of Frame 46709. KOS establishes such lists to handle conditions and non-local GOTOs. Their use will be explained in detail under those headings.
- Fields 46929 and 46933 contain information about Procedure 602 whose invocation is represented by MAS Frame 46709. Field 46929 contains the number of arguments required by procedure 602 and Field 46933 contains a resolvable pointer to Procedure 602's PED 30303. Both these fields are used primarily for debugging.
- Dynamic Back Pointer Field 46931 contains a resolvable pointer to the preceding MAS Frame 46709 belonging to Process 610's MAS 502 when that MAS Frame 46709 is contained in a different MAS Object 46703. In this case, Flag Field 46903 is set to TRUE. When the preceding MAS Frame 46709 is contained in the same MAS object 46703 field 46931 contains a pointer with a null UID 40401 and Flag Field 46903 is set to FALSE.
- Frame Label Field 46935 is for a Frame Label produced when a non-local GOTO is established which transfers control to the invocation represented by MAS Frame 46709. The label is generated by Frame Label Sequencer 46819 in KOS MAS Header 10410.

S-interpreter Portion 46713 of MAS Frame 46709 comprises those portions of MAS Frame 46709 which are under control of the S-interpreter. S-interpreter Portion 46713 in turn comprises two main subdivisions: those parts belonging to Mediated Frame 46947 and those belonging to Neighborhood Frames 46945.

The exact form of S-interpreter portion 46949 of KOS Frame 46947 and of S-Interpreter Frames 46945 depends on the Call SIN which created the frame in question. However all Neighborhood Frames 46945 and S-interpreter portions 46949 of Mediated Frames 46947 have the same arrangements for storing Linkage Pointers 10416 and local data in the frame. Linkage Pointers 10416 are pointers to the locations of actual arguments used in the invocation and Local Storage 10420 contains data which exists only during the invocation. In all Mediated Frames 46947 and Neighborhood Frames 46945. Linkage pointers 10416 precede Local Storage 10420. Furthermore, when a Mediated Frame 46947 or a Neighborhood Frame 46945 is the topmost frame of Process 610's MAS. i.e, when Process 610 is executing on that frame, the FP always points to the beginning of Local Storage 10420. and the beginning of Linkage Pointers 10416 is always at a known displacement from FP. References to Linkage Pointers 10416 may therefore be expressed as negative offsets from FP, and references to Local Storage 10420 as positive offsets.

In addition, S-interpreter Portion 46713 may contain lists of information used by KOS to execute non-local GOTOs and conditions, as well as S-interpreter frames for non-mediated calls. The lists of information used by KOS are contained in List Area 46943. The exact location of List Area 46943 is determined by the compiler which generates the SINs and Name Table for the Procedure 602 whose invocation is represented by Mediated Frame 46947. When Procedure 602's source text contains statements requiring storage in List Area 46943, the compiler generates SINs which place the required amount of storage in Local Storage 10420. KOS routines then build lists in Area 46943, and place the offsets of the heads of the lists in Fields 46923, 46925 or 46927, depending on the kind of list. The lists and their uses are described in detail later.

3. SS 504 (Fig. 470)

Figure 470 presents an overview of SS 504 belonging to a Process 610. SS 504 is contained in SS Object 10336. SS Object 10336 is manipulated only by KOS microcode routines. Neither Procedures 602 being executed by Process 610 nor S-interpreter or Namespace microcode may access information contained in SS Object 10336.

SS Object 10336 comprises two main components, SS Base 47001 and SS Frames 47003. Turning first to the general structure of SS Frames 47003, each time a Process 610 executes a Mediated Call KOS

microcode creates a new SS Frame 47003 on SS Object 10336 belonging to Process 610 and each time a Process 610 executes a Mediated Return, KOS microcode removes the current top SS Frame 47003 from SS Object 10336. There is thus one SS Frame 47003 on SS Object 10336 belonging to a process 610 for each Mediated Frame 46947 on Process 610's MAS 502.

SS Frames 47003 comprise two kinds of frames:

Ordinary Frames 10510 and Cross-domain Frames 47039. Cross-domain Frames 47039 are created whenever Process 610 executes a Cross-domain Call; for all other Mediated Calls. Ordinary Frames 10510 are created. Cross-domain Frames 47039 divide SS Frames 47003 into Groups 47037 of SS Frames 47003 belonging to sequences of invocations in a single domain. The first SS Frame 47003 in a Group 47037 is a Cross-domain Frame 47039 for the invocation which entered the domain, and the remainder of the SS Frames 47003 are Ordinary Frames 10510 for a sequence of invocations in that domain. These groups of SS Frames 47003 correspond to groups of Mediated Frames 46947 in a single MAS Object 46703.

a.a. SS Base 47001 (Fig. 471)

SS Base 47001 comprises four main parts: SS Header 10512 Process Microstate 47017, Storage Area 47033 for JP 10114 register contents, and Initialization Frame Header 47035. Secure Stack Header 10512 contains the following information:

- Fields 47001 and 47009 contain flag and format information; the exact contents of these fields are unimportant to the present discussion.
- Previous Frame Offset Value Field 47011 is a standard field in headers in SS Object 10336; here it is set to 0, since there is no previous frame.
- Secure Stack First Frame Offset Field 47013 contains the offset of the first SS Frame 47039 in SS object 10336, i.e., Initialization Frame Header 47035.
- Process UID field 47015 contains UID 40401 of Process 610 to which SS Object 10336 belongs.
- Number of Cross Domain Frames Field 47016 contains the number of Cross-domain Frames 47039 in SS Object 10336.

Process Microstate 47017 contains information used by KOS microcode when it executes Process 610 to which SS Object 10336 belongs Fields 47019, 47021 and 47022 contain the offsets of locations in SS Object 10336. Field 47019 contains the value of SSTO the location of the first free bit in SS Object 10336; Field 47021 contains the value of SSFO, the location of the topmost frame in SS object 10336; Field 47022 finally contains the value of XDFO, the location of the topmost Cross-domain Frame 47039 in SS Object 10336. All of these locations are marked in Figure 470.

Other fields of interest in Process Microstate 47017 comprise the following: Offsets in Storage Area Field 47023 contains offsets of locations in Storage Area 47033 of SS Object 10336; Domain Number Field 47025 contains the domain number for the DOE of Procedure 602 currently being executed by Process 610. The relationship between domain UIDs and domain numbers is explained in the discussion of domains. VPAT Offset Field 47027 contains the offset in VPAT 45401 of VPAT Chunk 45402 belonging to Virtual Processor 612 to which Process 610 is bound. Signal Pointer Field 47029 contains a resolved pointer to the Signaller (a Procedure 602 used in condition handling) belonging to the domain specified by Domain Number Field 47025 and Trace Information Field 47031 contains a resolved pointer to that domain's Trace Table, described later.

Storage Area for JP 10114 register Contents 47033 is used when a Virtual Processor 612 must be removed from JP 10114. When this occurs, either because Virtual Processor 612 is unbound from JP 10114, because CS 10110 is being halted, or because CS 10110 has failed, the contents of JP 10114 registers which contain information specific to Virtual Processor 612 are copied into Storage Area 47033. When Virtual Processor 612 is returned to JP 10114, these register contents are loaded back into the JP 10114 registers from whence they came. Initialization Frame Header 47035, finally, is a dummy frame header which is used in the creation of SS Object 10336.

b.b. SS Frames 47003 (Fig. 471)

Commencing the discussion of SS Frames 47039 and 10510. Figure 471 illustrates these structures in detail. Ordinary SS Frame 10510 comprises three main divisions: Ordinary SS Frame Header 10514, Macrostate 10516 and Microstate 10520. Ordinary SS Frame Header 10514 contains information used by KOS microcode to manipulate Ordinary SS Frame 10510 to which Header 10514 belongs. Macrostate 10516 contains the values of the ABPs for the frame's mediated invocation and other information required to resume execution of the invocation. Microstate 10520 contains micromachine state from FU 10120 and EU 10122 registers. The amount of micromachine state depends on the circumstances; in the present embodiment, some micromachine state is saved on all Mediated Calls; furthermore, if a Process 610 executes a microcode-to-software Call, the micromachine state that existed at the time of the call is saved; finally. Microstate 10520 belonging to the topmost SS Frame 47003 may contain information which was transferred from FU 10120 GRF registers 10354 or EU 10122 register and stack mechanism 10216 when their capacity was exceeded. For details about this portion of Microstate 10520 see the discussion of the FU 10120 micromachine in Chapter 2. The discussion of SS Object 10336 continues with details concerning SS Header 10514 and Macrostate 05163.

a.a.a. Ordinary SS Frame Headers 10514 (Fig. 741)

Fields of interest in Ordinary Secure Stack Frame Header 10514 are the following:

— Format Information 47103 which identifies the format of Header 10514.
— Flags Field 47105 which contains one flag of interest in this discussion: Frame Type Flag 47107: in Ordinary SS Frames 10510 this field is set to FALSE.
— Offset Fields 47109 through 47113: Field 47109 contains the offset of the previous SS Frame 47039 or 10510. Field 47111 contains the offset of the following SS Frame 47039 or 10510. and Field 47113 contains the offset of the last SS Frame 47039 or 10510 preceding the next Crossdomain Frame 47039
— Field 47117 contains the current domain number for the domain in which the mediated invocation represent SS Frame 47039 or 10510 is executing.
— Field 47119 contains the offset of the preceding Cross-domain Frame 47039.
— Field 47121 contains offsets for important locations in Microstate 10520.

b.b.b. Detailed Structure of Macrostate 10516 (Fig. 471)

These fields are of interest in Macrostate 10516:

— Syllable Size Field 47125 contains the value of K, i.e., the size of the Names in the SINs belonging to Procedure 602 which the invocation is executing.
— End of Name Table Field 47127 contains the location of the last Name in Name Table 10350 belonging to Procedure 602 which the invocation is executing.
— Fields 47129 through 47143 are resolved pointers to locations in Procedure Object 901 containing Procedure 602 being executed by the invocation and resolved pointers to locations containing data being used by Procedure 602. Field 47129 contains a pointer to Procedure 602's PED 30303; if Procedure 602 is an External Procedure 602. Field 47131 contains a pointer to Procedure 602's entry in Gates 10340; Field 47135 contains the UID-offset value of FP for the invocation; Field 47135 contains a pointer to SEB 46864 used by Procedure 602's S-interpreter. Field 47137 contains the UID-offset value of SDP and Field 47139 contains that of PBP. SIP Field 47141 contains a pointer to Procedure 602's S-interpreter object, and NTP, finally, is a pointer to Procedure 602's Name Table 10350.
— Field 47145 contains the PC for the SIN which is to be executed on return from the mediated invocation to which SS Frame 47003 belongs.

c.c.c. Cross domain SS Frames 47039 (Fig. 471)

Cross-domain SS Frames 47039 differ from Ordinary SS Frames 10510 in two respects: they have an additional component, Cross-domain State 10513, and fields in Cross domain Frame Header 47157 have different meanings from those in Ordinary Frame Header 10514.

Cross-domain State 10513 contains information which KOS Call microcode uses to verify that a return to a Procedure 602 whose DOE differs from that of Procedure 602 whose invocation has ended is returning to the proper domain. Fields of interest in Cross-domain State 10513 include GOTO Tag 47155 used for non-local GOTOs which cross domains, Stack Top Pointer Value 47153, which gives the location of the first free bit in the new domain's MAS Object 46703 and Frame Header Pointer Value 47151, which contains the location of the topmost Mediated Frame Header 46709 in new MAS Object 46703.

There are three fields in Cross-domain Frame Header 47157 which differ from those in Ordinary SS Frame Header 47101. These fields are Flag Field 47107 which in Cross-domain Frame Header 47157 always has the value TRUE, preceding Cross-domain Frame Offset Field 47161, which contains the offset of preceding Cross-domain Frame 47039 in SS Object 10336 and Next Cross domain Frame Offset Field 47159, which contains the location of the next Cross-domain Frame 47039. These last two fields occupy the same locations as Fields 47111 and 47109 respectively in Ordinary SS Frame Header 10514.

As will be noted from the above description of SS Frames 47003. Secure Stack Object 10336 in the present embodiment contains three kinds of information: macrostate cross-domain state and microstate. In other embodiments, the information in SS object 10336 may be stored in separate stack structures, for example, separate microstate and cross-domain stacks, or information presently stored in MAS Objects 46703 may be stored in SS Object 10336, and vice-versa.

4. Portion of Procedure Object 608 Relevant to Call and Return (Fig. 472)

The information which Process 610 requires to construct new frames on its MAS Objects 46703 and SS Object 10336 and to transfer control to invoked Procedure 602 is contained in invoked Procedure 602's Procedure Object 608. Figure 472 is an overview of Procedure Object 608 showing the information used in a Call. Figure 472 expands information contained in Figures 103 and 303; fields that appear in those Figures have the names and numbers used there.

Beginning with Procedure Object Header 10336, this area contains two items of information used in Calls: an offset in Field 47201 giving the location of Argument Information Array 10352 in Procedure Object 608 and a value in Field 47203 specifying the number of gates in Procedure Object 608. Gates allow the invocation of External Procedures 602 that is, Procedures 602 which may be invoked by Procedures 602 contained in other Procedure Objects 608. Procedure Object 608's gates are contained in External Entry Descriptor Area 10340. There are two kinds of gates: those for Procedures 602 contained in Procedure Object 608, and those for procedures 602 contained in other Procedure Objects 608, but callable via

Procedure Object 608. Gates for Procedures 602 contained in Procedure Object 608 are termed Local Gates 47205. Local Gates 47205 contain Internal Entry Offset (IEO) Field 47207 which contains the offset in Procedure Object 608 of Entry Descriptor 47227 for Procedure 602. If Procedure 602 is not contained in Procedure Object 472 its gate is a Link Gate 47206. Link Gates 47206 contain Binder Area Pointer (BAP) Fields 47208. A BAP Field 47208 contains the locations of an area in Binder Area 30323 which in turn contains a pointer to a Gate in another Procedure Object 608. The pointer in Binder Area 30323 may be either resolved or unresolved. If Procedure 602 is contained in that Procedure Object 608, the Gate is a Local Gate 47205; otherwise, it Is another Link Gate 47206.

Procedure Environment Descriptors (PEDS) 10348 contains PEDs 30303 for Procedures 602 contained in Procedure Object 608. Most of the macrostate information for a Procedure 602 may be found in its PED 30303. PED 30303 has already been described, but for ease of understanding, its contents are reviewed here.

- K Field 30305 contains the size of Procedure 602's Names.
- Largest Name (LN) Field 30307 contains the i

Beginning with Procedure Object Header 10336, this area contains two items of information used in Calls: an Offset in Field 47201 giving the location of Argument Information Array 10352 in Procedure Object 608 and a value in Field 47203 specifying the number of gates in Procedure Object 608. Gates allow the invocation of External Procedures 602, that is, Procedures 602 which may be invoked by Procedures 602 contained in other Procedure Objects 6nter to Static Data Block 46863. Thus, for that invocation of Procedure 602 on invocation, the SDP ABP is derived via SDPP field 30313.

- PBP Field 30315 is the pointer from which the current PC is calculated. When Procedure 602 is invoked, this value becomes the PBP ABP.
- S-interpreter Environment Prototype Pointer (SEPP) Field 30316 contains the location of SEB Prototype Field 30317. When Procedure 602 is invoked, Field 30316 locatas SEB 46864 via AAT 30201 in the same manner as SDPP field 30313 locates the invocation's static data.

A Procedure 602's PED 30303 may be located from its Internal Entry Descriptor 47227. A PED 30303 may be shared by several Procedures 602. Of course in this case, the values contained in shared PED 30303 are the same for all Procedures 602 sharing it. As will be explained in detail later in the present embodiment. if a calling Procedure 602 does not share a PED 30303 with called Procedure 602 the Call must be mediated. A calling Procedure 602 may make a Neighborhood Call only to Procedures 602 with which it shares a PED 30303.

The next portion of Procedure Object 608 which is of interest is Internal Entry Descriptors 10342. Each Procedure 602 contained in Procedure Object 608 has an Entry Descriptor 47227. Entry Descriptor 47227 contains four fields of interest:

- PBP Offset Field 47229 contains the offset from PBP at which the first SIN in Procedure 602's code is located.
- Flags Field 47230 contains flags which are checked when Procedure 602 is invoked. Four flags are of interest:
  - Argument Information Array Present Flag 47235 which is set to TRUE if Procedure 602 has entries in Argument Information Array 10352.
  - SEB Flag 47237 is set to TRUE if SEPP 47225 is non-null, i.e., if Procedure 602 has a SEB 46864 for its S-interpreter.
  - Do Not Check Access Flag 47239 is set to TRUE if KOS Call microcode is not to perform protection checking on the actual arguments used to invoke Procedure 602.
- PED Offset Field 47231 contains the offset of Procedure 602's PED 30303 from the beginning of Procedure Object 608.
- Frame Size Field 47233 contains the initial size of the Local Storage Portion 10420 of MAS Frame 46709 for an invocation of procedure 602.

Other areas of interest for Calls are SEB Prototype Area 47241, Static Data Area Prototype 30317, Binder Area 30323 and Argument Information Array 10352. SEB Prototype type Area 47241 and Static Data Area Prototype 30315 contain information used to create an SEB 46864 and Static Data Block 46863 respectively for Procedure 602. These areas are created on a per-MAS Object 46703 basis. The first time that a Process 610 executes a Procedure 602 in a domain, SEB 46864 and Static Data Block 46863 required for Procedure 602 are created either in MAS Object 46703 belonging to the domain or in another object accessible from MAS Object 46703. SEB 46864 and Static Data Block 46863 then remain as long as MAS Object 46703 exists.

Static Data Prototype 30317 contains two kinds of information: Static Data Links 30319 and Static Data Initialization Information 30321 Static Data Links 30319 contain locations in Binder Area 30323, which in turn contains pointers which may be resolved to yield the locations of data or External Procedures 602. When a Static Data Block 46863 is created for a Procedure 602, the information in Binder Area 30323 is used to create Linkage Pointers 46865. Static Data Initialization Information 30321 contains information required to create and initialize static data in Static Data Storage 46867.

As mentioned in the discussions of Link Gates 47206 and Static Data Links 30319 Binder Area 30323 contains pointers which may be resolved as described in Chapter 3 to yield locations of data and External Procedures 602.

Argument Information Array (AIA) 10352 contains information used by KOS Call microcode to check whether the subject which is invoking Procedure 602 has access to the actual arguments used in the invocation which allows the uses made of the arguments in Procedure 602. This so-called "Trojan horse check" is necessary because a Call may change the domain component of a subject. Thus, a subject which is lacking access of a specific kind to a data item could gain that access by passing the data item as an argument to a Procedure 602 whose DOE gives it access rights that the calling subject itself lacks.

Each Local Gate 47205 in Procedure Object 608 has an element in AIA 10352. Each of these Argument Information Array Elements (AIAEs) 60845 has fields indicating the following:

- The minimum number of arguments required to invoke Procedure 602 to which Local Gate 47205 belongs, in Field 47247.
- The maximum number of arguments which may be used to invoke Procedure 602 in Field 47249.
- The access rights that the invoking subject must have to the actual arguments in order to invoke Procedure 602 in Field 47251.

Field 47251 is itself an array which specifies the kinds of access that the invoking subject must have to the actual arguments it uses to invoke Procedure 602. Each formal argument for Procedure 602 has an Access Mode Array Entry (AMAE) 47255. The order of the AMAEs 47255 corresponds to the order of Procedure 602's formal arguments. The first formal argument has the first AMAE 47255, the second the second, and so forth. An AMAE 47253 is four bits long. There are two forms of AMAE 47253: Primitive Access Form 47255 and Extended Access Form 47257. In the former form, the leftmost bit is set to 0. The three remaining bits specify read, write, and execute access. If a bit is on, the subject performing the invocation must have that kind of primitive access to the object containing the data item used as an actual for the formal argument corresponding to that AMAE 47253. In the Extended Access Form 47257, the leftmost bit is set to 1 and the remaining bits are defined to represent extended access required for Procedure 602. The definition of these bits varies from Procedure 602 to Procedure 602.

5. Execution of Mediated Calls

Having described the portions of MAS Object 46703, SS Object 10336. and Procedure Object 608 which are involved in Calls, the discussion turns to the description of the Mediated Call Operation. First, there is presented an overview of the Mediated Call SIN and then the implementation of Mediated Calls in the present embodiment is discussed, beginning with a simple Mediated Call and continuing with Cross-Procedure Object Calls and Cross Domain Calls. The discussion closes with a description of software-to-microcode Calls.

a.a. Mediated Call SINs

While the exact form of a Mediated Call SIN is S-language specific, all Mediated Call SINs must contain four items of information:

- The SOP for the operation.
- A Name that evaluates to a pointer to the Procedure 602 to be invoked by the SIN.
- A literal (constant) specifying the number of actual arguments used in the invocation.
- A list of Names which evaluate to pointers to the actual arguments used in the invocation.

If Procedure 602 requires no arguments, the literal will be 0 and the list of Names representing the actual arguments will be empty.

In the present embodiment, Mediated Call and Return SINs are used whenever called Procedure 602 has a different PED 30303 from calling Procedure 602. In this case, the Call must save and recalculate macrostate other than FP and PC, and mediation by KOS Call microcode is required. The manner in which KOS Call microcode mediates the Call depends on whether the Call is a simple Mediated Call a Cross-procedure Object Call, or a Cross-Domain Call.

b.b. Simple Mediated Calls (Fig. 270, 468, 469, 470, 471, 472)

When the Mediated Call SIN is executed, S-interpreter microcode first evaluates the Name which represents the location of the called Procedure 602. The Name may evaluate to a pointer to a Gate 47205 or 4707 in another Procedure Object 608 or to a pointer to an Entry Descriptor 47227 in the present Procedure Object 608. When the Name has been evaluated, S-interpreter Call microcode invokes KOS Call microcode, using the evaluated Name as an argument. This microcode first fills in Macrostate Fields 10516, left empty until now, in the current invocation's SS Frame 47003. The microcode obtains the values for these fields from registers in FU 10120 where they are maintained while Virtual Processor 612 of Process 610 which is executing the Mediated Call is bound to JP 10114.

The next step to determine whether the pointer which KOS Call microcode received from S-interpreter Call microcode is a pointer to an External Procedure. To make this determination, KOS Call microcode compares the pointer's AON 41304 with that of Procedure Object 608 for Procedure 602 making the Call. If they are different, the Call is a Cross-Procedure Object Call, described below. In the case of the Simple Mediated Call, the format field indicates that the location is an Entry Descriptor 47227. KOS Call microcode continues by saving the location of Entry Descriptor 47227 and creating a new Mediated Frame 46947 on current MAS Object 46703 and a new Ordinary SS Frame 10510 on SS Object 10336 for called Procedure 602. As KOS Call microcode does so, it sets Fields 46917 and 46919 in Mediated Frame Header 10414 and

Fields 47109 and 47111 in Ordinary SS Frame Header 10514 to the values required by the addition of frames to MAS Object 46703 and SS Object 10336.

New Mediated Frame 46947 is now ready for Linkage Pointers 10416 to the actual arguments used in the Call, so KOS Call microcode returns to S-interpreter Call microcode, which parses the SIN to obtain the literal specifying the number of arguments and saves the literal value. S-interpreter Call microcode then parses each argument Name, evaluates it, and places the resulting value in Linkage Pointers Section 10416. When Linkage Pointers Section 10416 is complete, S-Interpreter Call Microcode calculates the new location of FP from the location of the top of Linkage Pointers Section 10416 and places a pointer for the location in the FU 10120 register reserved for FP. At this time, S-interpreter Call microcode also places the new location of the top of the stack in Stack Top Offset Field 46807.

S-interpreter Call microcode then invokes KOS Call microcode to place the value of the literal specifying the number of arguments in MAS Frame Field 46929, to calculate the new value of FHP 46702 and place it in the FU 10120 register reserved for that value, and finally to obtain the state necessary to execute called Procedure 602 from called Procedure 602's Entry Descriptor 47227 and PED 30303. As previously stated, S-interpreter Call microcode saved the location of Entry Descriptor 47227. Using this location, KOS Call Microcode obtains the size of the storage required for local data from Field 47233 and adds that amount of storage to the new MAS Frame 46709. Then KOS Call Microcode uses Field 47231 to locate PED 30303 for Procedure 602. PED 30303 contains the remainder of the necessary information about Procedure 602 and KOS Call microcode copies the location of PED 30303 into PED Pointer Field 46933 and then copies the values of K Field 30305. Last Name Field 30307, NTP Field 30311 and PBP Field 30315 into the relevant registers in FU 10120. KOS Call microcode next translates the pointer in SIP Field 30309 into a dialect number as explained in Chapter 3, and places it in register RDIAL 24212 of FU 10220 and thereupon derives SDP by resolving the pointer in SDPP Field 30313 and a pointer to SEB 46864 by resolving the pointer in SEPP Field 30316. Having performed these operations, KOS Call microcode returns to S-interpreter Call microcode, which finishes the Call by obtaining a new PC, that is, resetting registers in I-stream Reader 27001 in FU 10120 so that the next SIN to be fetched will be the first SIN of called procedure 602 S-interpreter Call microcode obtains the information required to change PC from Field 47229 in Entry Descriptor 47227 which contains the offset of the first SIN of called Procedure 602 from PBP.

In the present embodiment, some FU 10120 state produced by the Mediated Call SIN is retained on SS 504 throughout the duration of Procedure 602's invocation. The saved state allows Process 610 to reattempt the Mediated Call if the Call fails before the called Procedure 602 begins executing. When a Mediated Return SIN is executed, it resumes execution on the retained state from the CALL SINT. The Mediated Return is much simpler than the Call. Since all of the information required to resume execution of the invocation which performed the Call is contained in Macrostate 10516 in the calling invocation's SS Frame 47003, Return need only pop the called invocation's frames from current MAS Object 46703 and SS Object 10336, copy Macrostate 10516 47123 from the calling invocation's SS Frame 47003 into the proper FU 10120 registers, translate SIP Value 47141 into a dialect number, and resume executing the calling invocation. The pop operation involves nothing more than updating those pointers in MAS Object 46703 and SS Object 10336 which pointed to locations in the old topmost frame so that they now point to equivalent locations in the new topmost frame.

c.c. Invocations of Procedures 602 Requiring SEBs 46864 (Fig. 270, 468, 469, 470, 471, 472)

If a Procedure 602 requires a SEB 46864, this fact is indicated by Flag Field 47237 in Procedure 602's Entry Descriptor 47227. PED 30303 for such a Procedure 602 contains SEPP Field 47225, whose value is a non-resolvable pointer. The manner in which a SEB 46864 is created for Procedure 602 and SEPP field 47225 is translated into SEP, a pointer which contains the location of SEB 46864 and is saved as part of the invocation's macrostate on SS 10336, is similar to the manner in which a Static Data Block 46863 is created and the non-resolvable pointer contained in SDPP field 47225 is translated into SDP. The first time that a Procedure 602 requiring a SEB 46864 is invoked on a MAS Object 46703, a SEB 46864 is created for the Procedure 602 and an AATE 46857 is created which associates the nonresolvable pointer in SEPP field 47225 and the location of SEB 46864. That location is the value of SEP when the procedure is executing on MAS object 46703. On subsequent invocations of Procedure 602. AATE 46857 serves to translate the value in SEPP field 47225 into SEP.

d.d. Cross-Procedure Object Calls (Fig. 270, 468, 469, 470, 471, 472)

A Mediated Call which invokes an External Procedure 602 is called a Cross-Procedure Object Call. As previously mentioned. KOS Call microcode assumes that any time the Name representing the called Procedure 602 in a Mediated Call SIN resolves to the location of a Gate that the Call is to an External Procedure 602. As long as newly-called External procedure 602 has the same DOE as calling Procedure 602. Cross-Procedure Object Calls differ from the Simple Mediated Call only in the manner in which called Procedure 602's Entry Descriptor 47227 is located. Once KOS Call microcode has determined as described above that a Mediated Call is a Cross-Procedure Object Call it must next determine whether it is a Cross-Domain Call. To do so, KOS Call microcode compares the DOE Attribute of called Procedure 602's Procedure Object 608 with the domain component of the current subject. KOS Call microcode uses Procedure Object 608's AON 41304 to obtain Procedure Object 608's DOE from Field 41521 of its AOTE

41306 and it uses the ASN for the current subject, stored in an FU 10120 register, to obtain the current subject's domain component from AST 10914. If the DOE and the current subject's domain component differ, the Call is a Cross-domain Call, described below; otherwise, the Call locates the Gate 47205 or 47206 specified by the evaluated Name for called Procedure 602 in its Procedure Object 608. If the Gate is a Local Gate 47205, the Call uses Entry Descriptor Offset Field 47207 to locate Entry Descriptor 47227 belonging to Called Procedure 602 and then proceeds as described in the discussion of a Simple Mediated Call.

If the Gate is a Link Gate 47206, KOS Call microcode obtains the pointer corresponding to Link Gate 47206 from Binder Area 47245 and resolves it to obtain a pointer to another Gate 47205 or 47206, which KOS Call microcode uses to repeat the External Procedure 602 call described above. The repetitions continue until the newly-located gate is a Local Gate 47205, whereupon Call proceeds as described for Simple Mediated Calls.

e.e. Cross-domain Calls (Fig. 270, 408, 418, 468, 469, 470, 471, 472)

If a called Procedure 602's Procedure Object 608 has a DOE attribute differing from that of calling Procedure 602's Procedure Object 608, the Call is a Cross-domain Call. The means by which KOS Call microcode determines that a Mediated Call is a Cross-Domain Call have previously been described; If the Call is a Cross-Domain Call, KOS Call microcode must inactivate MAS Object 46703 for the domain from which the Call is made, perform trojan horse argument checks, switch subjects, place a Cross-domain Frame 47039 on SS object 10336, and locate and activate MAS Object 46703 for the new domain before it can make a Mediated Frame 46947 on new MAS Object 46703 and continue as described in the discussion of a Simple Mediated Call.

Cross-domain Call microcode first inactivates the current MAS Object 46703 by setting Domain Active Flag 46804 to FALSE. The next step is the trojan horse argument checks. In order to perform trojan horse argument checks, Cross-domain Call must have pointers to the actual arguments used in the cross-domain invocation. Consequently, Cross-domain Call first continues like a non-cross-domain Call: it creates a Mediated Frame Header 10414 on old MAS Object 46703 and returns to S-interpreter microcode, which evaluates the Names of the actual arguments, and places the pointers in Linkage Pointers 10416 above Mediated Frame Header 10414. However, the macrostate for the invocation performing the call was placed on SS Object 10336 before Mediated Frame Header 10414 and Linkage Pointers 10416 were placed on old MAS Object 46703. Consequently, when calling Procedure 602 resumes execution after a Return, it will resume on MAS Frame 46709 preceding the one built by Cross-domain Call microcode.

Once the pointers to the actual arguments are available, Cross-domain Call Microcode performs the trojan horse check. As described in the discussion of Procedure Object 608 and illustrated in Figure 472, the information required to perform the check is contained in AIA 10352. Each Local Gate 47205 in Procedure Object 608 has an AIAE 47245, each formal argument in Local Gate 47205's procedure has an entry in AIAE 47245's AMA 47251, and the formal argument's AMAE 47253 indicates what kind of access to the formal argument's actual argument is required in called Procedure 602.

Field AIA OFF 47201 contains the location of AIA 10352 in Procedure Object 608, and using this information and Local Gate 47205's offset in Procedure Object 608, Cross-domain Call microcode locates AIAE 47245 for Local Gate 47205. The first two fields in AIAE 47245 contain the minimum number of arguments in the invocation and the maximum number of arguments. Cross-domain Call microcode checks whether the number of actual arguments falls between these values. If it does, Cross-domain Call microcode begins checking the access allowed individual arguments. For each argument pointer. Cross-domain Call microcode calls LAR microcode to obtain the current AON 41304 for the pointer's UID and uses AON 41304 and the ASN for Process 610's current subject (i.e., the caller's subject) to locate an entry in either APAM 10918 or ANPAT 10920, depending on whether the argument's AIAE specifies primitive access (47255) or extended access (47257) respectively. If the information from APAM 10918 or ANPAT 10920 confirms that Process 610's current subject has the right to access the argument in the manner required in called Procedure 602, the Trojan Horse microcode goes on to the next argument. If the current subject has the required access to all arguments, the trojan horse check succeeds and the Cross-domain Call continues, Otherwise, it fails and Cross-domain Call performs a microcode-to-software Call as explained below.

Next, Cross-domain Call microcode places Cross domain State 10513 on SS Object 10336. As explained in the discussion of SS object 10336, Cross-domain State 10513 contains the information required to return to the caller's frame on former MAS Object 46703. Having done this, Cross-domain Call microcode changes subjects. Using the current subject's ASN, Cross-Domain Call microcode obtains the current subject from AST 10914 replaces the subject's domain component with DOE Attribute 41225 for called Procedure 602's Procedure Object 608 and uses AST 10914 to translate the new subject thus obtained into a new ASN. That ASN then is placed in the appropriate FU 10120 register.

After the subject has been changed, Cross-domain Call microcode uses Domain Table 41801 to translate the DOE of called Procedure 602 into a domain number. Cross-domain Call microcode then uses the domain number as an index into Array of MAS AONs 46211 in VPSB 614 for Virtual Processor 612 belonging to Process 610 making the cross-domain call. The entry corresponding to the domain number contains AON 41304 of MAS Object 46703 for that domain.

Having located the proper MAS Object 46703, Cross-domain Call microcode uses STO field 46807 in MAS Header 10410 belonging to the new domains MAS Object 46703 to locate the top of the last MAS

Frame 46709. It then saves the value of FHP 46702 used in the preceding invocation in a FU 10120 register, adds a Mediated Frame Header 10414 to the top of MAS Object 46703, and calculates a new FHP 46702 which points to new Mediated Frame Header 10414. KOS Cross-Domain Call microcode then places the old value of FHP 46702 in FHP Value Field 47151 of SS Object 10336 and the old value of STO 46704 (pointing to the top of the last complete MAS Frame 46709 on previous MAS Object 46703) in Field 47153 of Cross-Domain State 10513 and fills in Mediated Frame Header 10414 fields as follows: Result of Cross-domain Call Field 46903 is set to TRUE. Previous Frame Offset Field 46917 is set to 0, and Dynamic Back Pointer Field 46931 is set to the saved value of FHP 46702. Dynamic Back Pointer Field 46931 thus points to the header of the topmost Mediated Frame 46947 on the previous MAS Object 46703. The values of the remaining fields are copied from Mediated Frame Header 10414 which Cross-Domain Call created on previous MAS Object 46703.

Cross-domain Call microcode next copies the argument pointers for the formal arguments from the top of previous MAS Object 46703 to new Mediated Frame 46947 and calculates FP. Cross-domain Call Microcode finishes by returning to S-interpreter Call microcode, which completes the Call as described for Simple Mediated Calls.

Except for the work involved in transferring to a new MAS Object 46703, Cross-domain Return is like other Returns from Mediated Calls. Old FHP 46701 from Field 47151 of Cross-Domain State 10513 and old STO 46704, from Field 47153 of Cross-domain State are placed in FU 10120 registers. Then the frames belonging to the invocation that is ending are popped off of SS Object 10336 and off of MAS Object 46703 belonging to the domain of called Procedure 602 and MAS Object 46703 is inactivated by setting Domain-Active Flag 46804 to FALSE. Then KOS Cross-domain Return microcode uses old FHP 46701 and old STO 46704 to locate MAS Object 46703 being returned to and the topmost Mediated Frame 46947 on that MAS Object 46703. MAS Object 46703 being returned to is activated, and finally, the contents of Macrostate 10516 belonging to the invocation being returned to are placed in the appropriate registers of FU 10120 and execution of the invocation resumes.

f.f. Failed Cross-Doamin Calls (Fig. 270, 468, 469, 470, 471, 472)

A Cross-Domain Call as described above may fail at several points between the time that the calling invocation begins the call and called Procedure 602 begins executing. On failure, Cross-Domain Call microcode performs a microcode-to-software Call. KOS Procedures 602 invoked by this Call may remedy the reason for the Cross Domain Call's failure and reattempt the Cross-domain Call. This is possible because the implementation of Cross Domain Call in CS 10110 saves sufficient FU 10120 state to allow Process 610 executing the Cross-Domain Call to return to the invocation and the Mediated Call SIN from which the Cross-Domain Call began. On failure, the invocation's MAS Frame 46709 may be located from the values of STO Field 47153 and FHP Field 47151 in Cross-Domain State 10513, and the Mediated Call SIN may be located by using information saved in FU 10120 state.

6. Neighborhood Calls (Fig. 468, 479, 472)

As previously mentioned, Procedures 602 called via Neighborhood Calls must have the same PED 30303 as calling Procedure 602. The only macrostate values which are not part of PED 30303 are PC and FP; consequently Neighborhood Call need only save PC and FP of the invocation performing the call and calculate these values for the new invocation. In addition, Neighborhood Call saves STO 46704 in order to make it easier to locate the top of the previous invocation's Neighborhood Frame 46947. Neighborhood Return simply restores the saved values. Since the macrostate values copied from or obtained via PED 30303 do not change during the sequence of invocations, and therefore need not be saved on SS Object 10336. Neighborhood Calls do not have SS Frames 47003.

The invention may be embodied in yet other specific forms without departing from the spirit or essential characteristics thereof. Thus, the present embodiments are to be considered in all respects as illustrative and not restrictive, the scope of the invention being indicated by the appended claims rather than by the foregoing description.

Claims

1. A digital computer system (CS 101) including processor means (JP 114) for performing operations upon operands, memory means (MEM 112) for storing said operands and procedures, said procedures including instructions for controlling said operations and names referring to certain of said operands to be operated upon, ALU means (2034, 2074) for performing said operations, bus means (MOD 140, JPB 142) for conducting said instructions, names and operands between said memory means and said processor means, and I/O means (IOS 116) for conducting at least said operands between said memory means and devices external to said digital computer system, characterised in that said processor means (JP 114) comprises means for addressing said operands, including name table means (10350) for storing name table entries, each name table entry corresponding to one of said names included in each one of said procedures and each name table entry comprising first data from which may be determined an address of a location in said memory means of the operand referred to by one of said names and second data identifying a format of that operand, and translation means (NAME TRANS UNIT 27015) connected to said

bus means and responsive to said name table entries for providing outputs to said memory means representing said addresses, and further characterised in that said instructions are intermediate level S-language instructions from a plurality of sets of such instructions, each set corresponding to a particular higher level user programming language, and further characterised by receiving means (INSTB 20262) connected to said bus means for receiving said instructions from said memory means, and microcode control means (10240, 27003, 27013) connected between said receiving means and said ALU means for providing sequences of microinstructions for controlling said ALU means, said sequences being selected from a plurality of sequences of microinstructions corresponding to said S-language instructions respectively.

2. A digital computer system according to claim 1, characterised in that the S-language instructions have a uniform, fixed format.

3. A digital computer system according to claim 1 or 2, characterised in that the names are of uniform length and format.

4. A digital computer system according to any of claims 1 to 3, characterised in that each procedure further includes a name table pointer (NTP 30311) representing a base location in said memory means (MEM 112), and said first data of each name table entry contains information from which may be determined an address offset of a memory location relative to the base location, and in that said translation means (NAME TRANS UNIT 27015) further comprises base register means (NCR, MCR 10366) connected to said bus means for receiving and storing said name table pointer of the procedure currently controlling the operations performed by said ALU means.

5. A digital computer system according to any of claims 1 to 4, characterised by name cache means (10226) connected to outputs of said translation means (NAME TRANS UNIT 27015) and having outputs to said memory means (MEM 112) for storing said addresses, and further connected to said receiving means (INSTB 20262) and responsive to said names to provide name cache outputs to said memory means representing said addresses of certain operands for which said name cache means has stored said addresses.

6. A digital computer system according to any of claims 1 to 5, characterised in that each of said S-Language instructions is a member of an S-Language dialect of a plurality of S-Language dialects, and in that said receiving means (INSTB 20262) further comprises dialect code means (RDIAL 24212) for storing a dialect code specifying the dialect of which the received S-Language instructions are members, and in that said sequences of microinstructions include a set of sequences of microinstructions, corresponding to each said S-Language dialect, each set of sequences of microinstructions including at least one sequence of microinstructions corresponding to each S-Language instruction in a corresponding S-Language dialect, and in that said microcode control means (10240, 27003, 27013) is responsive to the dialect code and to each received S-Language instruction to provide to said ALU means (2034, 2074) a sequence of microinstructions corresponding to that S-Language instruction.

7. A digital computer system according to claim 1 or 2, characterised in that each procedure includes a dialect code denoting an S-Language dialect of which the S-Language instructions of the procedure are members, and in that said microcode control means (10240, 27003, 27013) further comprises control store means (SITT 11012) for storing said sequences of microinstructions for controlling said ALU means (2034, 2074), and dispatch table means (SIDT 11010) for storing addresses corresponding to locations in said control store means of each sequence of microinstructions, and in that said dispatch table means is responsive to said dialect code and to each instruction to provide to said control store means each address corresponding to said at least one microinstruction sequence corresponding to each said instruction, and said control store means is responsive to each address to provide to said ALU means said sequence of microinstructions corresponding to each instruction.

8. A digital computer system according to claim 1, 6 or 7, characterised in that said microcode control means (10240, 27003, 27013) comprises writable control store means (11012) connected to said bus means for storing said sequences of microinstructions, and control store addressing means (SITTNAS 20286) responsive to each S-Language instruction and to operation of said processor means for generating control store read addresses and write addresses (CSADR 20204), and in that said writable control store means is responsive to said read addresses to provide said sequences of microinstructions to said ALU means (2034, 2074) and is responsive to said write addresses to store said sequences of microinstructions.

9. A digital computer system according to claim 7, characterised in that said control store means (SITT 11012) comprises writable control store means connected to said bus means for storing said sequences of microinstructions, and in that said dispatch table means comprises write address means responsive to operation of said processor means for generating write addresses, and in that said writable control store means is responsive to said write addresses for storing said sequences of microinstructions.

Patentansprüche

1. Digitales Datenverarbeitungssystem (CS 101), enthaltend: Prozessormittel (MEM 114) zur Durchführung von Operationen an Operanden, Speichermittel (MEM 112) zum Speichern der Operanden und von Prozeduren, die Befehle zur Steuerung der Operationen und Namen enthalten, die auf gewisse der Operanden Bezug nehmen, an denen Operationen durchgeführt werden sollen, eein Rechenwerk (2034,

2074) zur Durchführung der Operationen, Bus-Mittel (MOD 140, JPE 118) für den Verkehr der Befehle, Namen und Operanden zwischen den Speichermitteln und den Prozessormitteln, und Eingabe/Ausgabe-Mittel (IOS 116) für den Verkehr wenigstens der Operanden zwischen den Speichermitteln und Geräten außerhalb des digitalen Datenverarbeitungssystems, gekennzeichnet durch Prozessormittel (JP 114), die Mittel zur Adressierung der Operanden einschließlich Namenstabellenmittel (10350) zur Speicherung von Namenstabellen-Einsprungpunkten enthalten, wobei jeder Namenstabellen-Einsprungpunkt einem der Namen entspricht, die in jeder der Prozeduren enthalten sind, und erste Daten, aus denen eine Adresse eines Platzes derjenigen Operanden in den Speichermitteln bestimmt werden kann, auf die durch einen der Namen Bezug genommen wird, und zweite Daten enthalten die ein Format dieses Operanden identifizieren, und durch Übersetzungsmittel (NAME TRANS UNIT 27015), die mit den Bus-Mitteln verbunden sind und auf die Namenstabellen-Einsprungpunkte unter Bereitstellung von diese Adressen repräsentierenden Ausgaben für die Speichermittel ansprechen, ferner dadurch gekennzeichnet, daß die Befehle mittlere S-Sprache-Befehle von einer Vielzahl von Sätzen solcher Befehle sind, von denen jeder Satz einer besonderen höheren Benutzerprogrammiersprache entspricht, und ferner gekennzeichnet durch ein mit den Bus-Mitteln verbundenes Empfangsmittel (INSTB 20262) zum Empfang der Befehle von den Speichermitteln, und durch mit dem Empfangsmittel und dem Rechenwerk verbundene Mikrocode-Steuermittel (10240, 27003, 27013) zur Bereitstellung von Mikrobefehlssequenzen zur Steuerung des Rechenwerks, wobei diese Sequenzen aus einer Vielzahl von Mikrobefehlssequenzen ausgewählt sind, die den jeweiligen S-Sprache-Befehlen entsprechen.

2. Digitales Datenverarbeitungssystem nach Anspruch 1, dadurch gekennzeichnet, daß die S-Sprache-Befehle ein gleichförmiges, festes Format haben.

3. Digitales Datenverarbeitungssystem nach Anspruch 1 oder 2, dadurch gekennzeichnet, daß die Namen eine gleichförmige Länge und ein gleichförmiges Format haben.

4. Digitales Datenverarbeitungssystem nach einem der Ansprüche 1 bis 3, dadurch gekennzeichnet, daß jede Prozedur weiter einen Namenstabellenzeiger (NTP 30311) enthält, der einen Basisplatz in den Speichermitteln (MEM 112) repräsentiert, daß die ersten Daten jedes Namenstabellen-Einsprungpunktes Informationen enthalten, aus denen die Adresse eines vom Basisspeicherplatz versetzten Speicherplatzes bestimmt werden können, und daß die Übersetzungsmittel (NAME TRANS UNIT 27015) weiter Basisregistermittel (NCR, MCR 10366) enthalten, die mit den Bus-Mitteln verbunden sind, um den Namenstabellenzeiger derjenigen Prozedur zu empfangen und zu speichern, die gerade die vom Rechenwerk durchgeführten Operationen steuert.

5. Digitales Datenverarbeitungssystem nach einem der Ansprüche 1 bis 4, gekennzeichnet durch Namens-Cache-Speichermittel (10226), die mit den Ausgängen der Übersetzungsmittel (NAME TRANS UNIT 27015) verbunden sind und zu den Speichermitteln (MEM 112) führend Ausgänge zum Speichern der Adressen haben, und die weiter mit dem Empfangsmittel (INSTB 20262) verbunden sind und auf die Namen unter Bereitstellung von Namens-Cache-Ausgaben für die Speichermittel ansprechen, die die Adressen von gewissen Operanden repräsentieren, für die die Namens-Cache-Speichermittel die Adressen gespeichert haben.

6. Digitales Datenverarbeitungssystem nach einem der Ansprüche 1 bis 5, dadurch gekennzeichnet, daß jeder der S-Sprache-Befehle ein Mitglied eines S-Sprache-Dialekts einer Vielzahl von S-Sprache-Dialekten ist, daß das Empfangsmittel (INSTB 20262) weiter ein Dialekt-Code-Mittel (RDIAL 24212) zur Speicherung eines Dialekt-Codes enthält, der den Dialekt bestimmt, von dem die empfangenen S-Sprache-Befehle Mitglieder sind, daß die Mikrobefehlssequenzen einen Satz von Mikrobefehlssequenzen entsprechend jedem S-Sprache-Dialekt enthalten, wobei jede Mikrobefehlssequenz wenigstens eine jedem S-Sprache-Befehl in einem entsprechenden S-Sprache-Dialekt entsprechenden Mikrobefehlssequenz enthält, und daß die Mikrocode-Steuermittel (10240, 27003, 27013) auf den Dialekt-Code und jeden empfangenen S-Sprache-Befehl unter Bereitstellung einer diesem S-Sprache-Befehl entsprechenden Mikrobefehlssequenz für das Rechenwerk ansprechen.

7. Digitales Datenverarbeitungssystem nach Anspruch 1 oder 2, dadurch gekennzeichnet, daß jede Prozedur einen Dialektcode enthält, der einen S-Sprache-Dialekt bezeichnet, von dem die S-Sprache Befehle der Prozedur Mitglieder sind, daß die Mikrocode-Steuermittel (10240, 27003, 27013) ferner Steuerspeichermittel (SITT 11012) zur Speicherung der Mikrobefehlssequenzen für die Steuerung des Rechenwerks (2034, 2074) und Verteilertabellenmittel (SIDT 11010) zur Speicherung von Adressen enthalten, die Plätzen jeder Mikrobefehlssequenz in den Steuerspeichermitteln entsprechen, und daß die Verteilertabellenmittel auf den Dialektcode und jeden Befehl unter Bereitstellung jeder Adresse, die der wenigstens einen, zu jedem Befehl gehörenden Mikrobefehlssequenz entspricht, für die Steuerspeichermittel ansprechen, während die Steuerspeichermittel auf jede Adresse unter Bereitstellung der jedem Befehl entsprechenden Mikrobefehlssequenz für das Rechenwerk ansprechen.

8. Digitales Datenverarbeitungssystem nach Anspruch 1, 6 oder 7, dadurch gekennzeichnet, daß die Mikrocode-Steuermittel (10240, 27003, 27013) ein mit den Bus-Mitteln verbundenes Schreibsteuerspeichermittel (11012) zur Speicherung der Mikrobefehlssequenzen und Steuerspeicheradressiermittel (SITTNAS 20286) enthalten, die auf jeden S-Sprache-Befehl und auf Operationen des Prozessormittels unter Erzeugung von Steuerspeicherlese- und -schreibadressen (CSADR 20204) ansprechen, und daß die Schreibsteuerspeichermittel auf die Leseadressen unter Bereitstellung der Mikrobefehlssequenzen für das Rechenwerk und auf die Schreibadressen unter Speicherung dieser Mikrobefehlssequenzen ansprechen.

9. Digitales Datenverarbeitungssystem nach Anspruch 7, dadurch gekennzeichnet, daß die Steuerspeichermittel (SITT 11012) mit den Bus-Mitteln verbundene Schreibsteuerspeichermittel zur Speicherung der Mikrobefehlssequenzen enthalten, daß die Verteilertabellenmittel Schreibadressenmittel enthalten, die auf Operationen des Prozessormittels unter Erzeugung von Schreibadressen ansprechen, und daß die Schreibsteuerspeichermittel auf die Schreibadressen unter Speicherung der Mikrobefehlssequenzen ansprechen.

Revendications

1. Un système d'ordinateur numérique (CS 101), comprenant un processeur (JP 114) pour effectuer des opérations sur des opérandes, une mémoire (MEM 112) pour mémoriser lesdits opérandes et des procédures, lesdites procédures contenant des instructions pour commander lesdites opérations et des désignations se rapportant à certains desdits opérandes pour les traiter, une unité arithmétique et logique ALU (2034, 2074) pour effectuer lesdites opérations, des bus (MOD 140, JPB 142) pour transmettre lesdites instructions, lesdites désignations et lesdits opérandes entre ladite mémoire et ledit processeur, et des moyens d'entrée/sortie I/O (IOS 116) pour transmettre au moins lesdits opérandes entre ladite mémoire et des dispositifs extérieurs audit système d'ordinateur numérique, caractérisé en ce que ledit processeur (JP 114) comprend des moyens pour l'adressage desdits opérandes, comportant une table de désignations (10350) pour mémoriser des entrées de table de désignations, chaque entrée de table de désignations correspondant à une desdites désignations incluses dans chacune desdites procédures et chaque entrée de table de désignations comprenant une première donnée à partir de laquelle peut être déterminée une adresse d'un emplacement de ladite mémoire contenant l'opérande auquel se reflète l'une desdites désignations et une seconde donnée identifiant un format de cet opérande, et des moyens de transcodage (NAME TRANS UNIT 27015) reliés auxdits bus et réagissant auxdites entrées de tables de désignations de façon à transmettre à ladite mémoire des signaux de sortie représentant lesdites adresses, et en outre caractérisé en ce que lesdites instructions sont des instructions en langage-S de niveau intermédiaire provenant d'une pluralité d'ensembles de telles instructions, chaque ensemble correspondant à un langage de programmation par utilisateur de niveau supérieur particulier, et en outre caractérisé en ce que des moyens de réception (INSTB 20262) sont reliés auxdits bus pour recevoir lesdites instructions à partir de ladite mémoire, et des moyens de commande de microcode (10240, 27003, 27013) connectés entre lesdits moyens de réception et ladite ALU pour fournir des séquences de microinstructions servant à commander ladite ALU, les dites séquences étant sélectionnées parmi une pluralité de séquences de micro-instructions correspondant respectivement auxdites instructions en langage-S.

2. Un système d'ordinateur numérique selon la revendication 1, caractérisé en ce que les instructions en langage-S ont un format fixe et uniforme.

3. Un système d'ordinateur numérique selon une des revendications 1 ou 2, caractérisé en ce que les désignations ont une longueur et un format uniformes.

4. Un système d'ordinateur numérique selon une quelconque des revendications 1 à 3, caractérisé en ce que chaque procédure comprend en outre un pointeur de table de désignations (NTP 30311) représentant un emplacement de base dans ladite mémoire (MEM 112) et ladite première donnée de chaque entrée de la table de désignations contient une information à partir de laquelle peut être déterminé un décalage d'adresse d'un emplacement de mémoire par rapport à l'emplacement de base, et en ce que lesdits moyens de transcodage (NAME TRANS UNIT 27015) comprennent en outre un moyen formant registre de base (NCR, MCR 10366), qui est relié auxdits bus de façon à recevoir et mémoriser ledit pointeur de table de désignations dans la procédure qui est en train de commander les opérations effectuées par ladite ALU.

5. Un système d'ordinateur numérique selon une quelconque des revendications 1 à 4, caractérisé par un moyen formant antémémoire de désignations (10226), relié aux sorties desdits moyens de transcodage (NAME TRANS UNIT 27015) et comportant des sorties reliées à ladite mémoire (MEM 112) pour mémoriser lesdites adresses, et en outre relié auxdits moyens de réception (INSTB 20262) et réagissant auxdites désignations pour fournir à ladite mémoire des sorties de l'antémémoire de désignations représentant lesdites adresses de certains opérandes pour lesquels ladite antémémoire de désignations a mémorisé lesdites adresses.

6. Un système d'ordinateur numérique selon une quelconque des revendications 1 à 5, caractérisé en ce que chacune desdites instructions en langage-S est un élément d'un dialecte en langage-S faisant partie d'une pluralité de dialectes en langage-S et en ce que lesdits moyens de réception (INSTB 20262) comprennent en outre un moyen de codage de dialecte (RDIAL 24212) pour mémoriser un code de dialecte spécifiant le dialecte dont les instructions en langage-s reçues sont des éléments, et en ce que lesdites séquences de micro-instructions contiennent un ensemble de séquences de micro-instructions correspondant à chacun desdits dialectes en langage-S, chaque ensemble de séquences de micro-instructions comprenant au moins une séquence de micro-instructions correspondant à chaque instruction en langage-S dans un dialecte en langage-S correspondant, et en ce que lesdits moyens de commande de microcode (10240, 27003, 27013) réagissent audit code de dialecte et à chaque instruction en langage-S reçue pour fournir à ladite ALU (2034, 2074) une séquence de micro-instructions correspondant à cette instruction en langage-S.

7. Un système d'ordinateur numérique selon une des revendications 1 et 2, caractérisé en ce que chaque procédure comprend un code de dialecte définissant un dialecte en langage-S dont les instructions en langage-S de la procédure sont des éléments et en ce que lesdits moyens de commande de microcode (1020, 27003, 27013 comprennent en outre une mémoire de commande (SITT 11012) pour mémoriser lesdites séquences de micro-instructions pour commander ladite ALU (2034, 2074), et un moyen à table de distribution (SIDT 11010) pour mémoriser des adresses correspondant aux emplacements de chaque séquence de micro-instructions dans ladite mémoire de commande, et en ce que ledit moyen à table de distribution réagit audit code de dialecte et à chaque instruction pour fournir à ladite mémoire de commande chaque adresse correspondant à ladite séquence de micro-instructions au moins prévue correspondant à chacune desdites instructions, et ladite mémoire de commande réagit à chaque adresse pour fournir à ladite ALU ladite séquence de micro-instructions correspondant à chaque instruction.

8. Un système à ordinateur numérique selon une des revendications 1, 6 et 7, caractérisé en ce que lesdits moyens de commande de microcode (10240, 27003, 27013) comprennent une mémoire de commande inscriptible (11012) reliée auxdits bus pour mémoriser lesdites séquences de micro-instructions et un moyen d'adressage de mémoire de commande (SITTNAS 20286) réagissant à chaque instruction en langage-S et au fonctionnement dudit processeur pour produire des adresses de lecture et des adresses d'écriture dans la mémoire de commande (CSADR 20204) et en ce que ladite mémoire de commande inscriptible réagit auxdites adresses de lecture pour fournir lesdites séquences de micro-instructions à ladite ALU (2034, 2074) et réagit auxdites adresses d'écriture pour mémoriser lesdites séquences de micro-instructions.

9. Un système d'ordinateur numérique selon la revendication 7, caractérisé en ce que ladite mémoire de commande (SITT 11012) comprend une mémoire de commande inscriptible qui est reliée auxdits bus de mémoriser lesdites séquences de micro-instructions et en ce que ledit moyen à table de distribution comprend un moyen d'adressage d'écriture réagissant au fonctionnement dudit processeur pour produire des adresses d'écriture, et en ce que la mémoire de commande inscriptible réagit auxdites adresses d'écriture pour mémoriser lesdites séquences de micro-instructions.

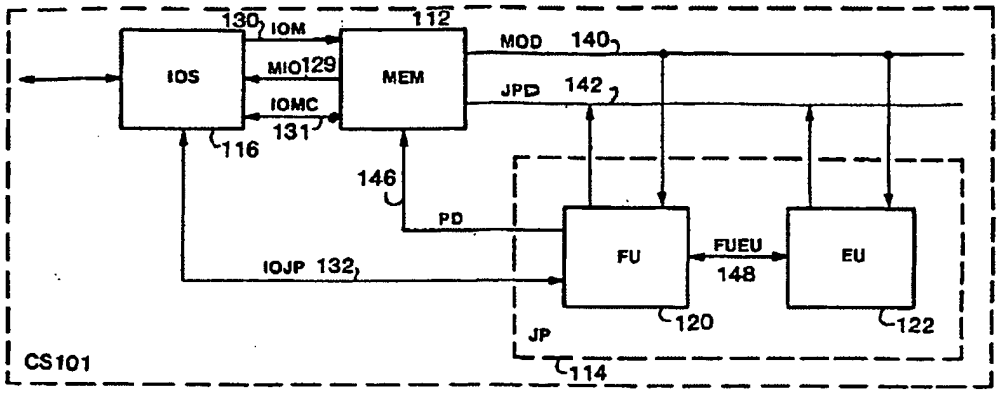


FIG 1

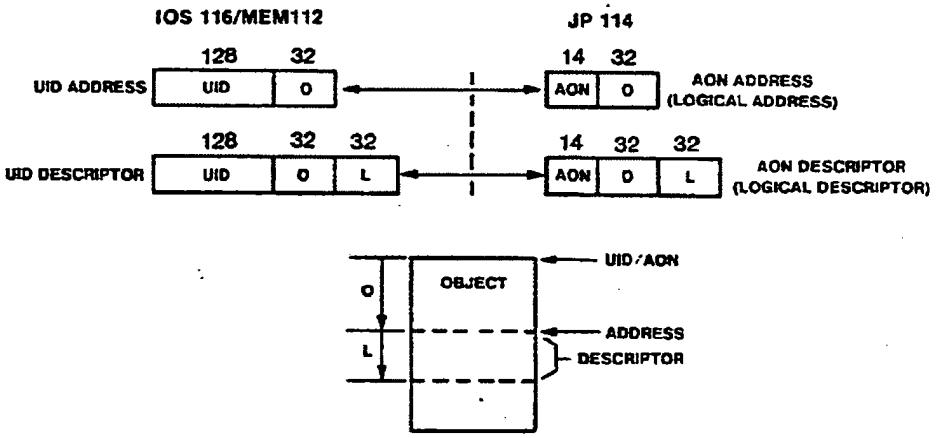


FIG 2

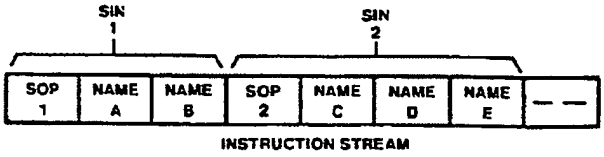


FIG 3

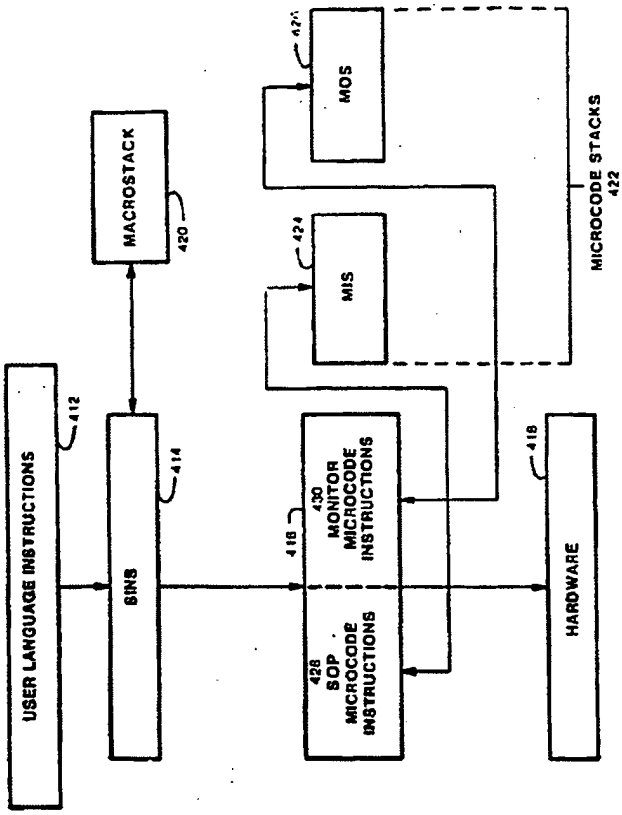


FIG 4A

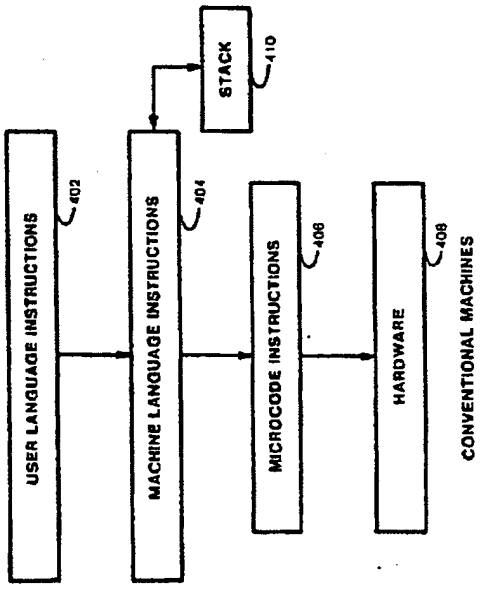


FIG 4

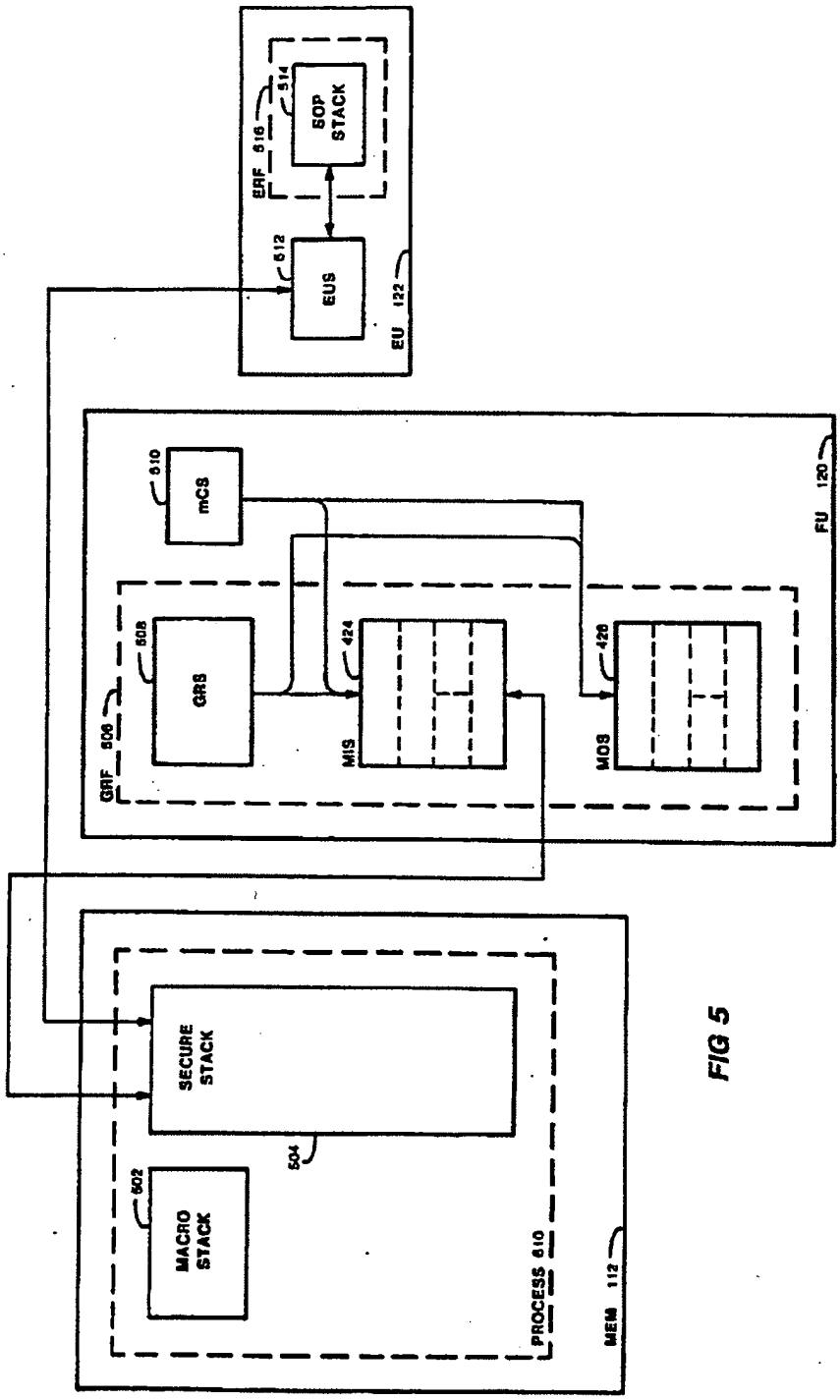


FIG 5

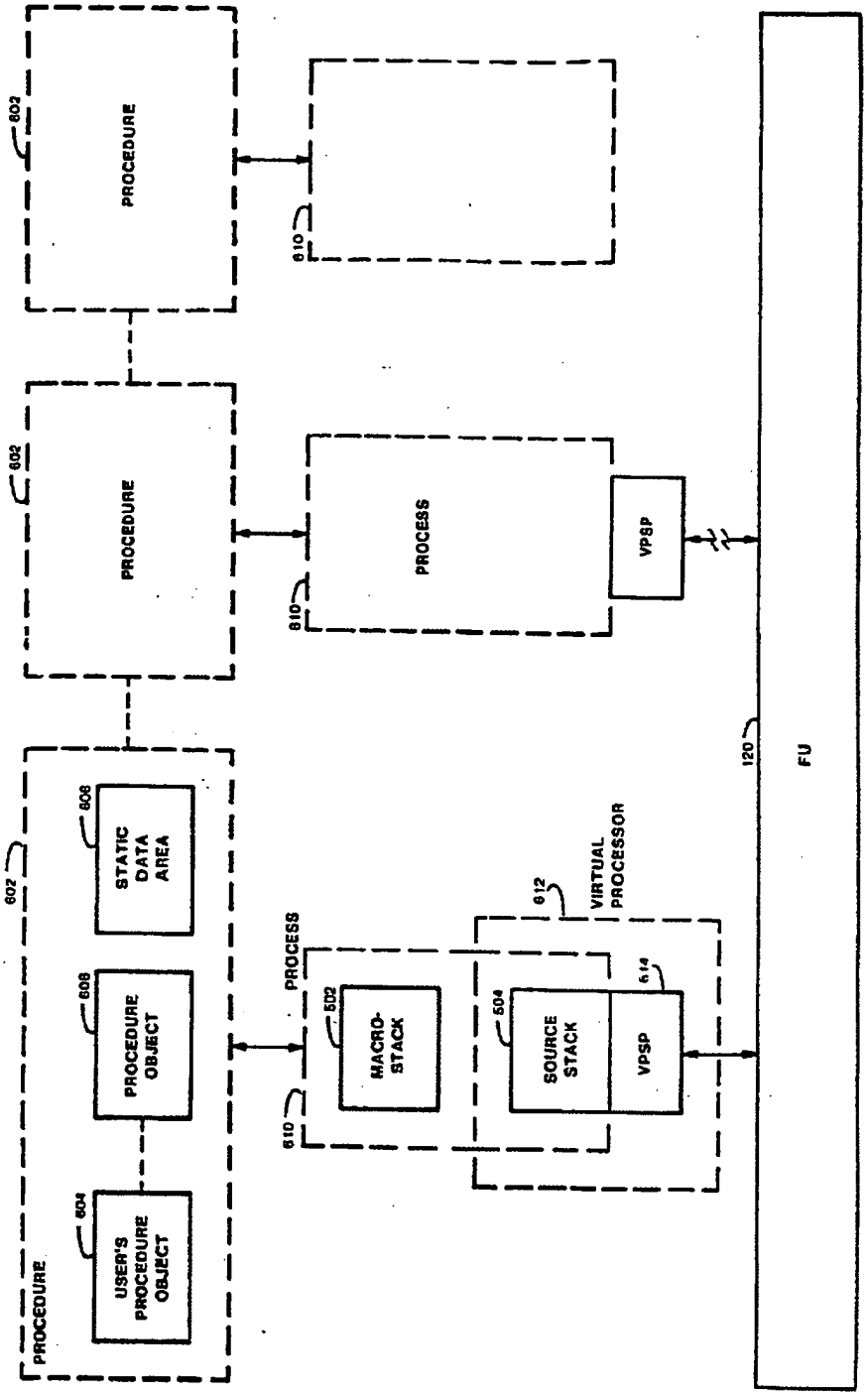


FIG 6

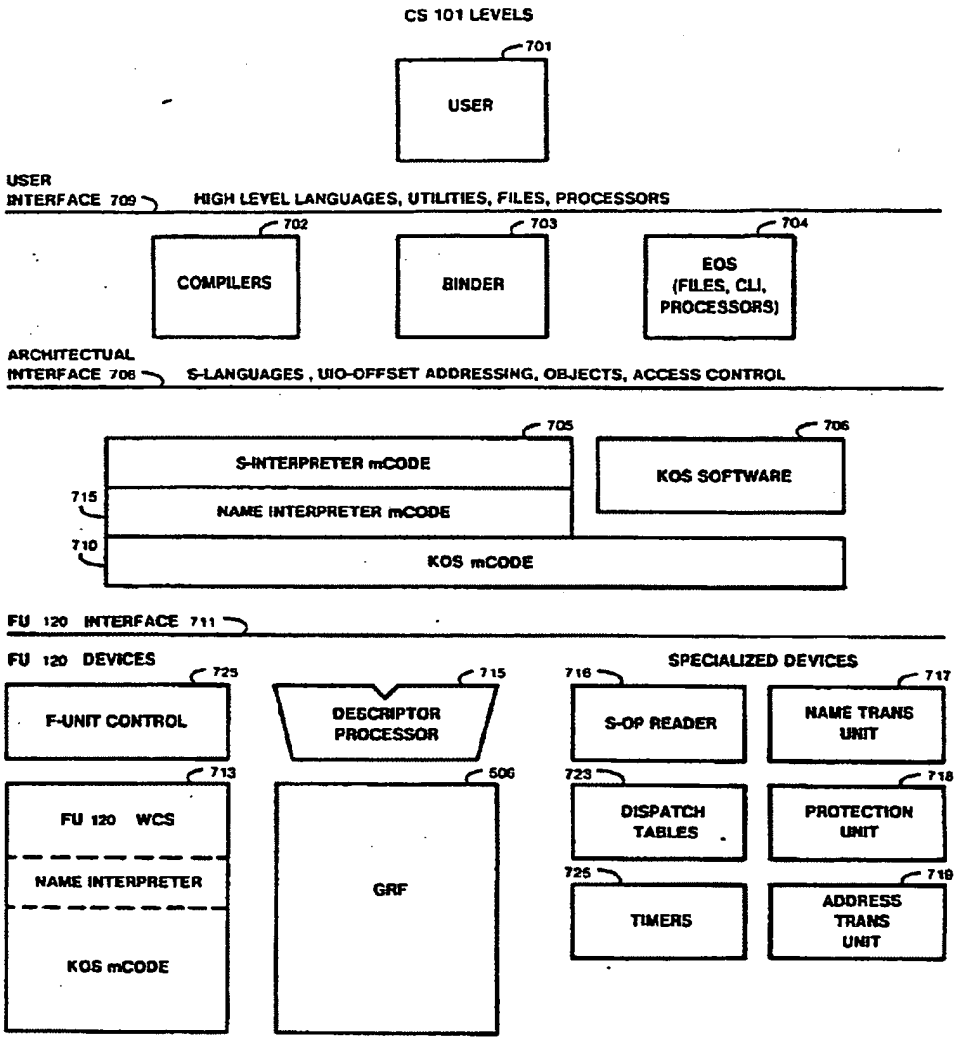


FIG 7

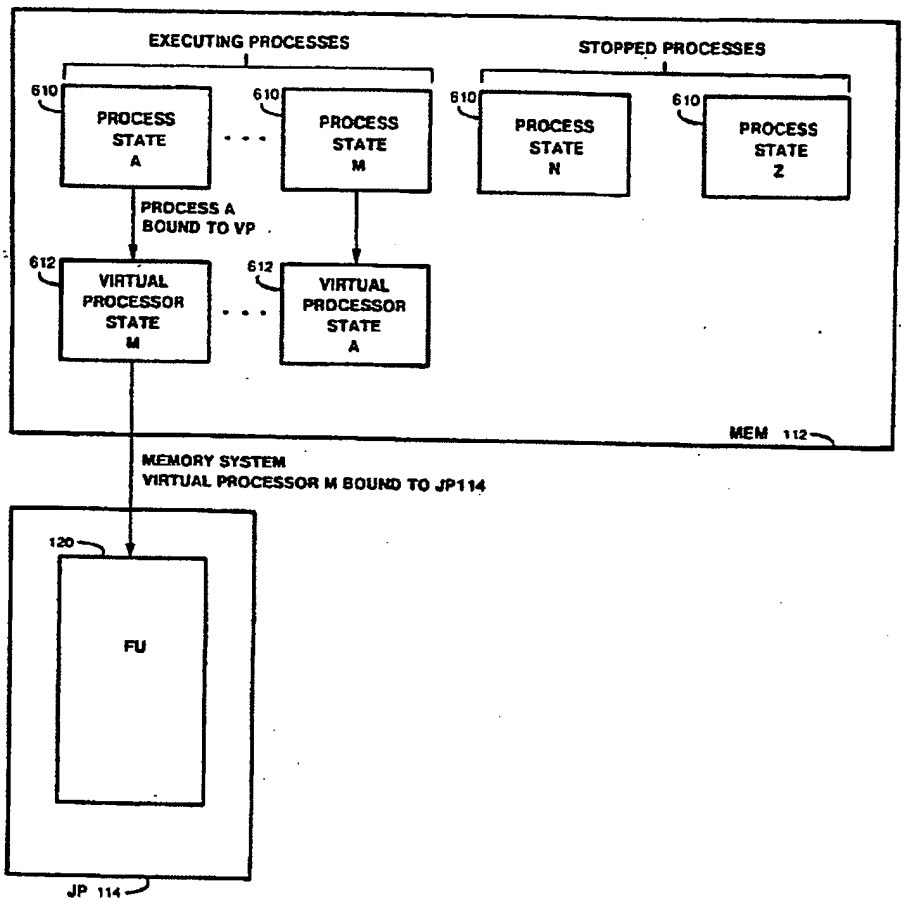


FIG 8

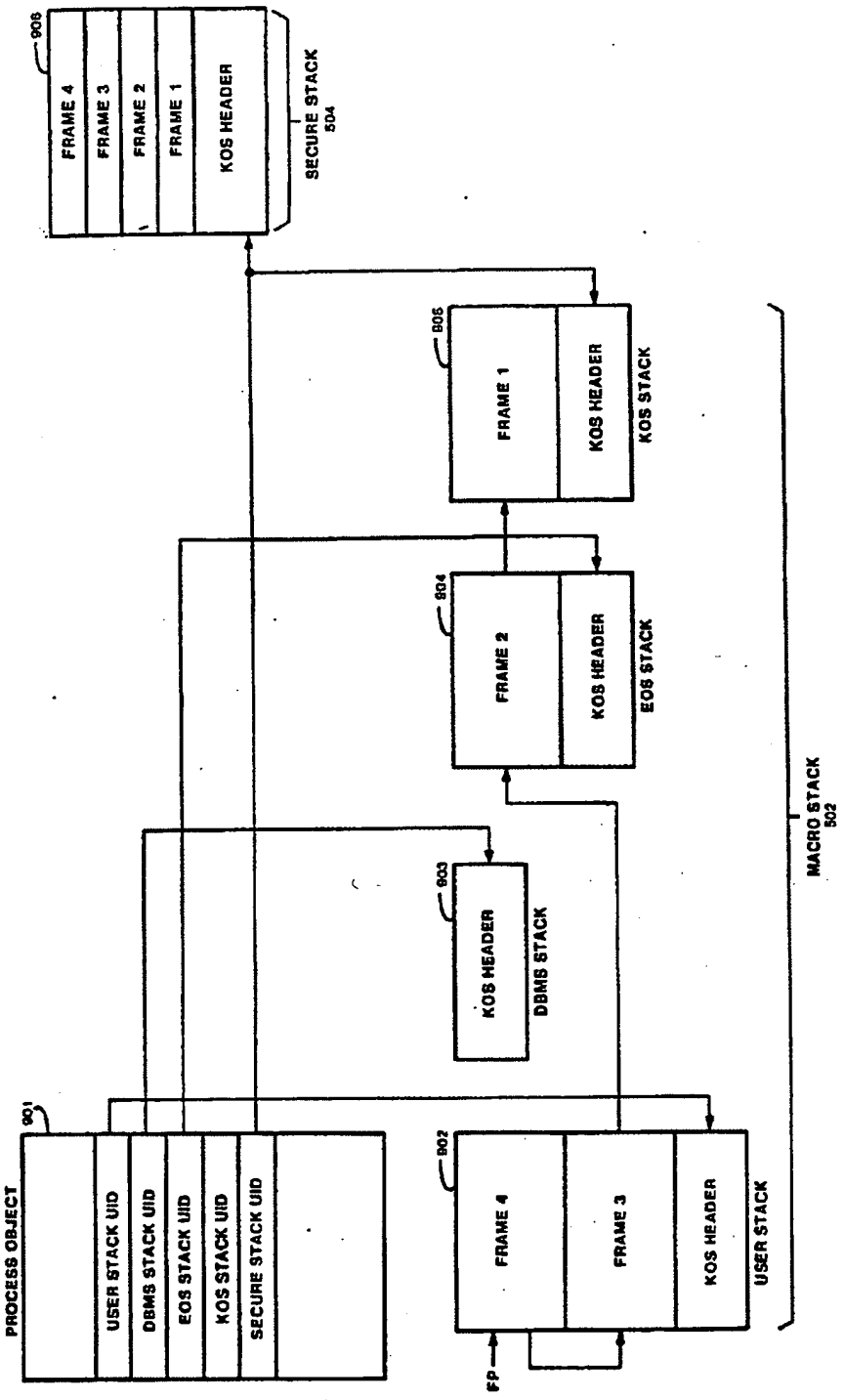


FIG 9

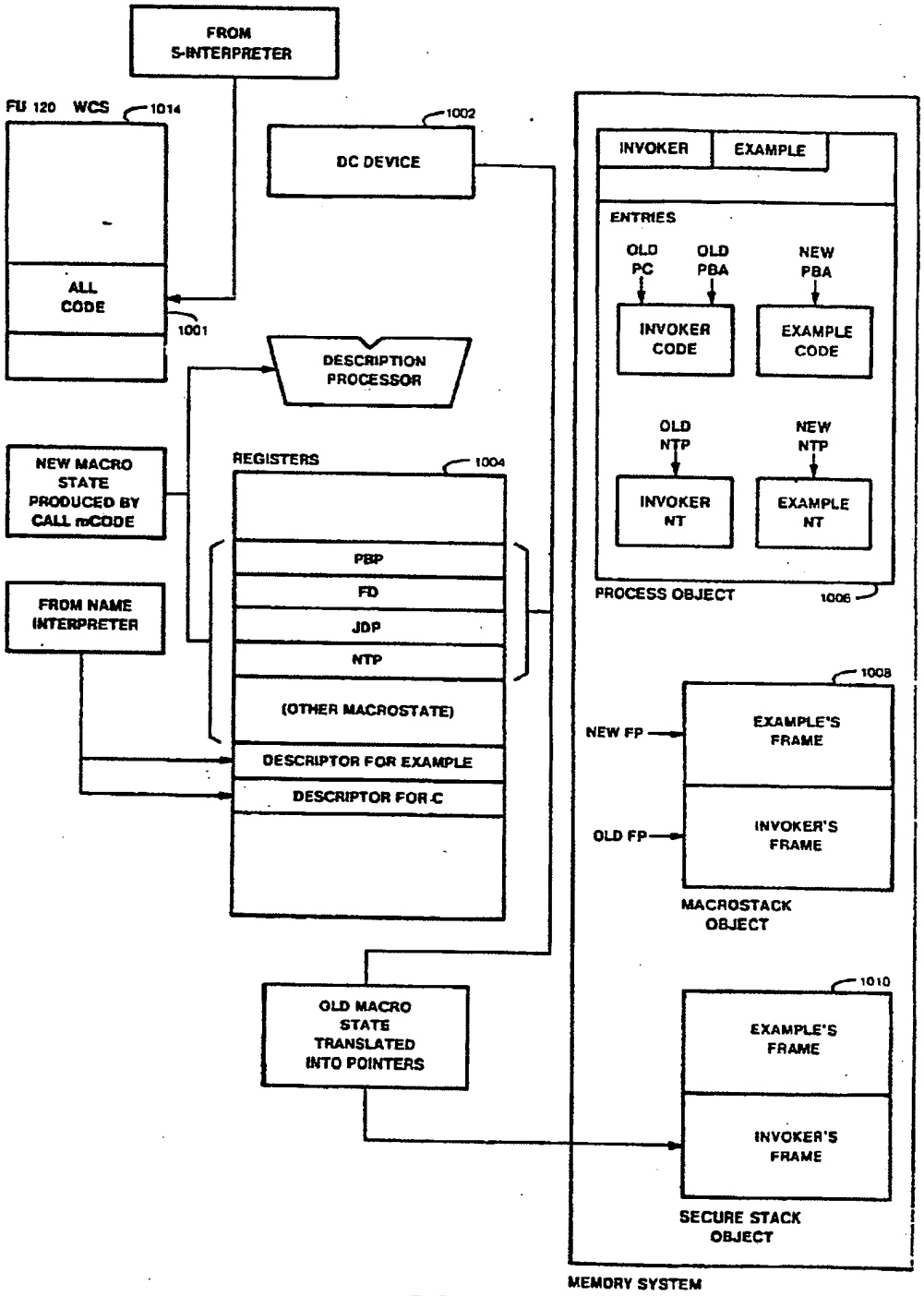


FIG 10

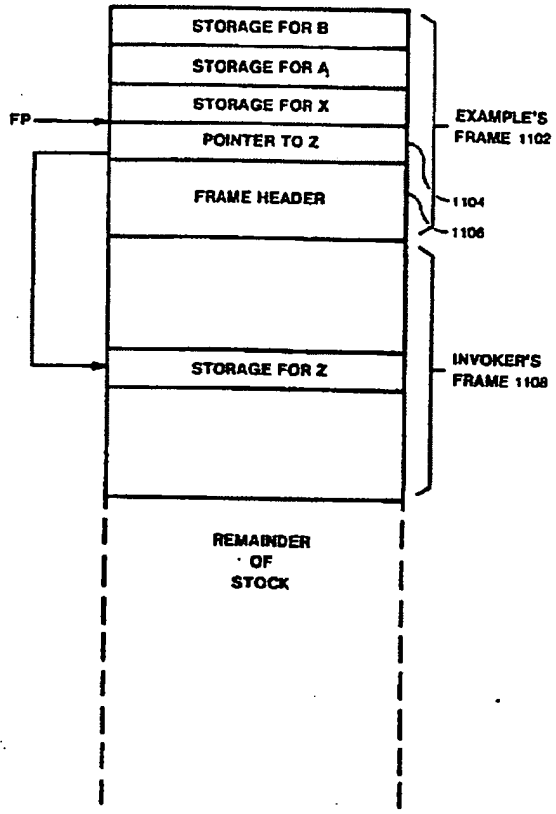


FIG 11

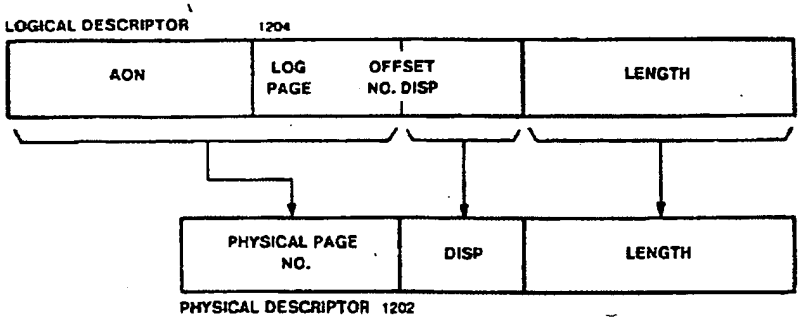


FIG 12

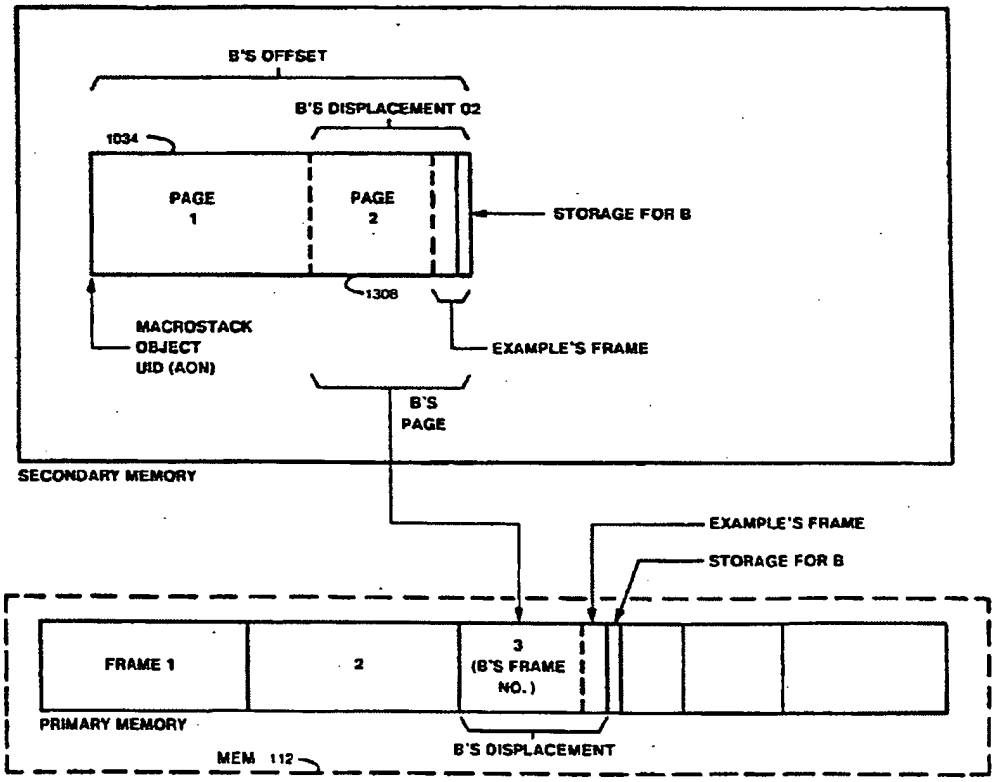


FIG 13

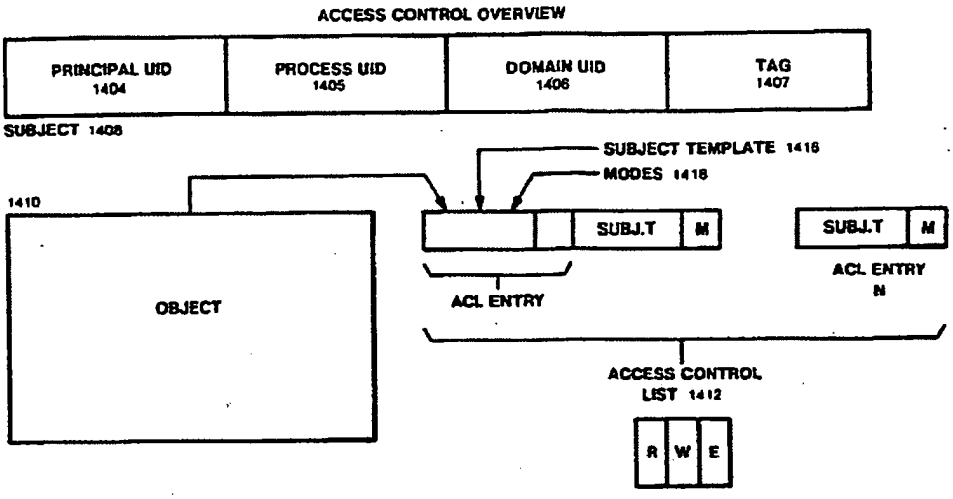


FIG 14

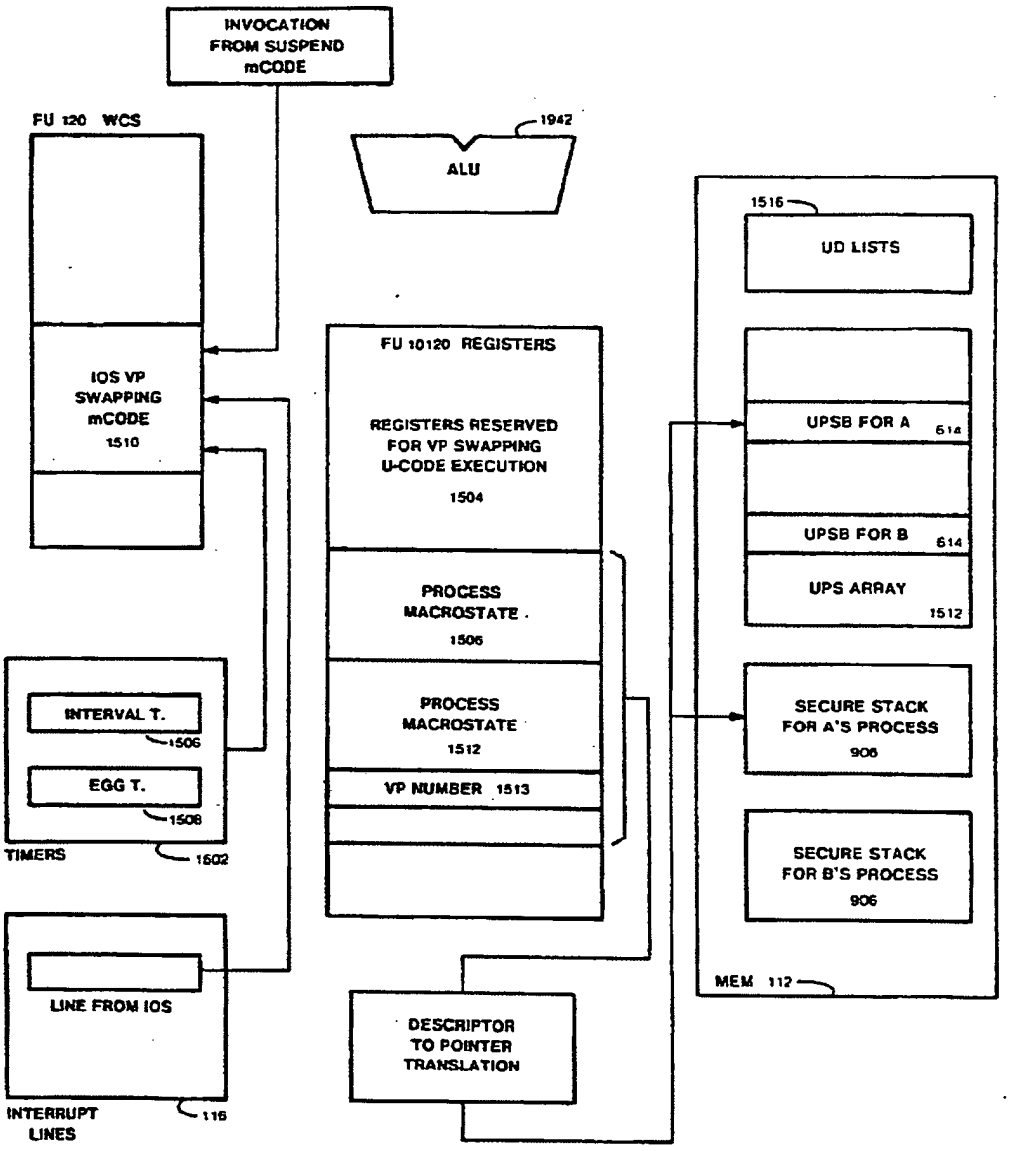


FIG 15

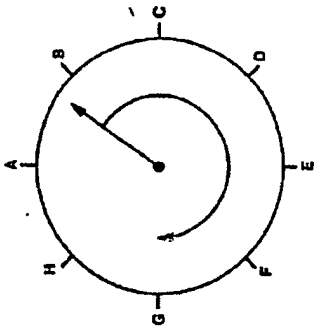


FIG 17

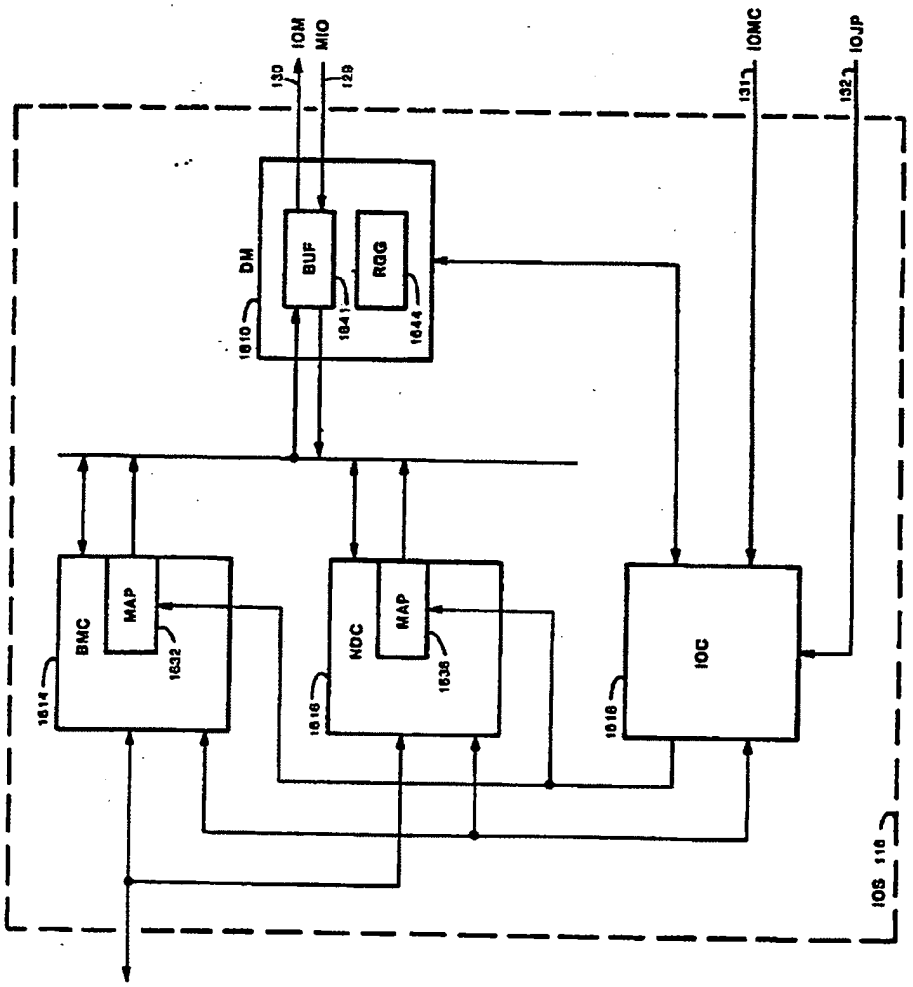


FIG 16

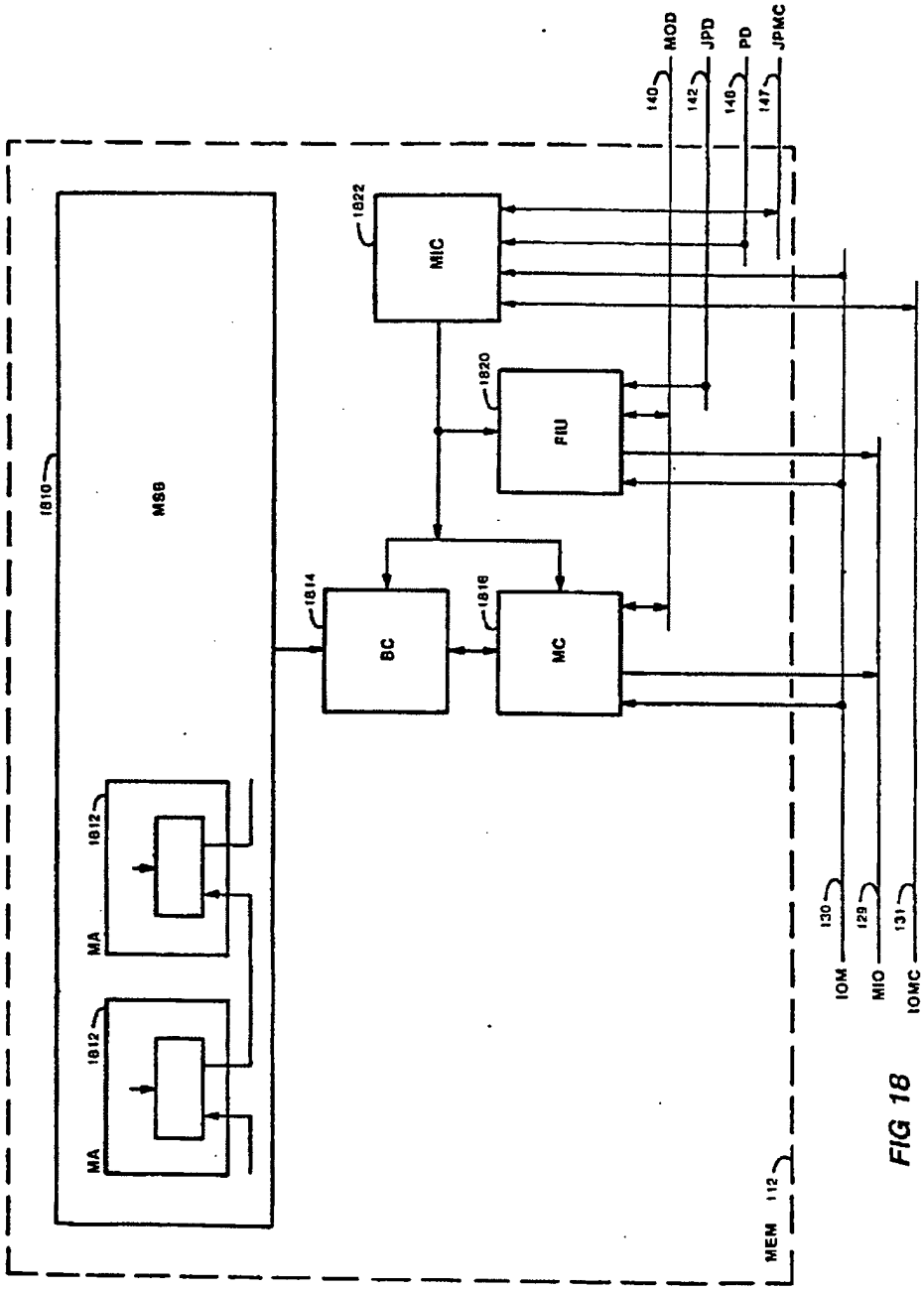


FIG 18

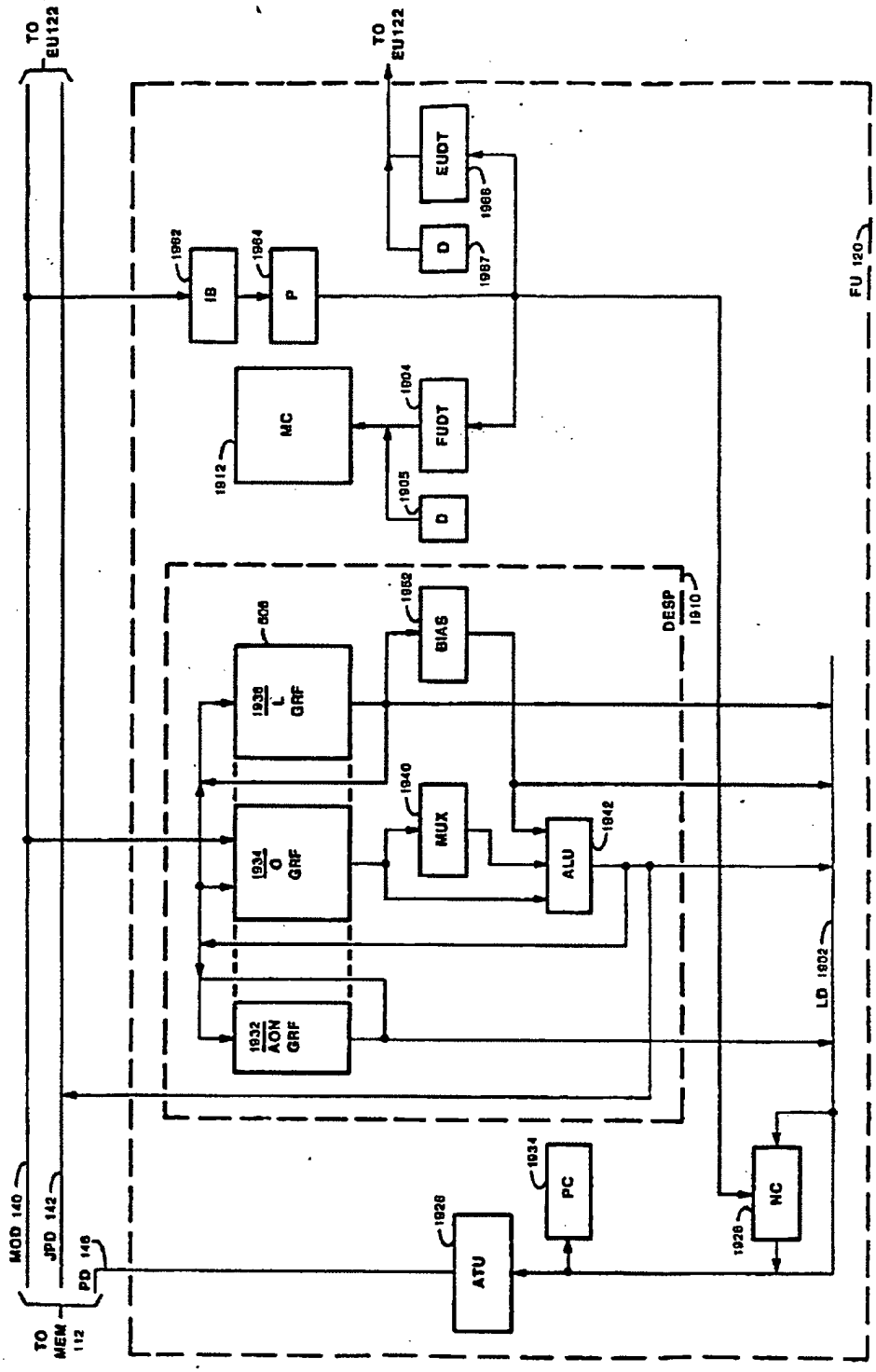


FIG 19

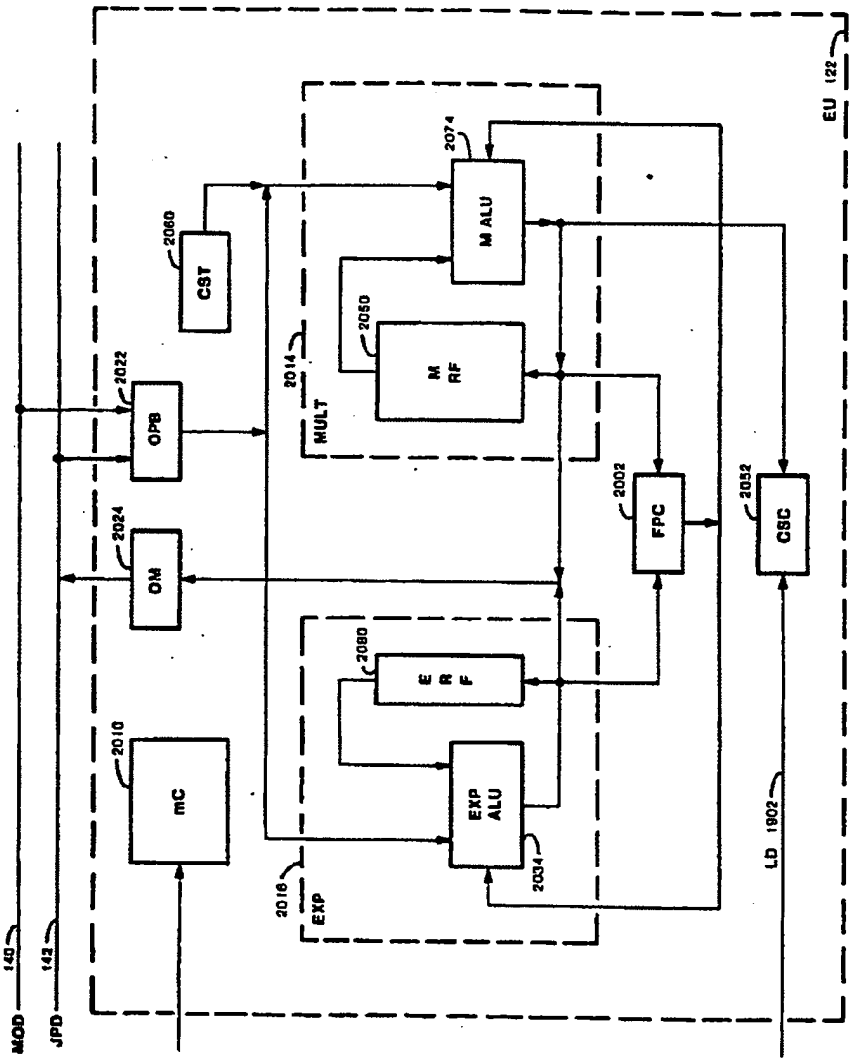


FIG 20

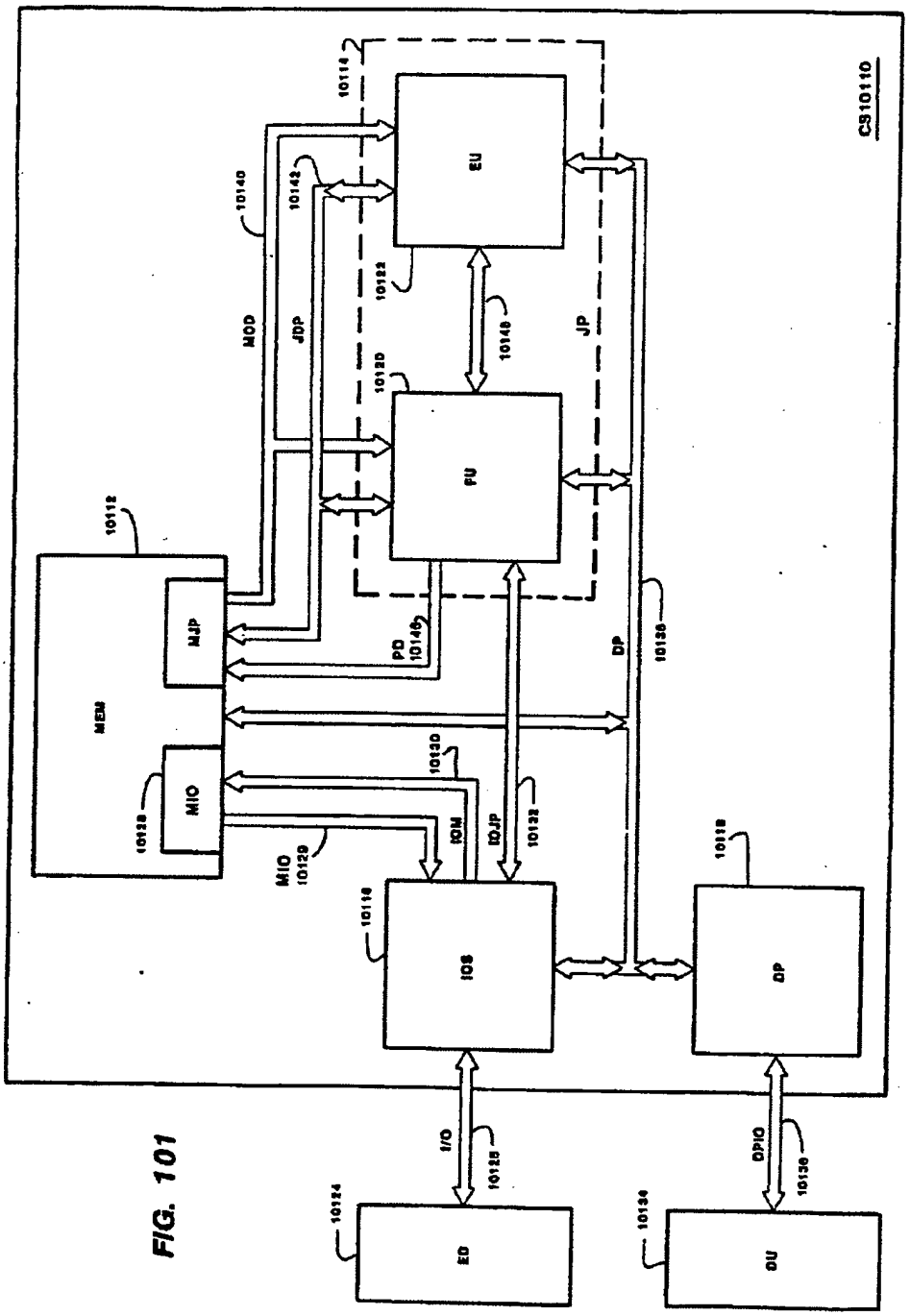


FIG. 101

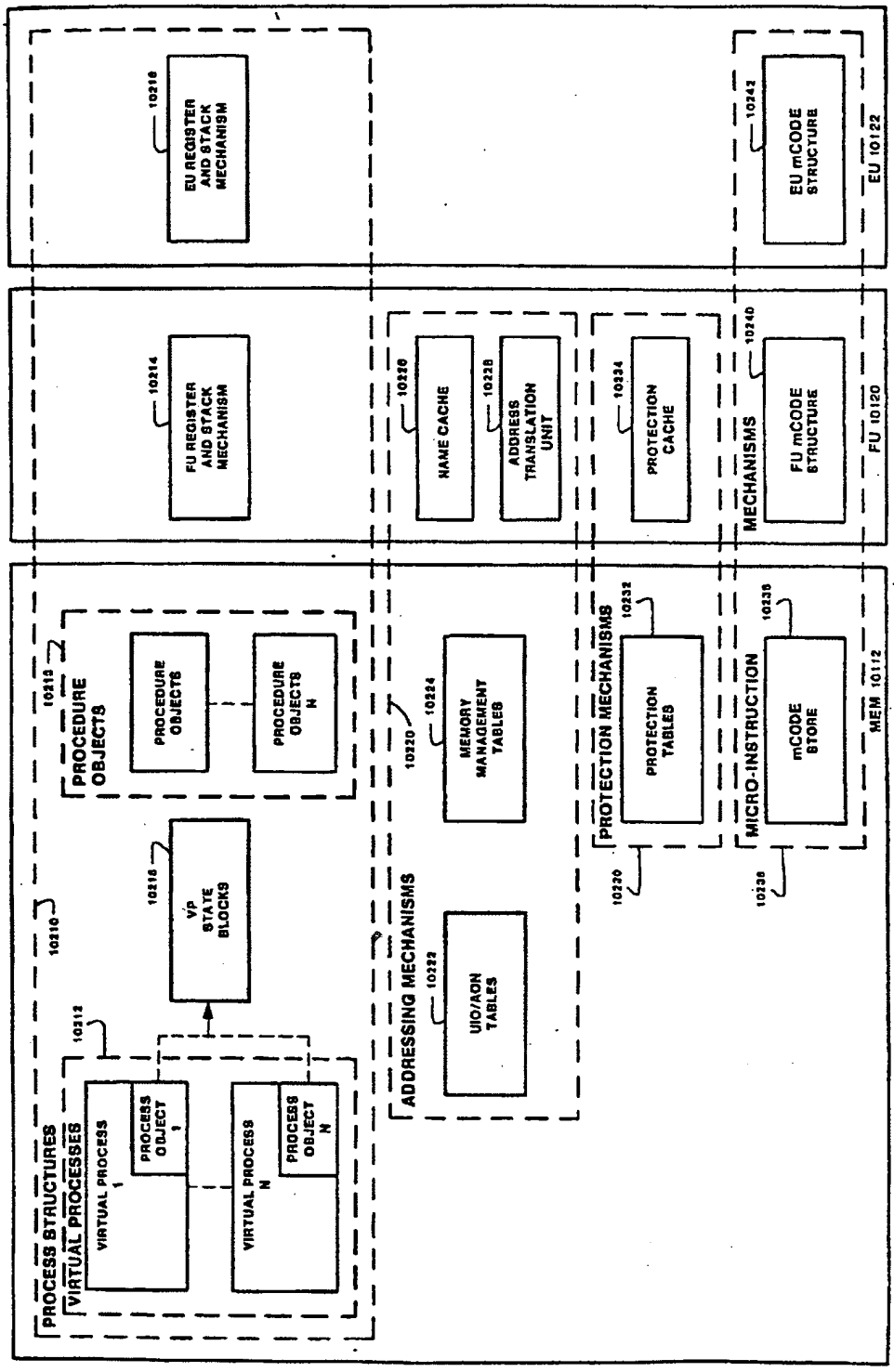


FIG. 102

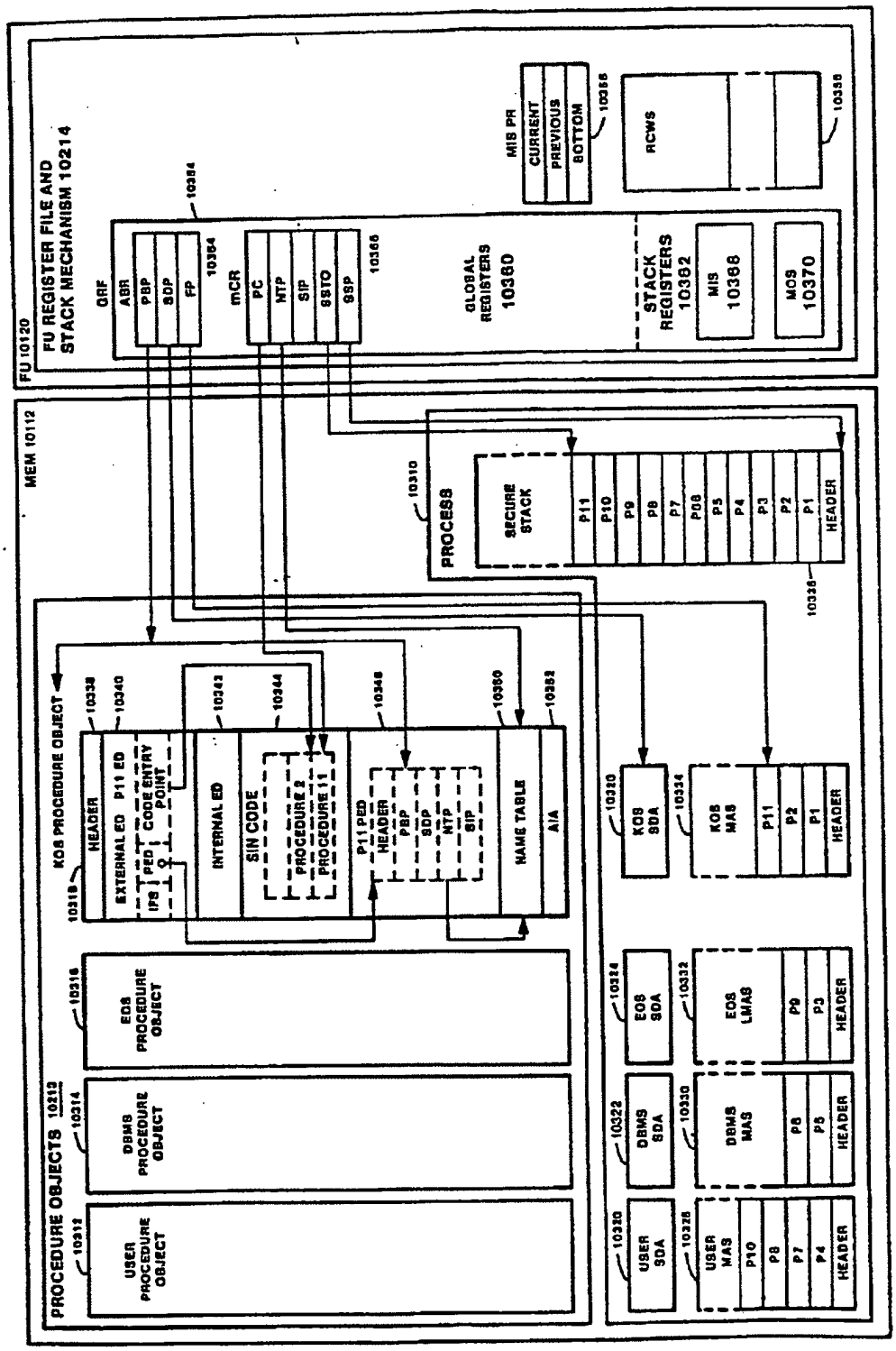


FIG. 103

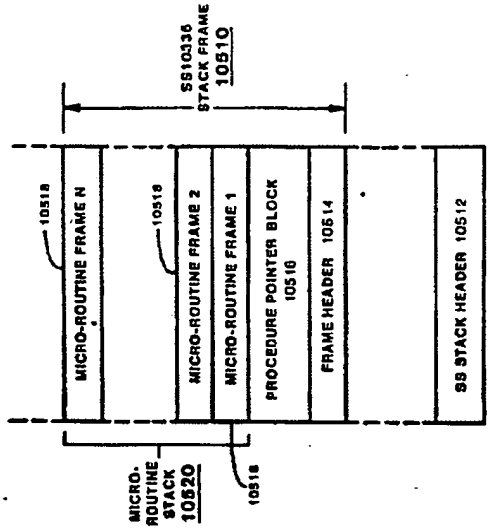


FIG. 105

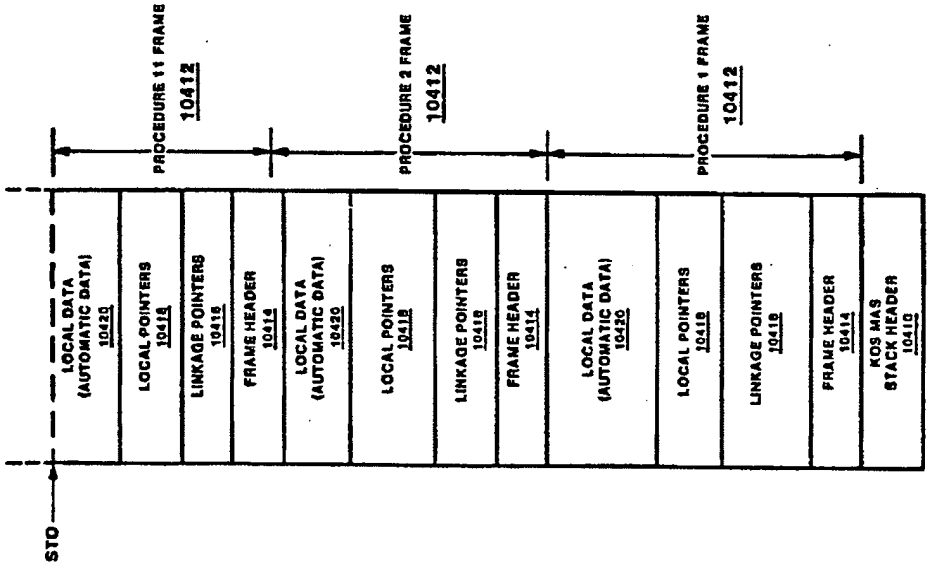


FIG. 104

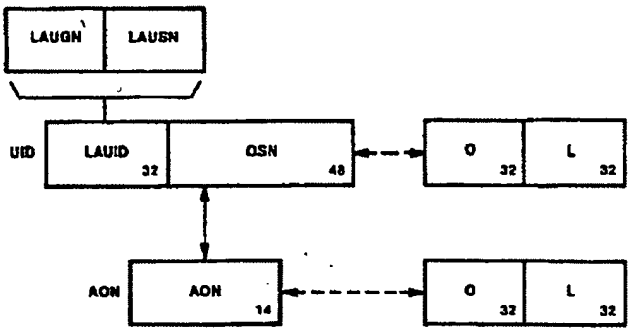


FIG. 106A

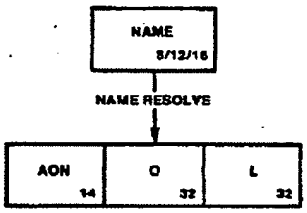


FIG. 106B

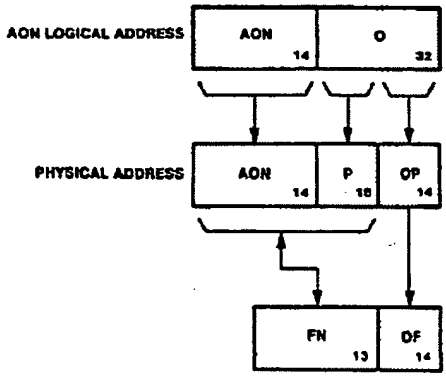


FIG. 106C

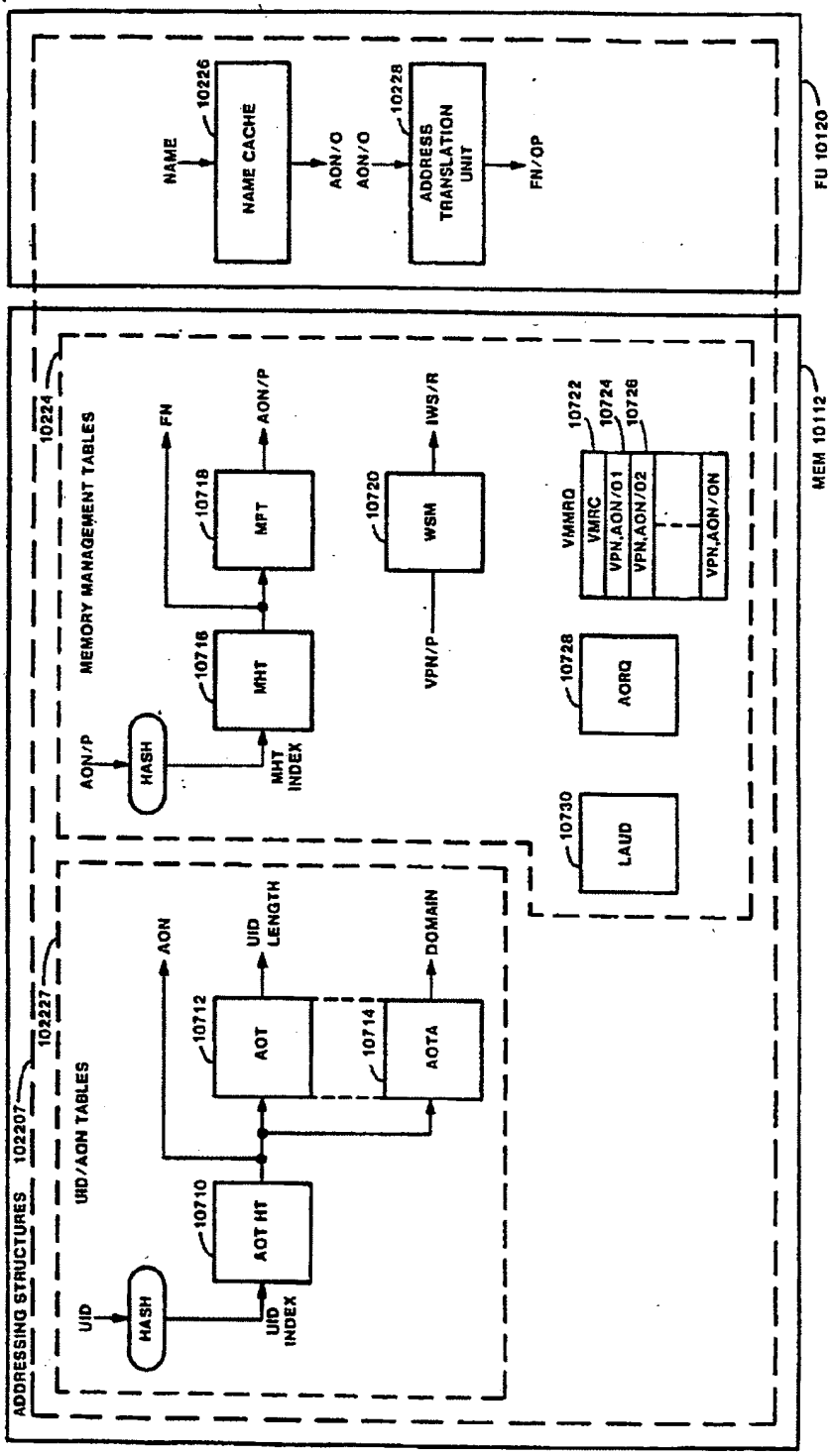


FIG 107

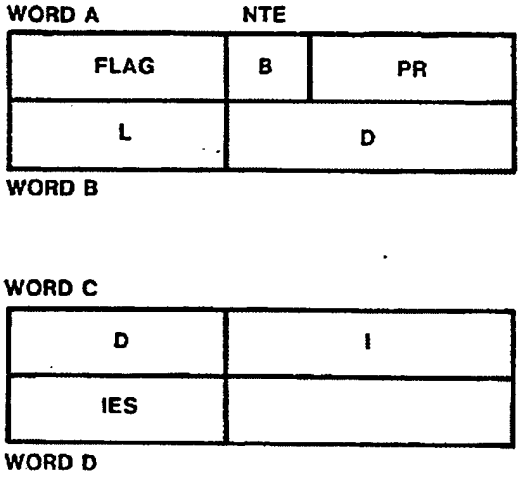


FIG. 108

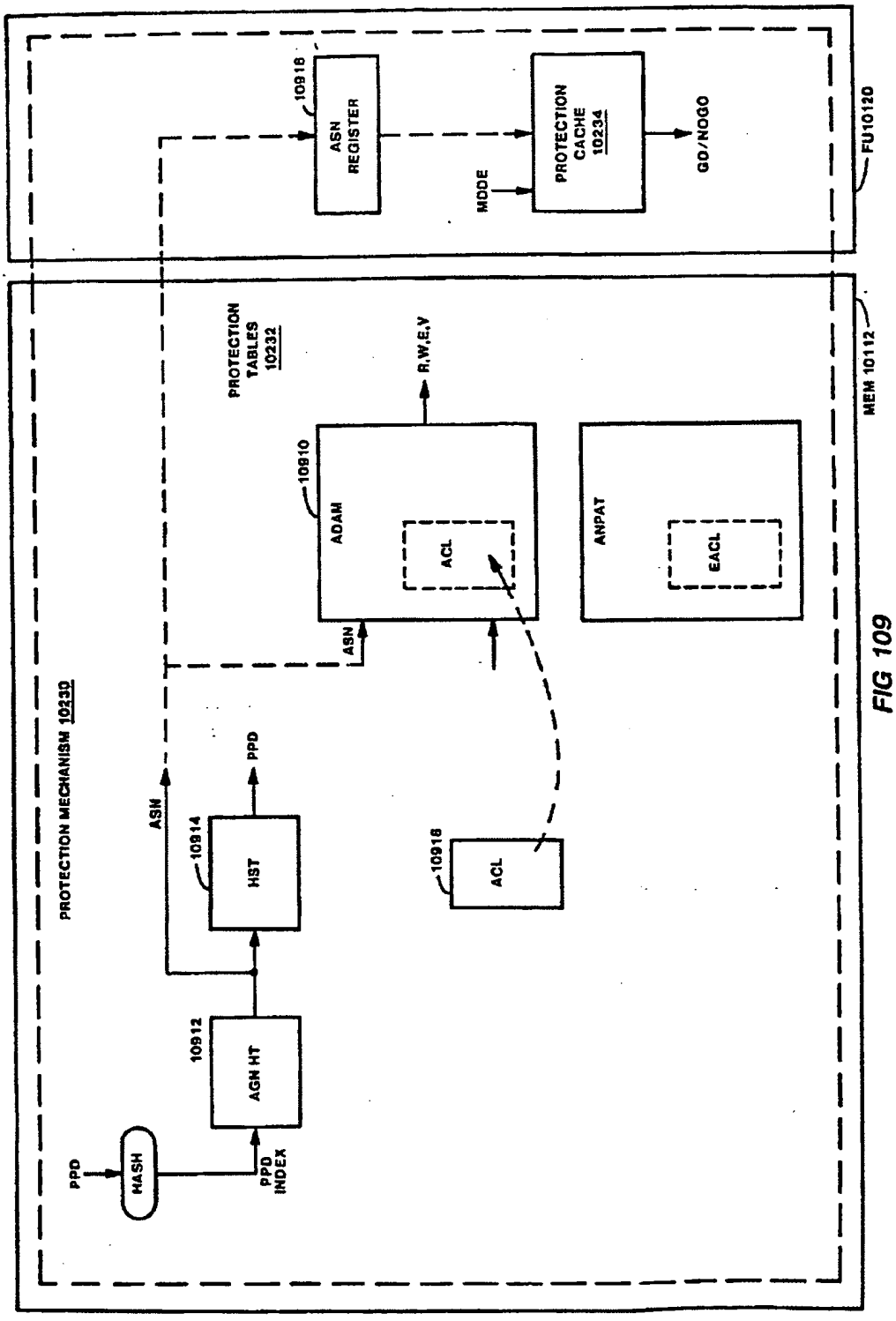


FIG 109

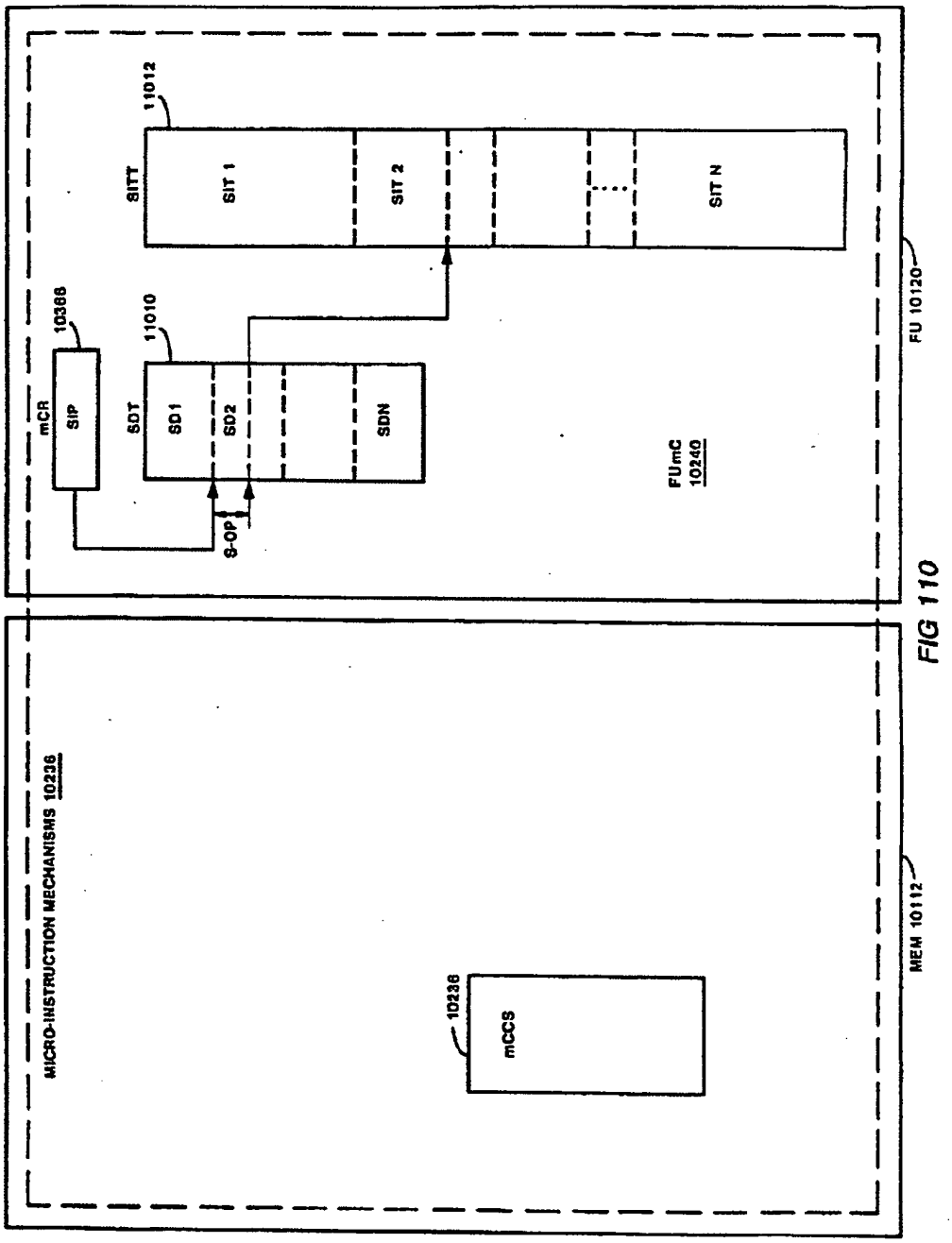


FIG 110

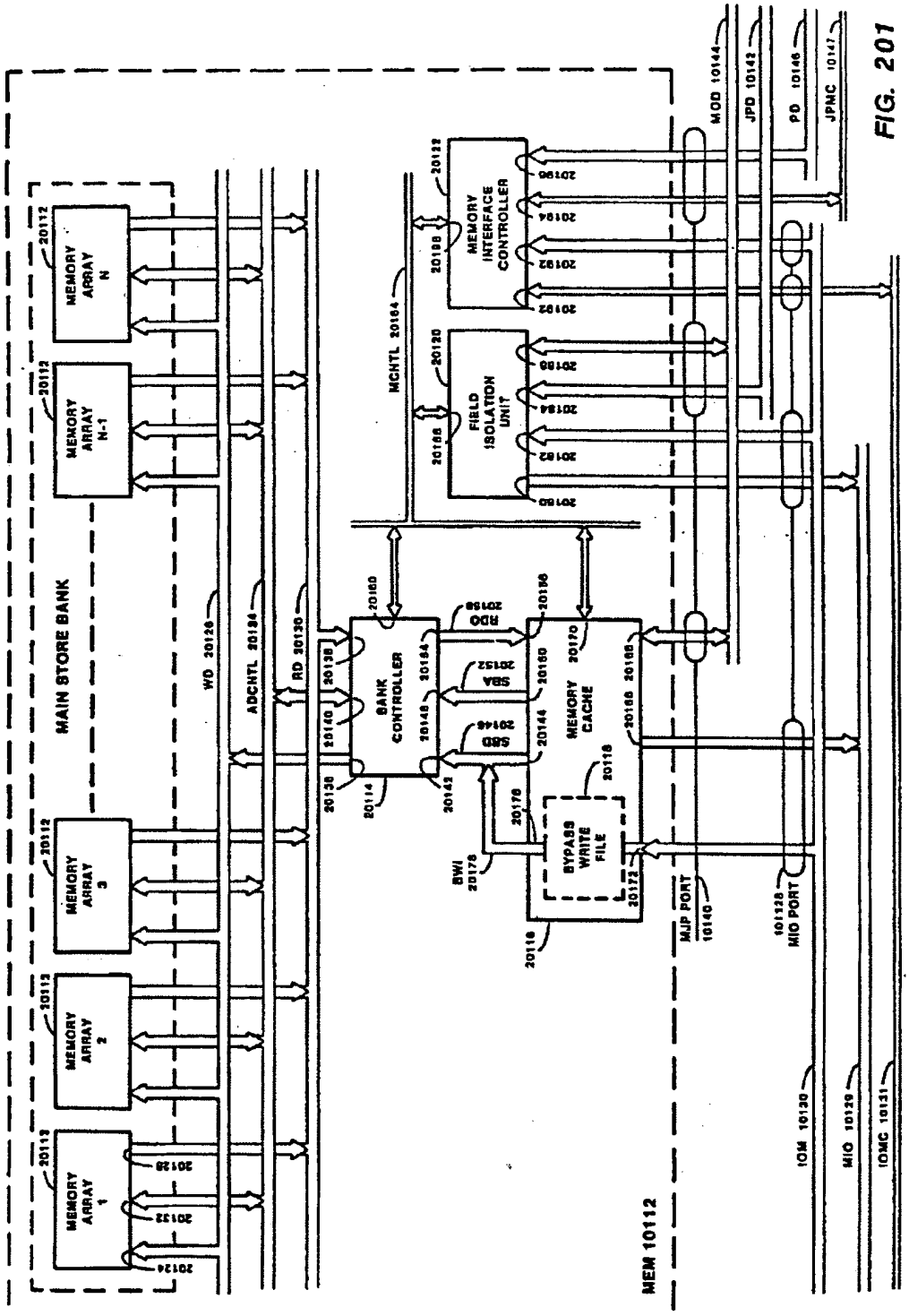


FIG. 201

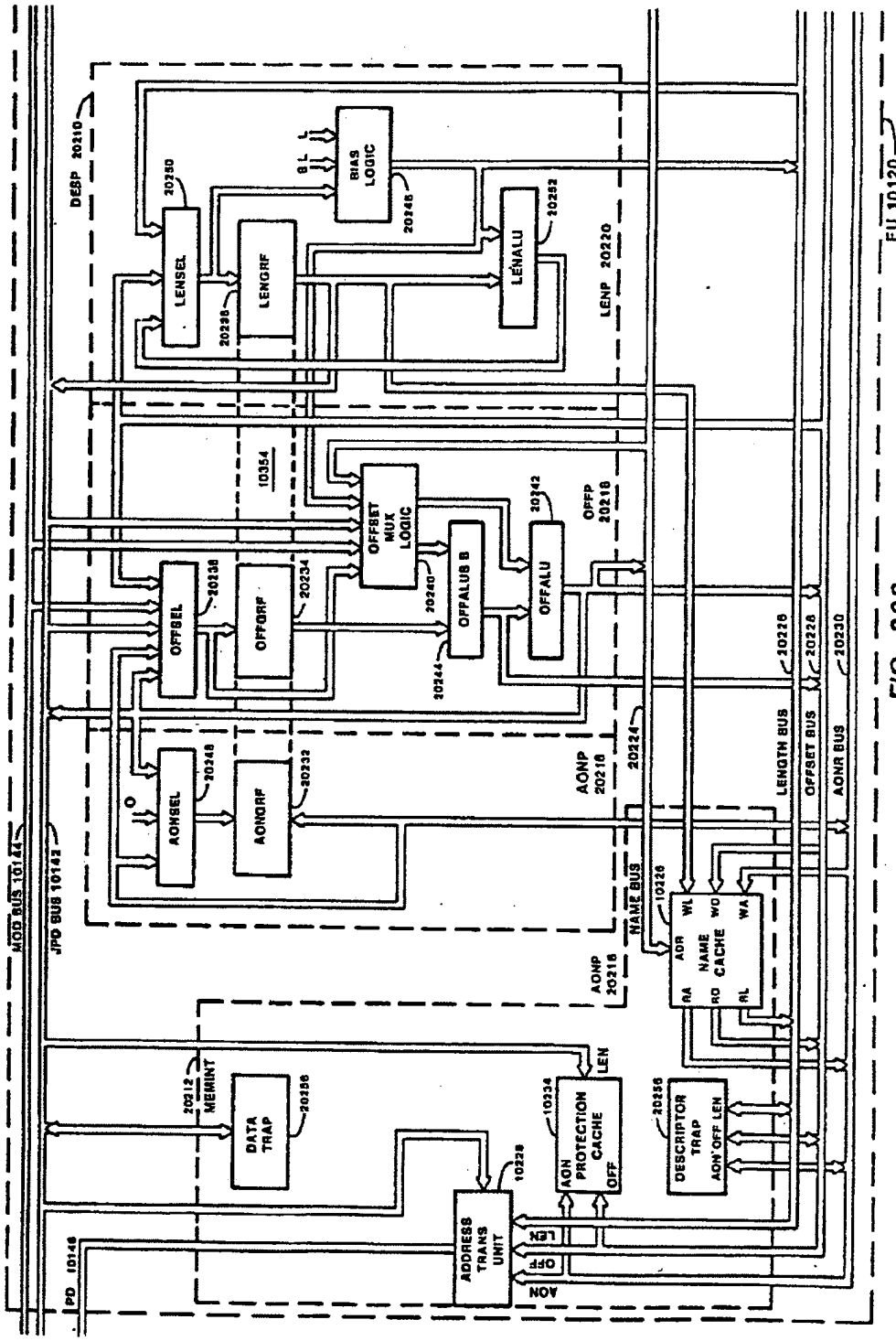


FIG. 202

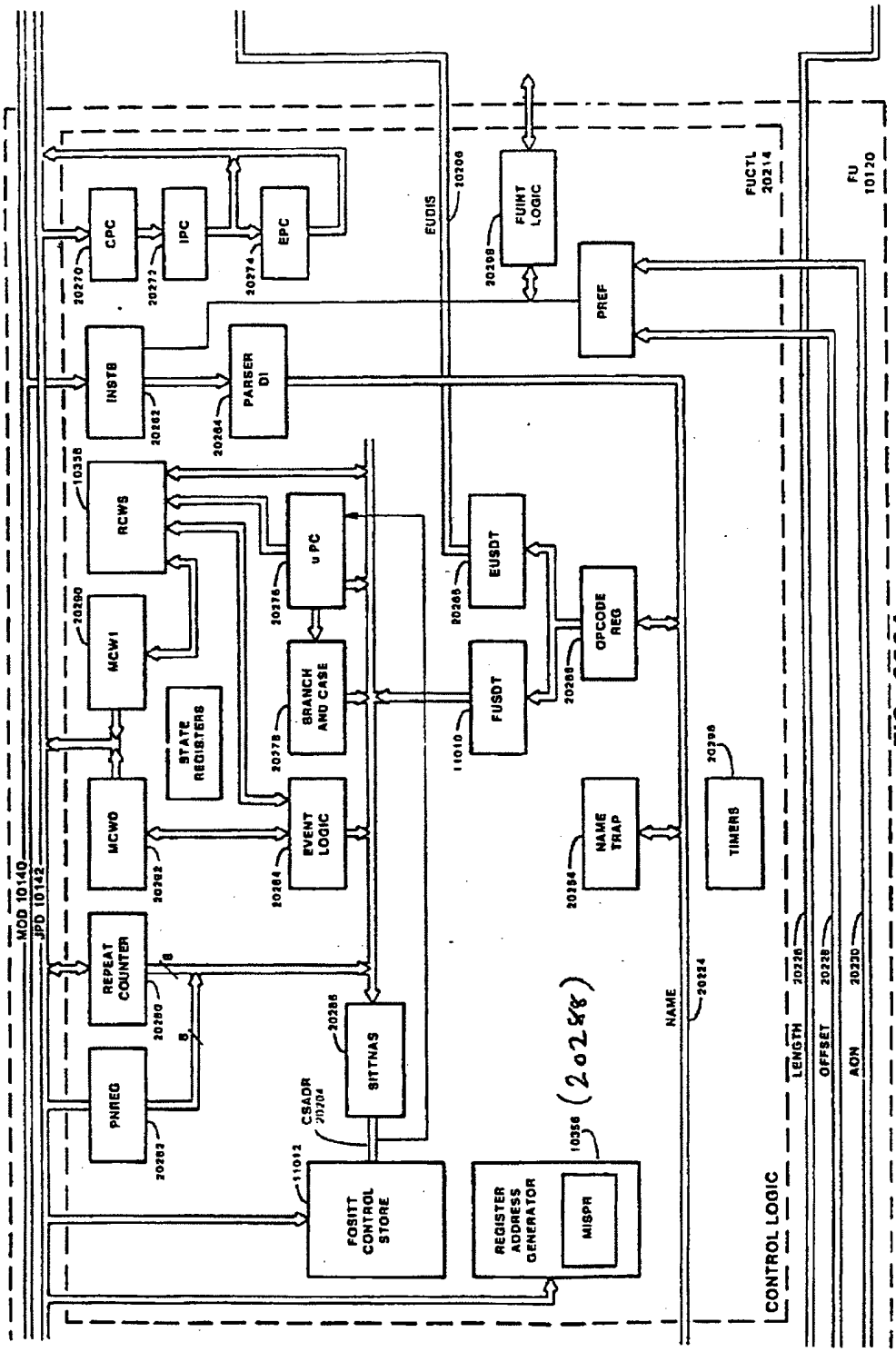


FIG. 202A

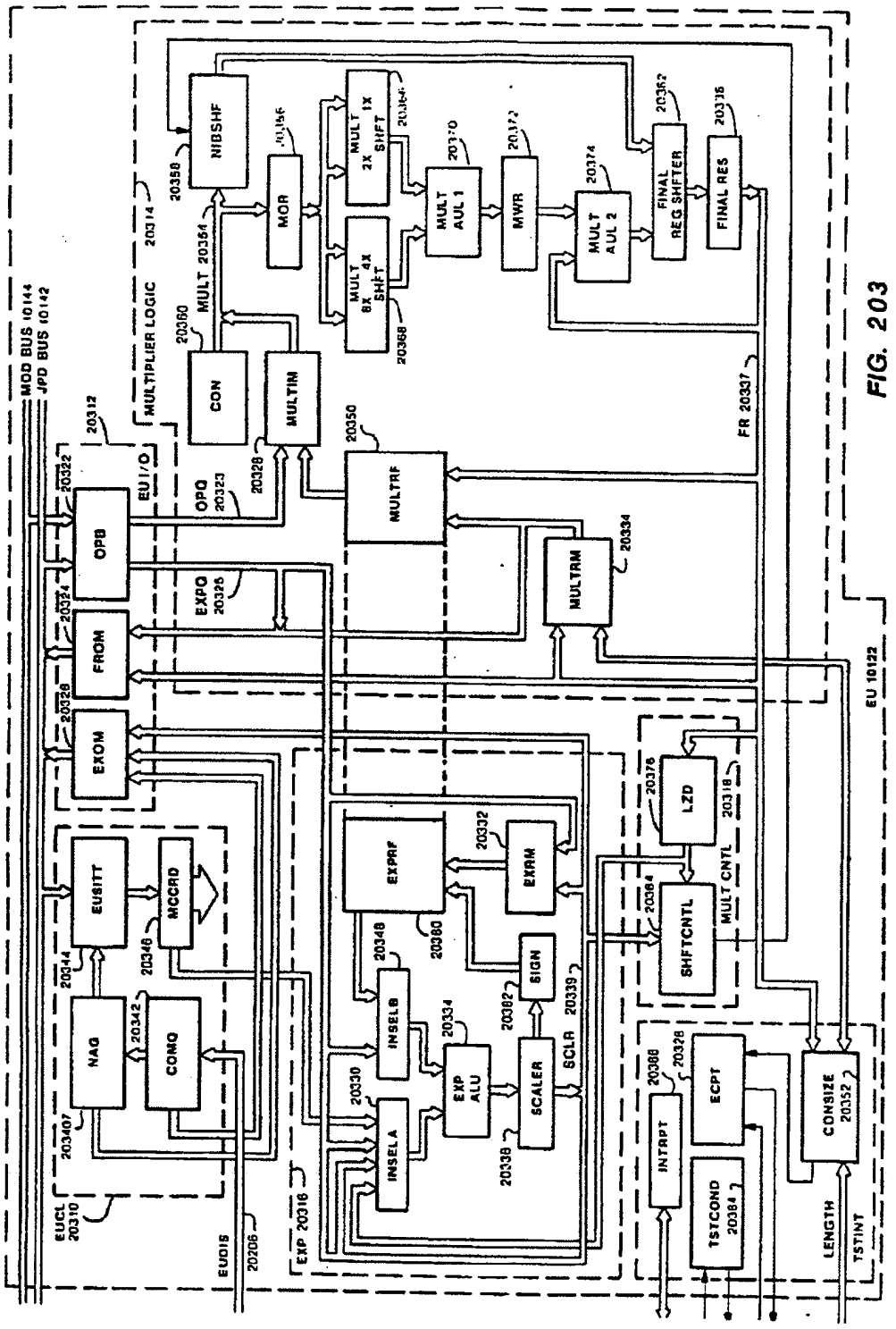


FIG. 203

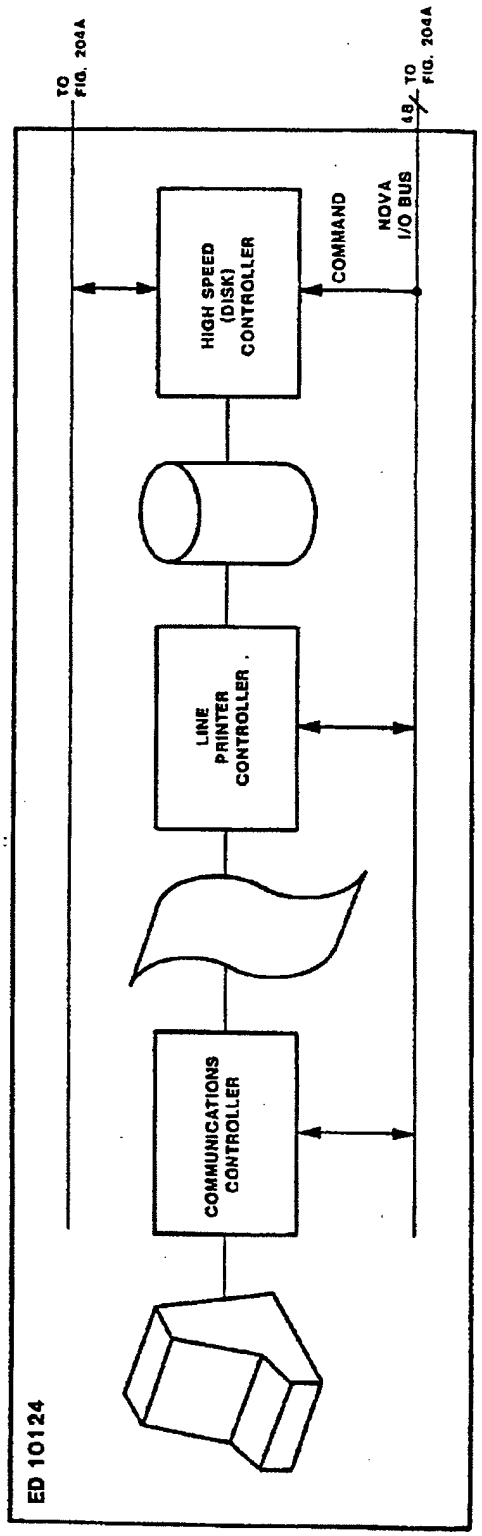


FIG. 204

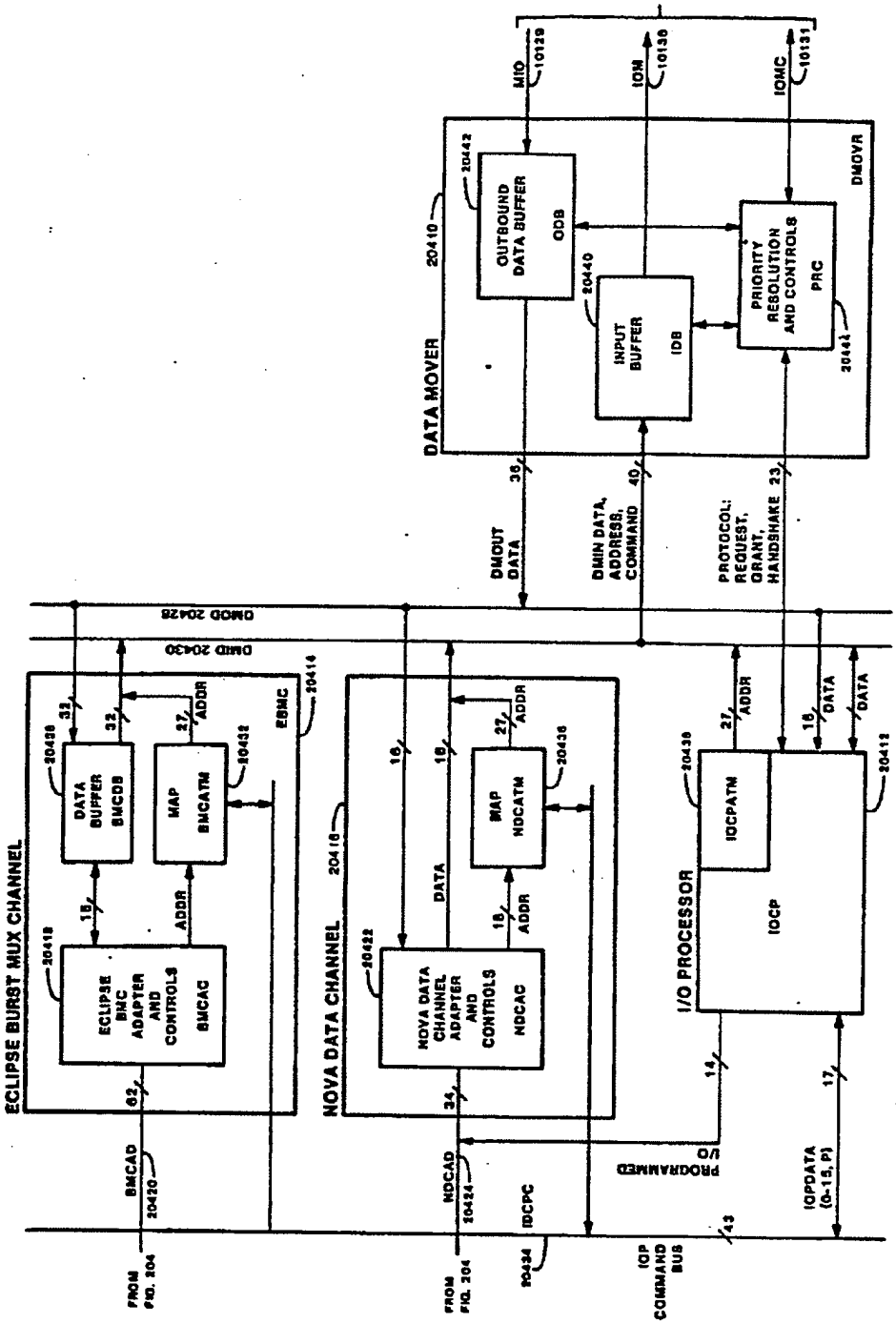


FIG. 204A

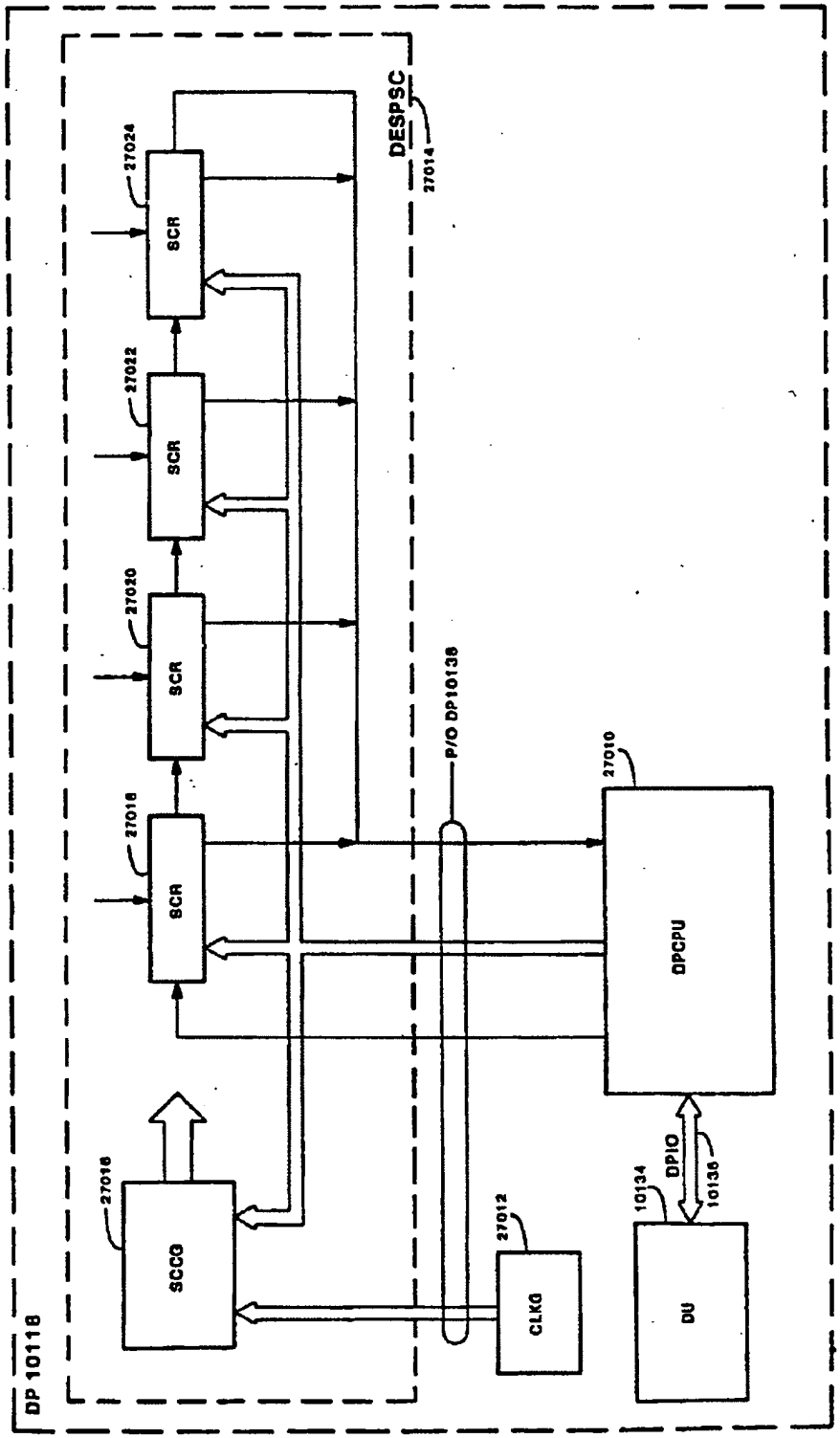


FIG. 205

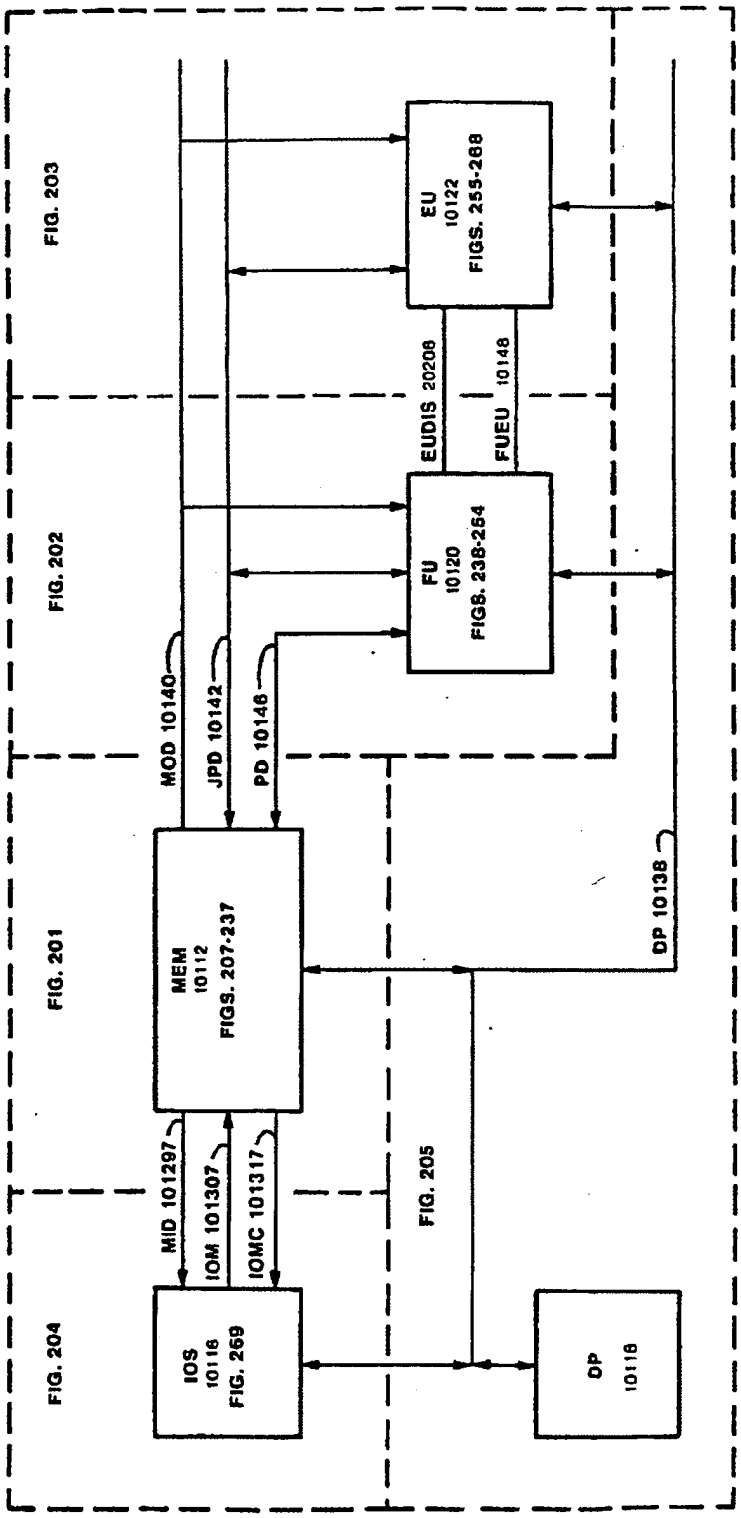


FIG. 206

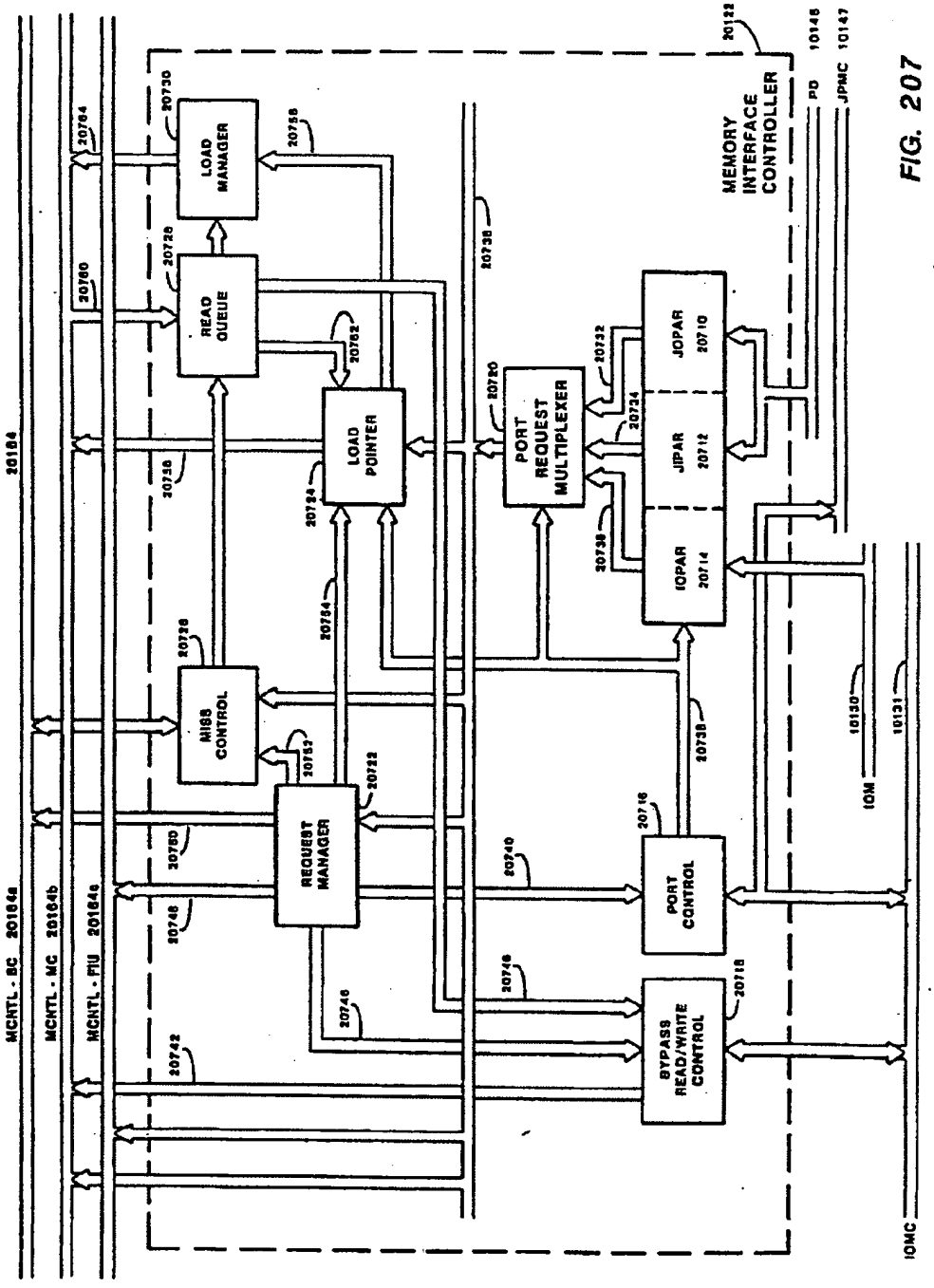


FIG. 207

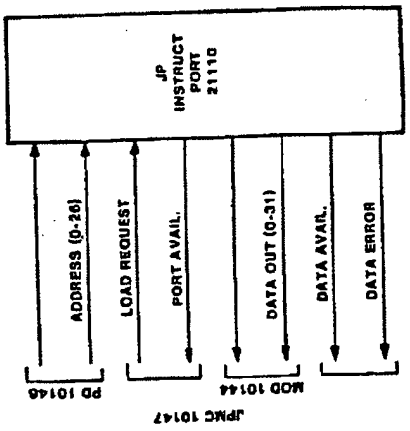


FIG. 211

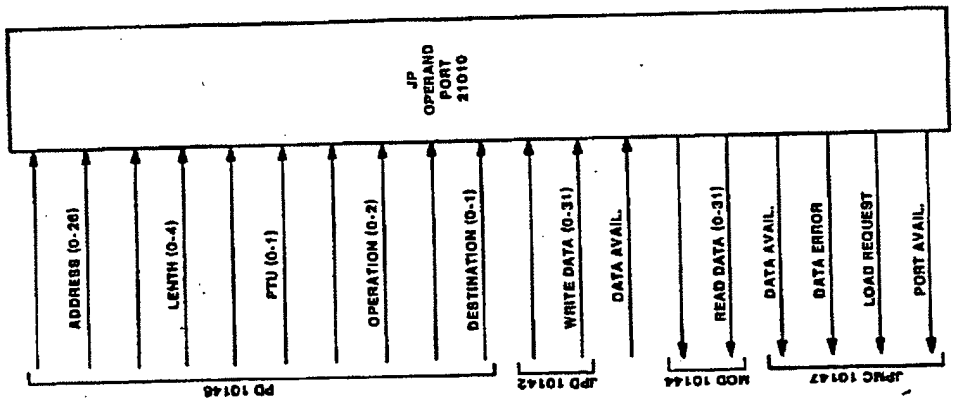


FIG. 210

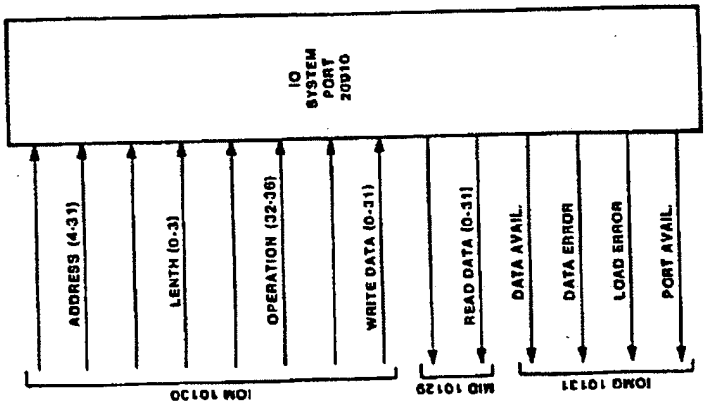
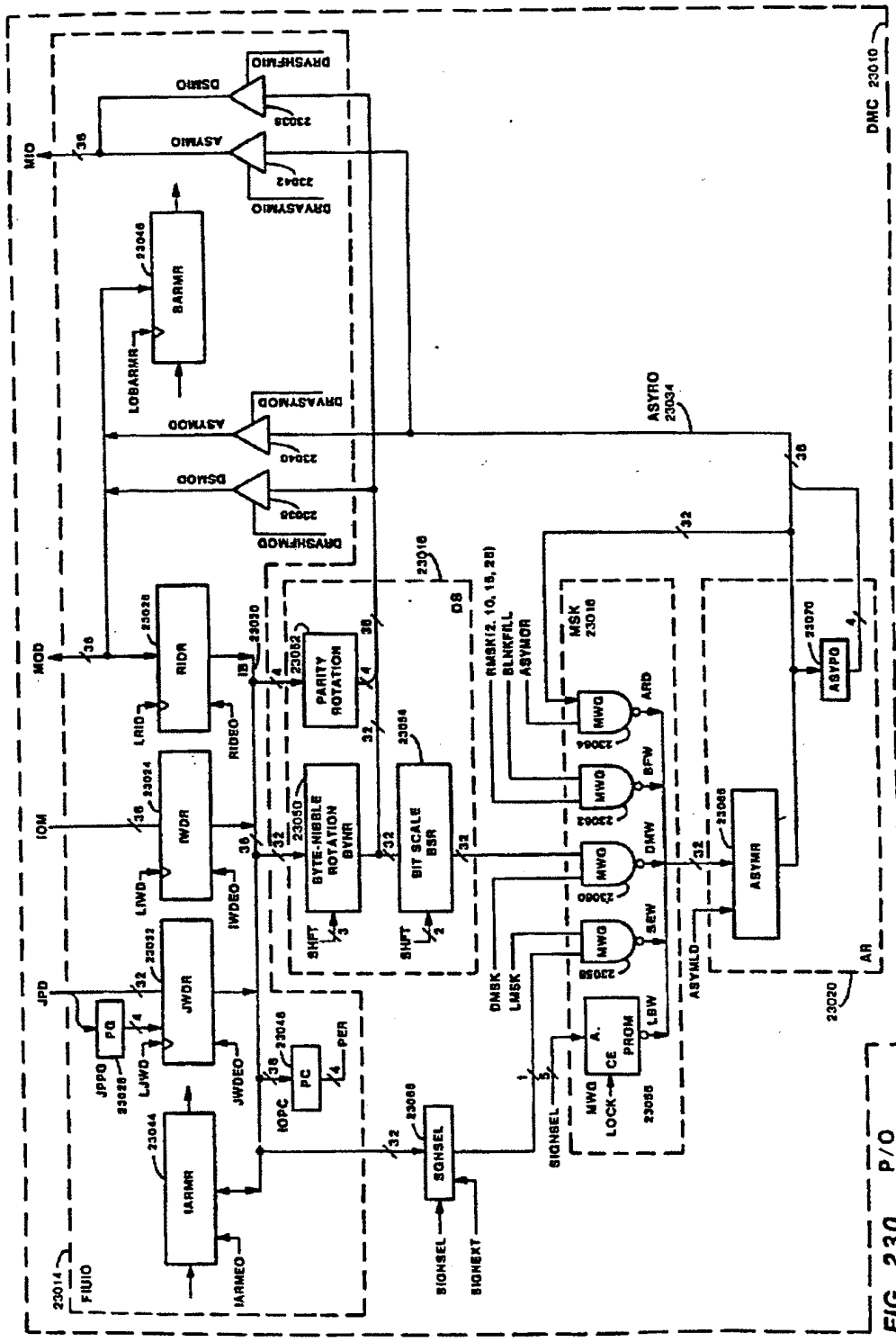


FIG. 209



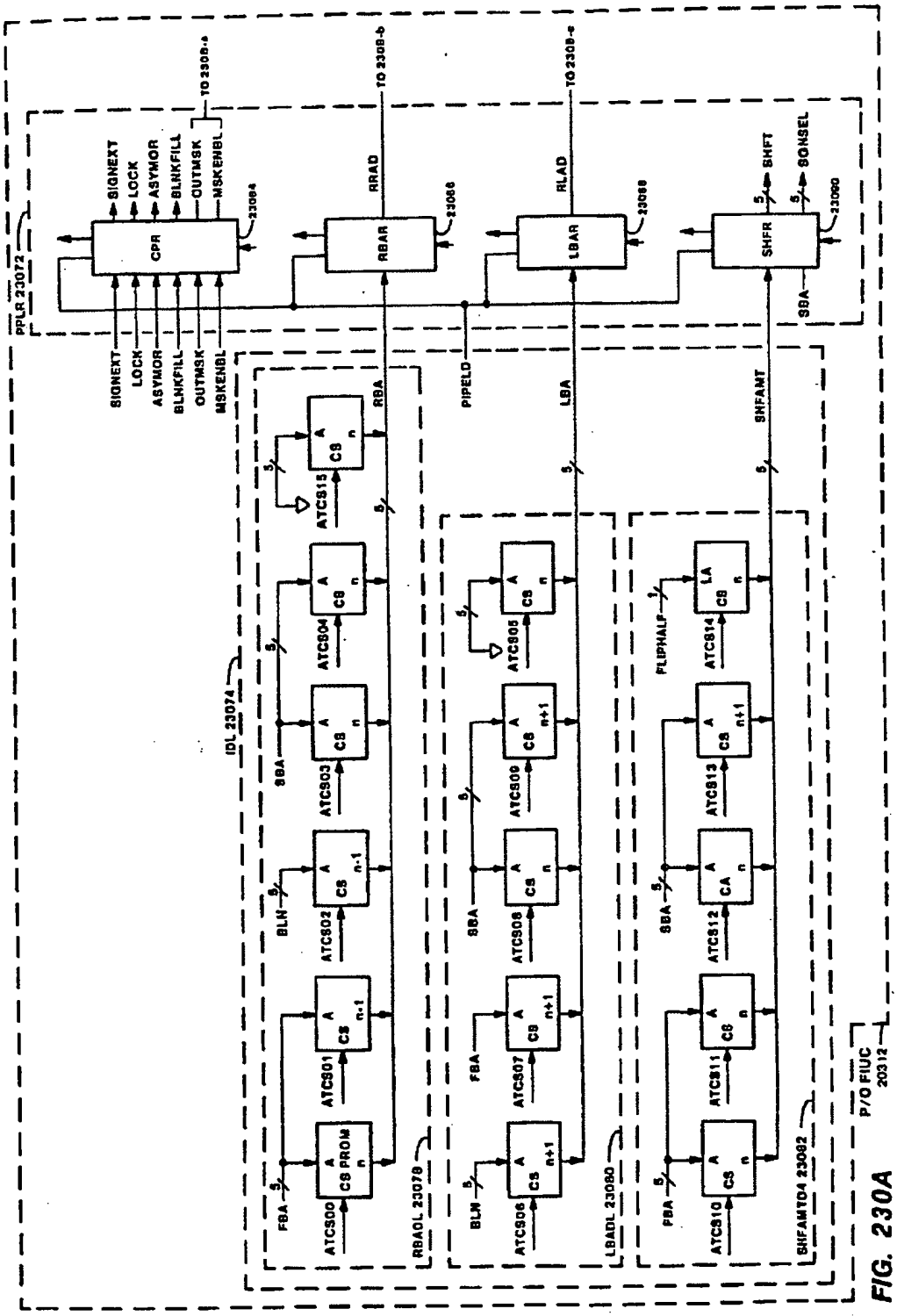


FIG. 230A

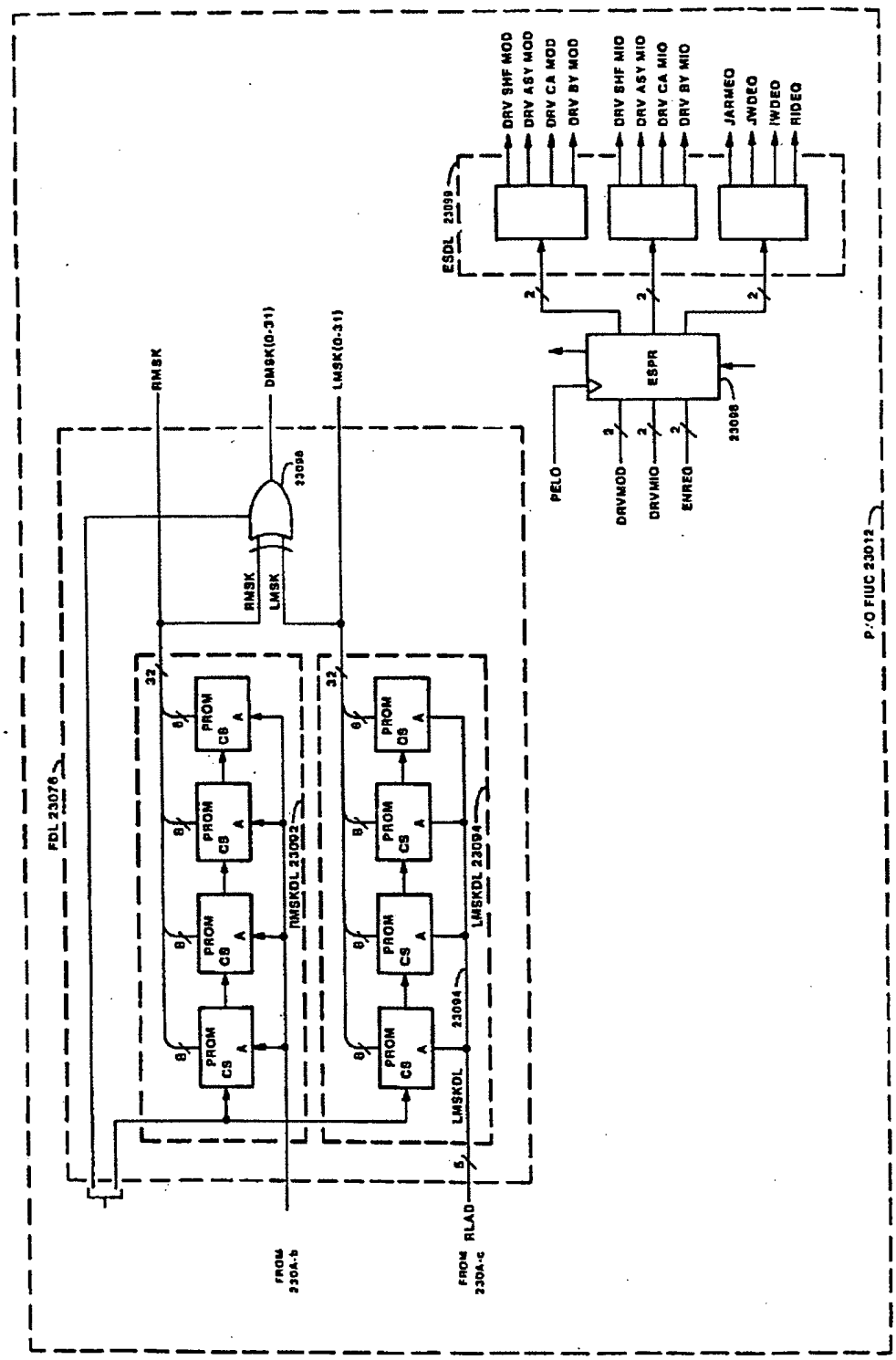


FIG 230B

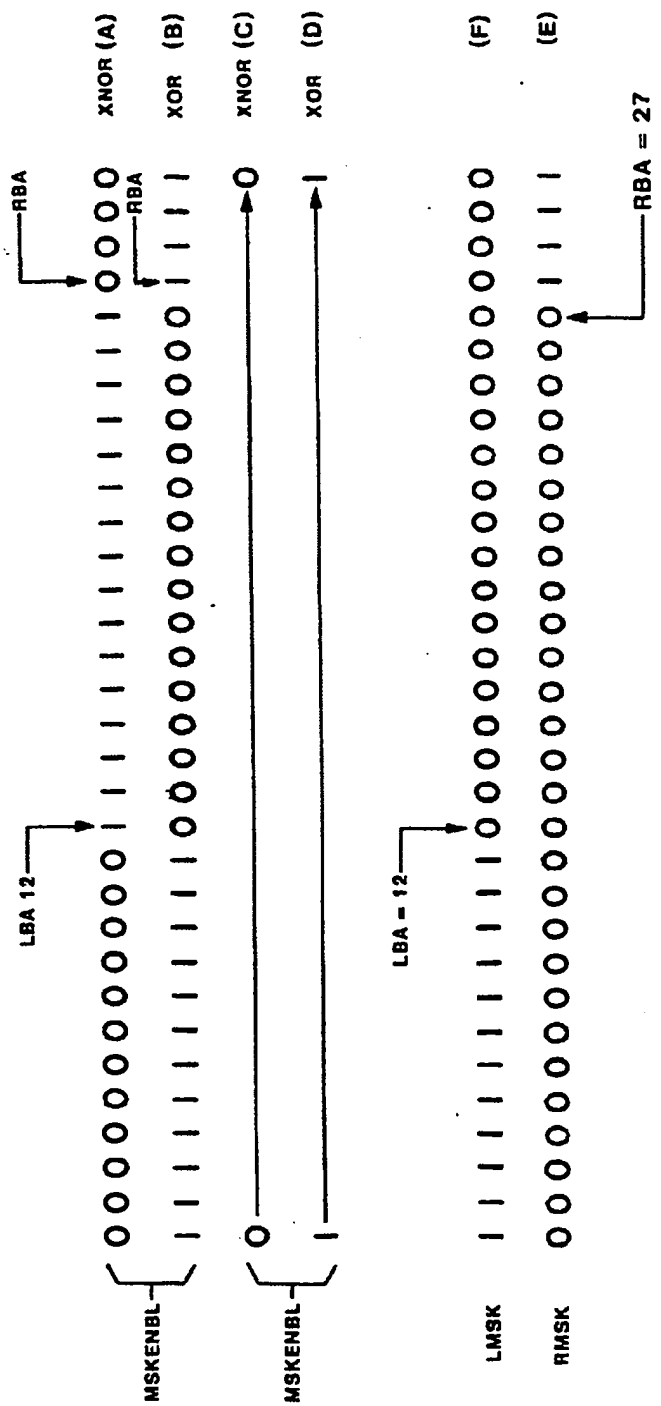


FIG. 231

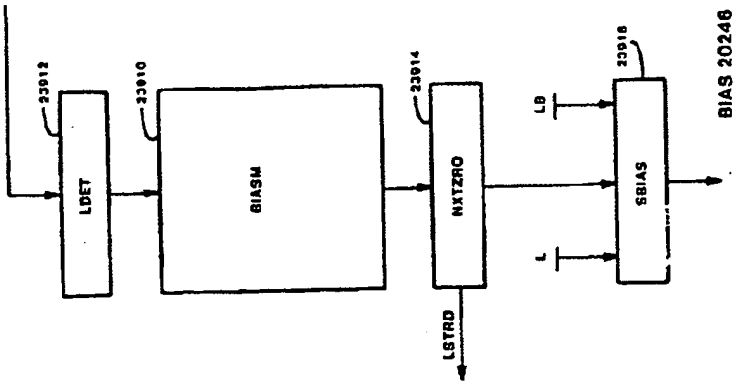


FIG. 239

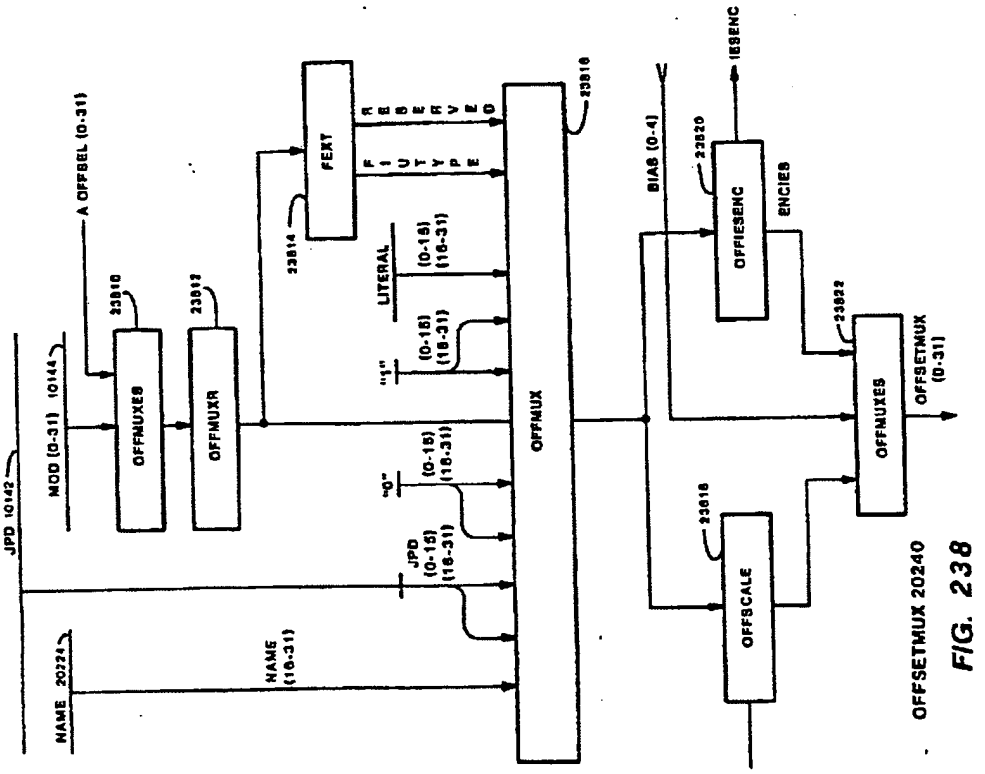
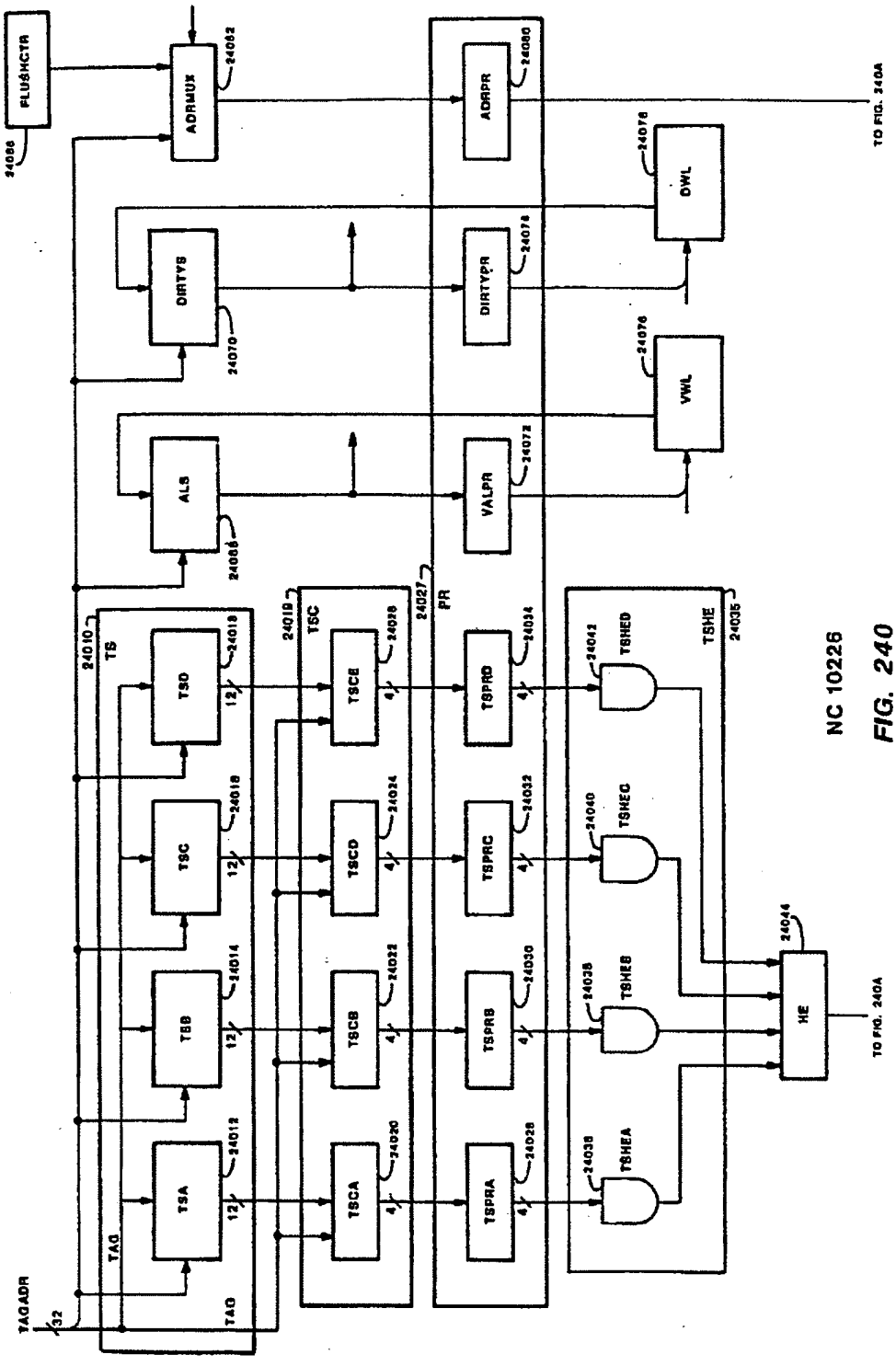
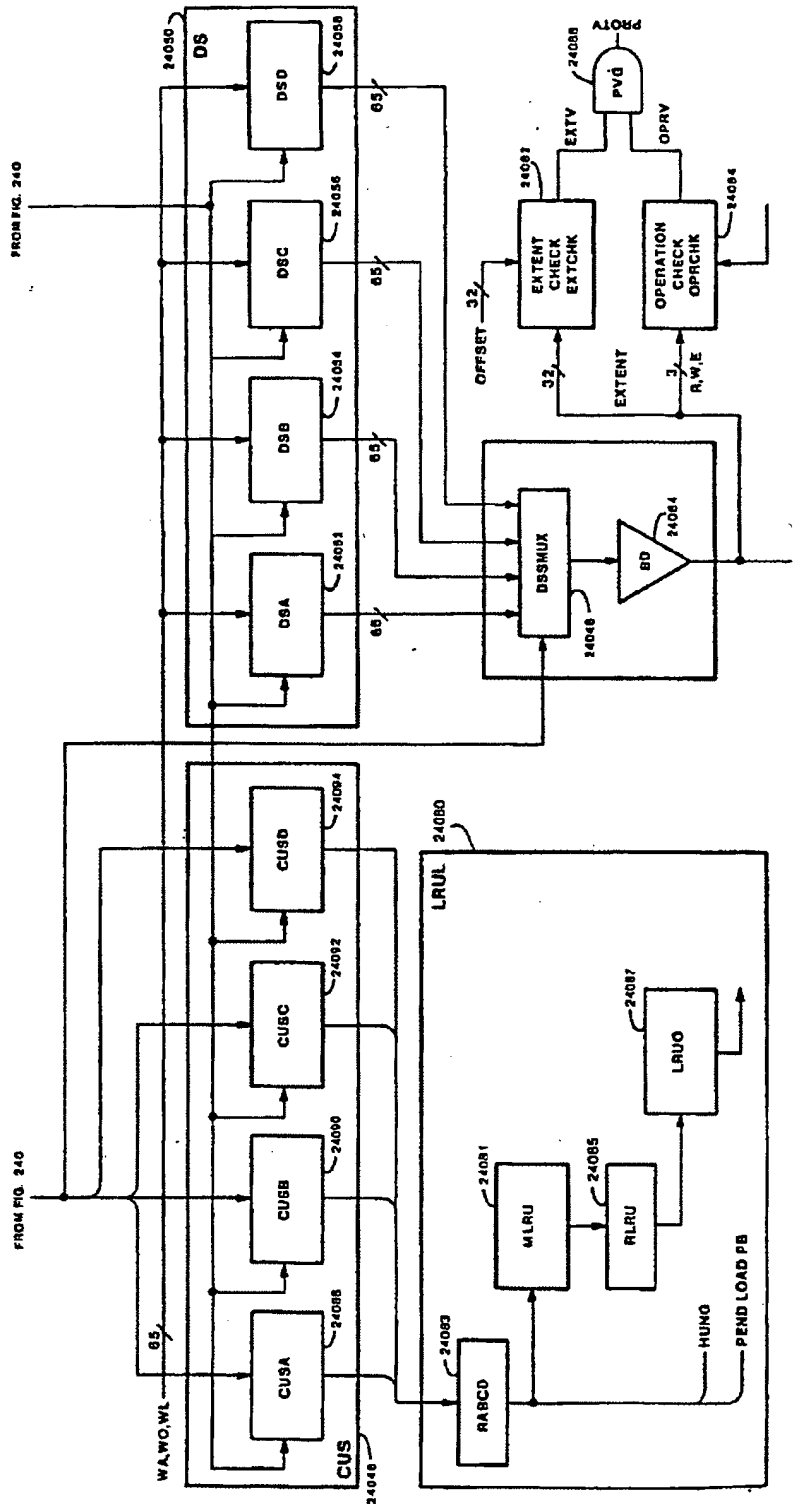


FIG. 238





NC 10228

FIG. 240A

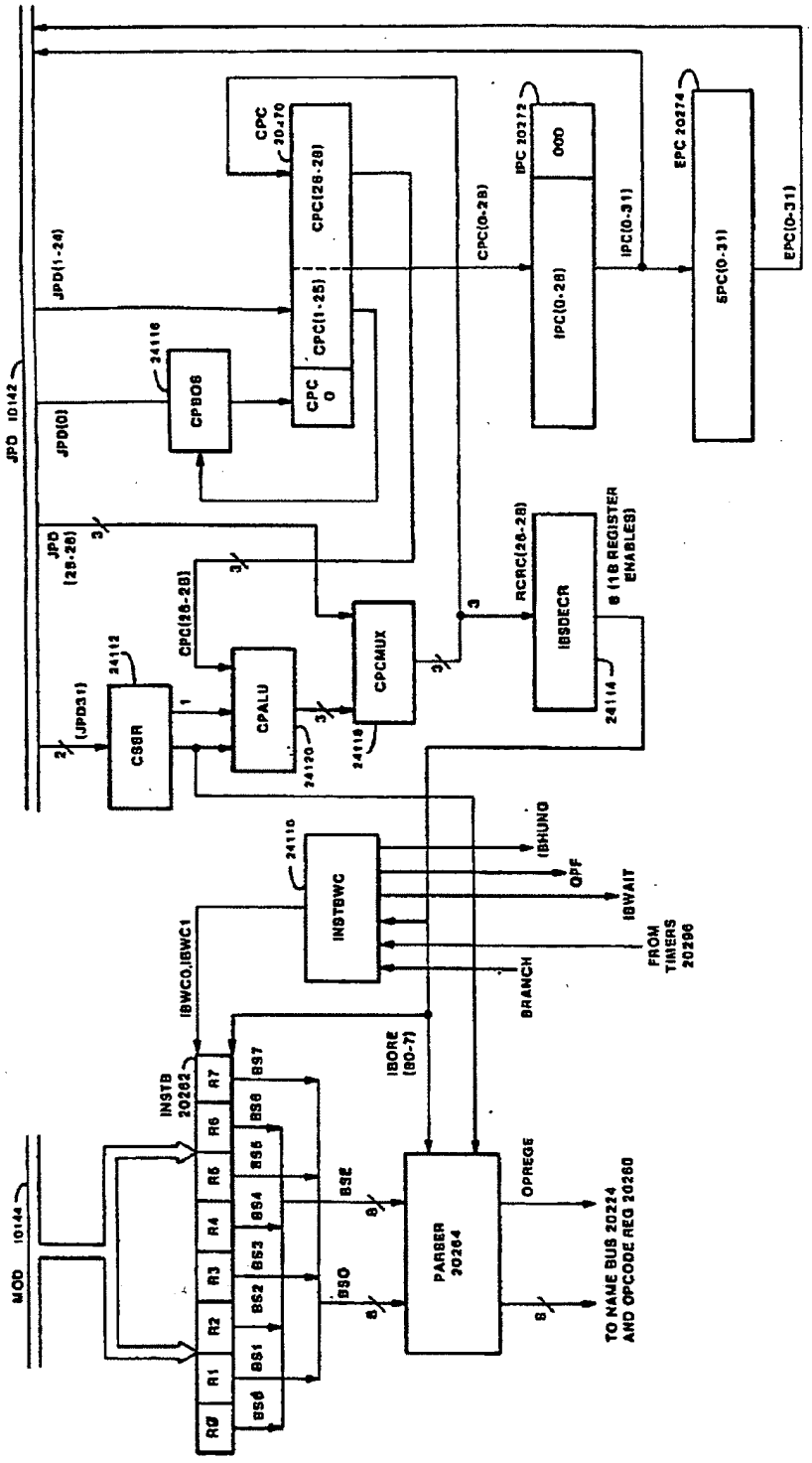


FIG. 241

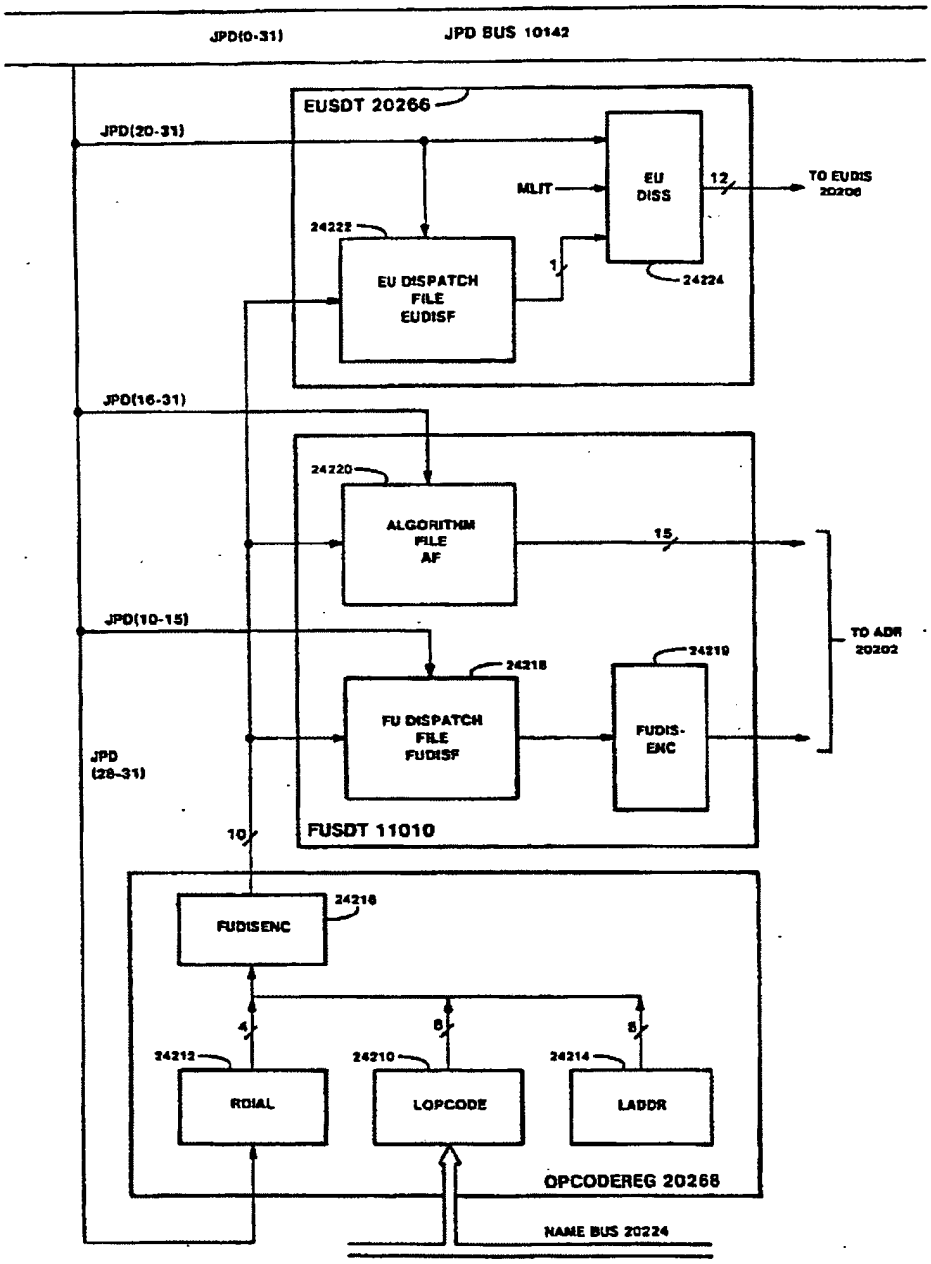


FIG. 242

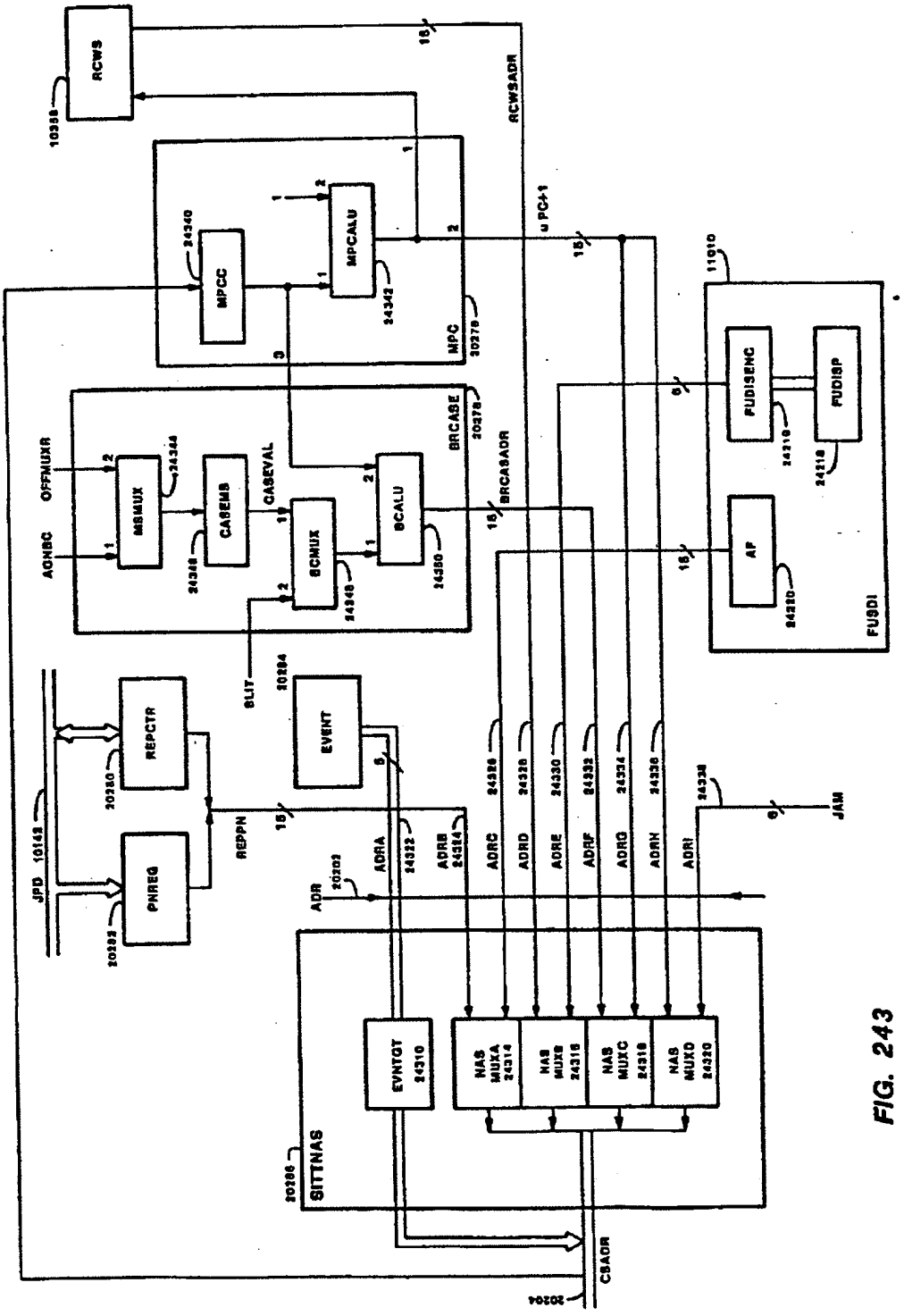


FIG. 243

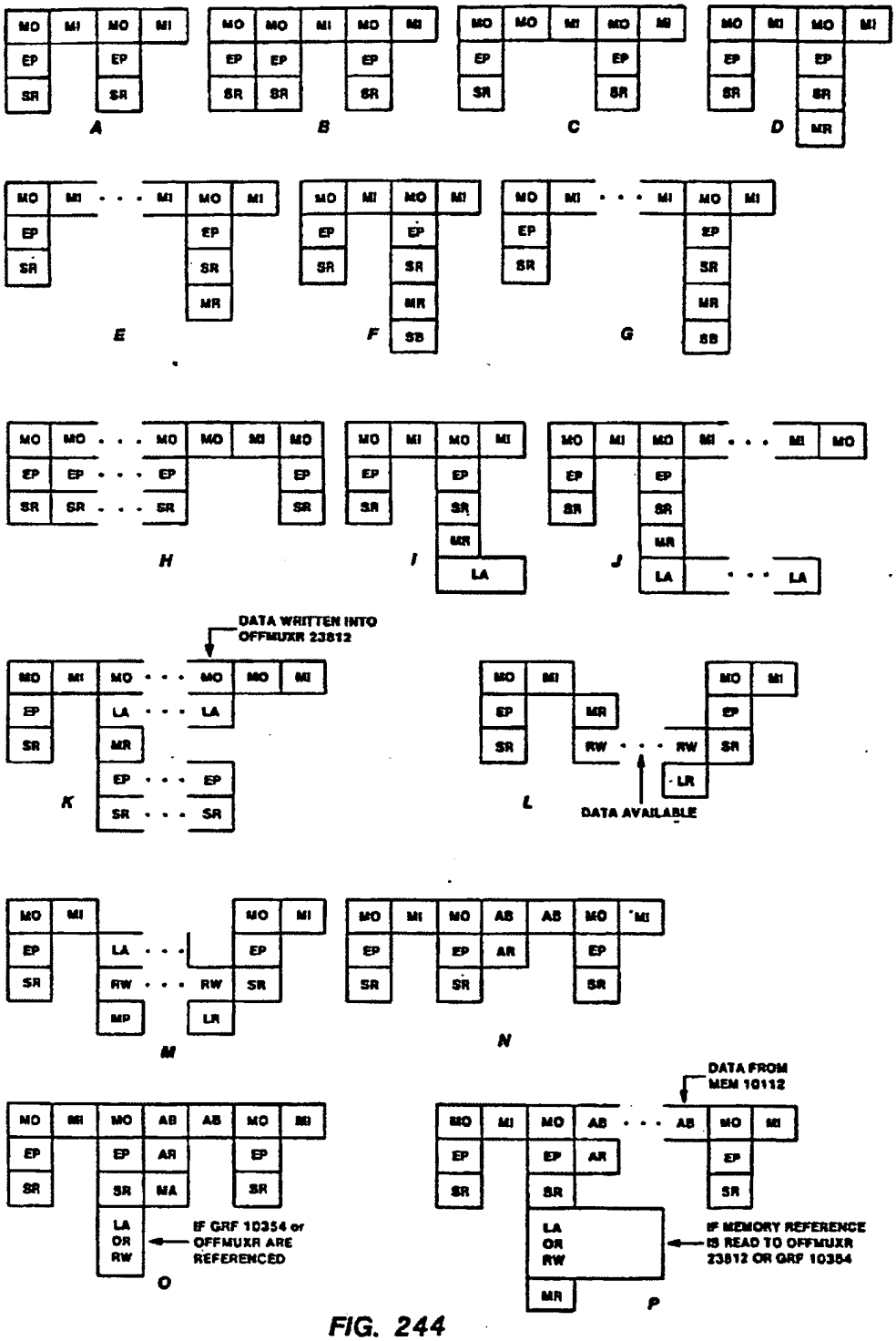


FIG. 244

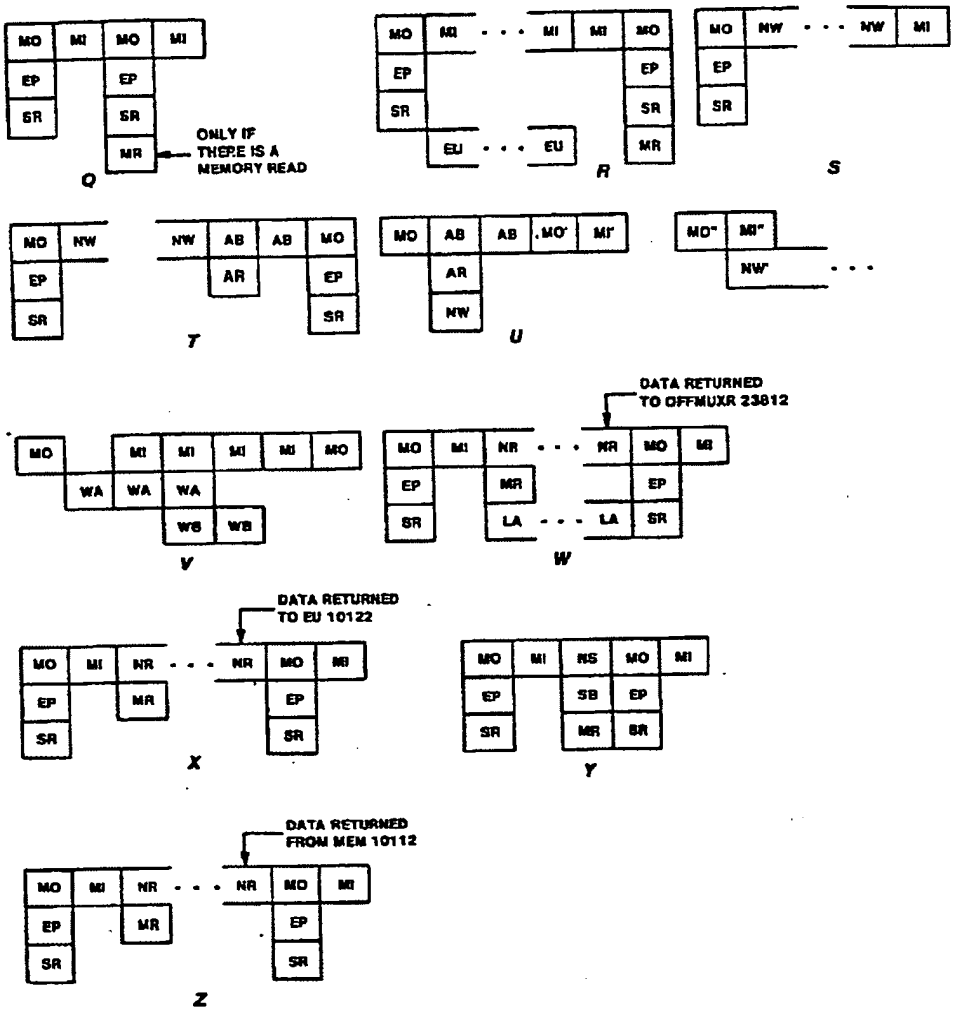


FIG. 244A

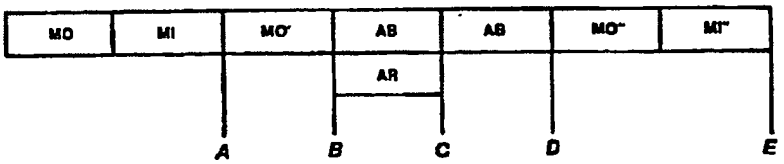


FIG. 245

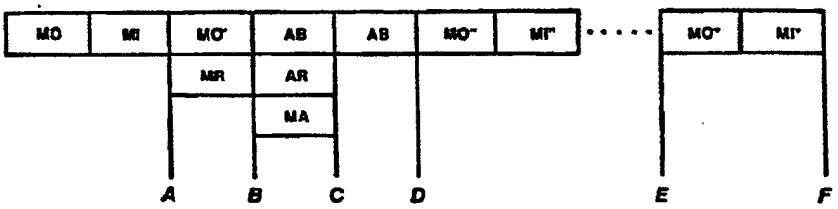


FIG. 246

PRIORITY LEVEL	EVENT	MASKED BY
0	E-UNIT STACK OVERFLOW	NONE
1	FATAL MEMORY ERROR	NONE
2	POWER FAIL	I
3	F-BOX STACK OVERFLOW	M,T,I
4	ILLEGAL E-UNIT DISPATCH (GATE FAULT)	NONE
5	STOREBACK EXCEPTION	MCWD
6	NAME TRACE TRAP	T,I
7	LOGICAL READ TRACE TRAP	T,I AND DES
8	LOGICAL WRITE TRACE TRAP	T,I AND DES
9	UID READ DEREFERENCE TRAP	DES
10	UID WRITE DEREFERENCE TRAP	DES
11	PROTECTION CACHE MISS	NONE
12	PROTECTION VIOLATION	MCWD
13	PAGE CROSSING INTERRUPT	NONE
14	LAT	NONE
15	WRITE LAT	NONE
16	MEMORY REFERENCE REPEAT	NONE
17	EGG TIMER OVERFLOW	A,M,T,I
18	E-BOX STACK UNDERFLOW	A,M,T,I
19	NON-FATAL MEMORY ERROR	NONE
20	INTERVAL TIMER OVEREFLOW	NONE
21	IPM INTERRUPT	NONE
22	S-OP TRACE TRAP	NONE
23	ILLEGAL S-OP	NONE
24	MICROINSTRUCTION TRACE TRAP	NONE
25	NON-PRESENT MICROINSTRUCTION	NONE
26	INSTRUCTION PREFETCH IS HUNG	NONE
27	F-BOX STACK UNDERFLOW	NONE
28	MICROINSTRUCTION BREAK POINT TRACE TRAP	T,I AND MCWD
29	MISS ON NAME CACHE LOAD OR READ REGISTER	NONE

FIG. 247

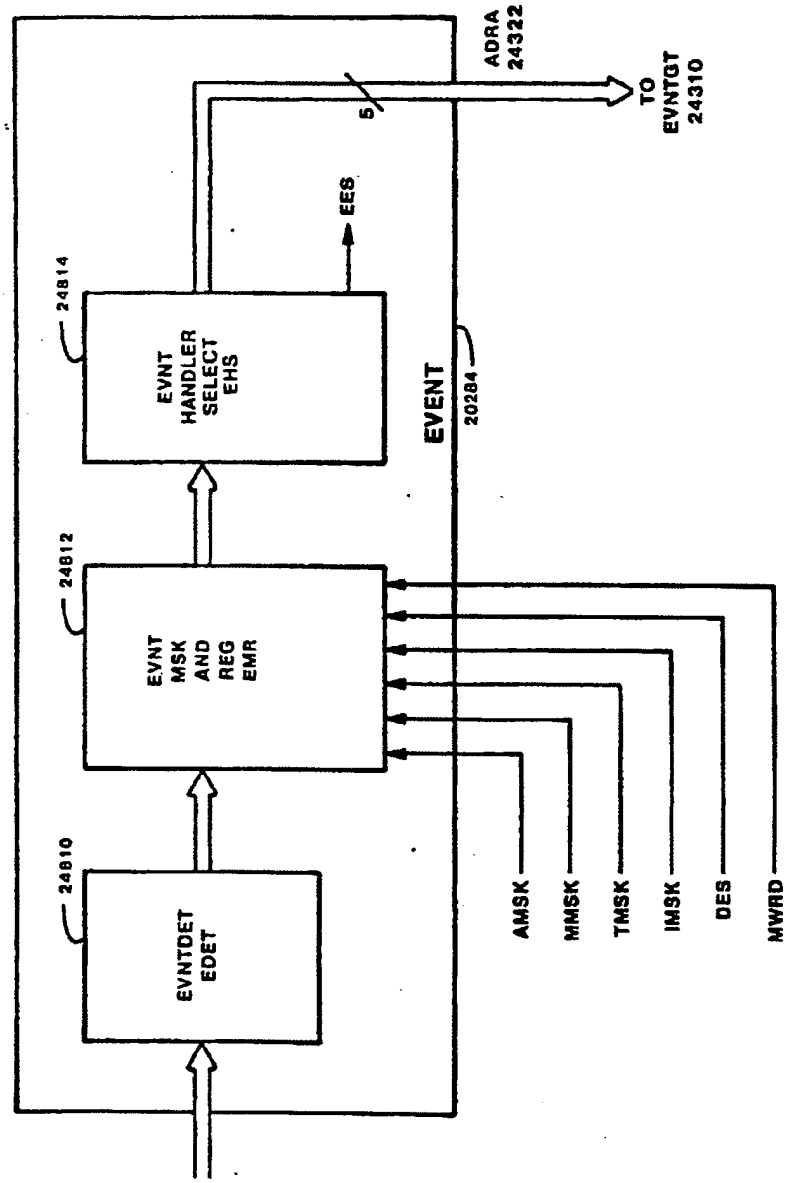


FIG. 248

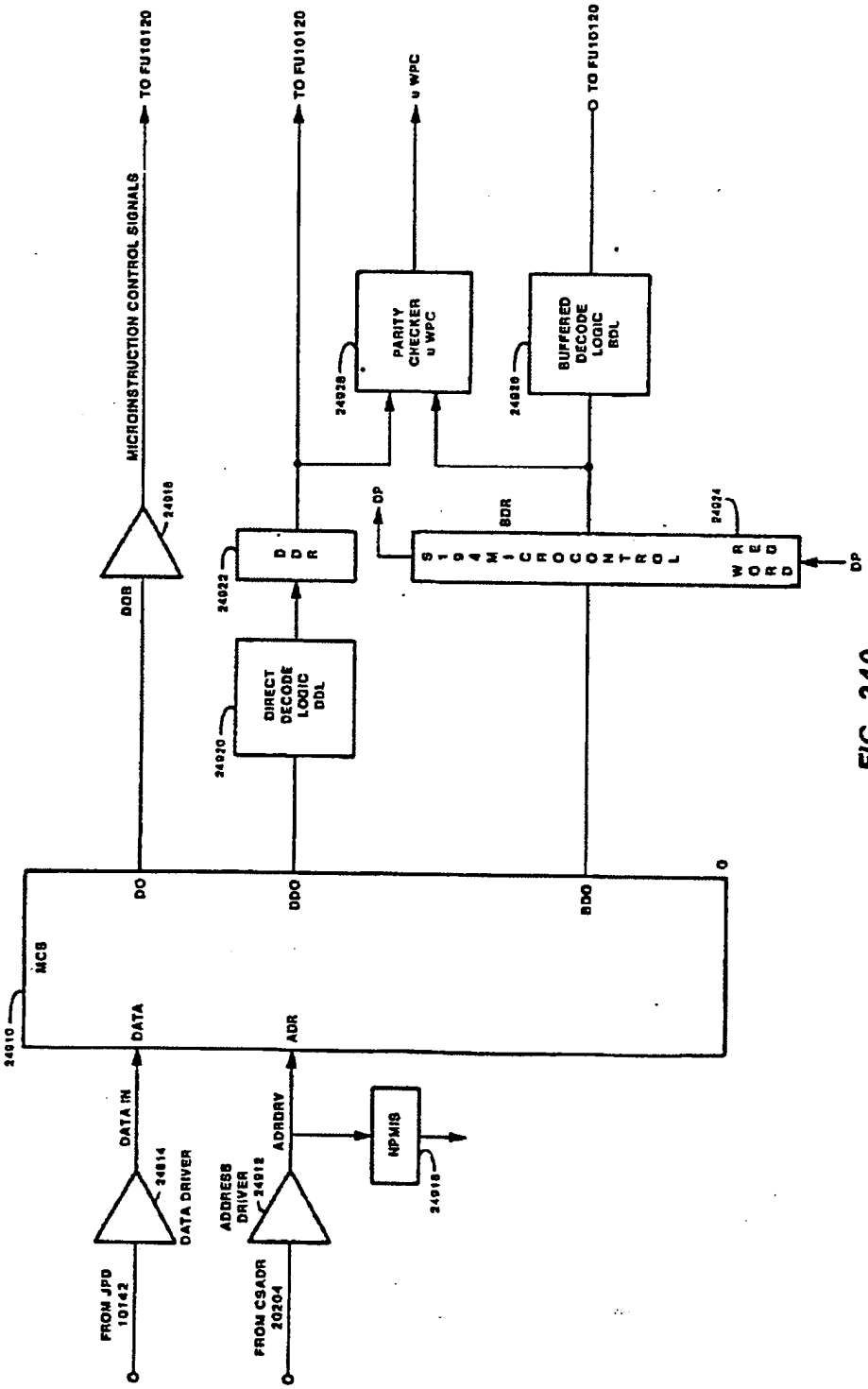


FIG. 249

FIG. 251

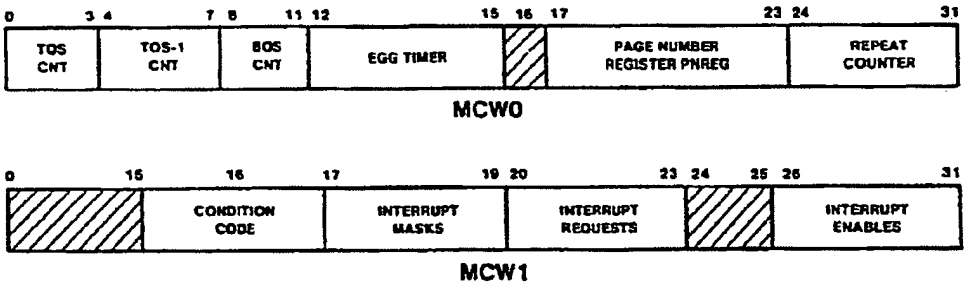


FIG. 252

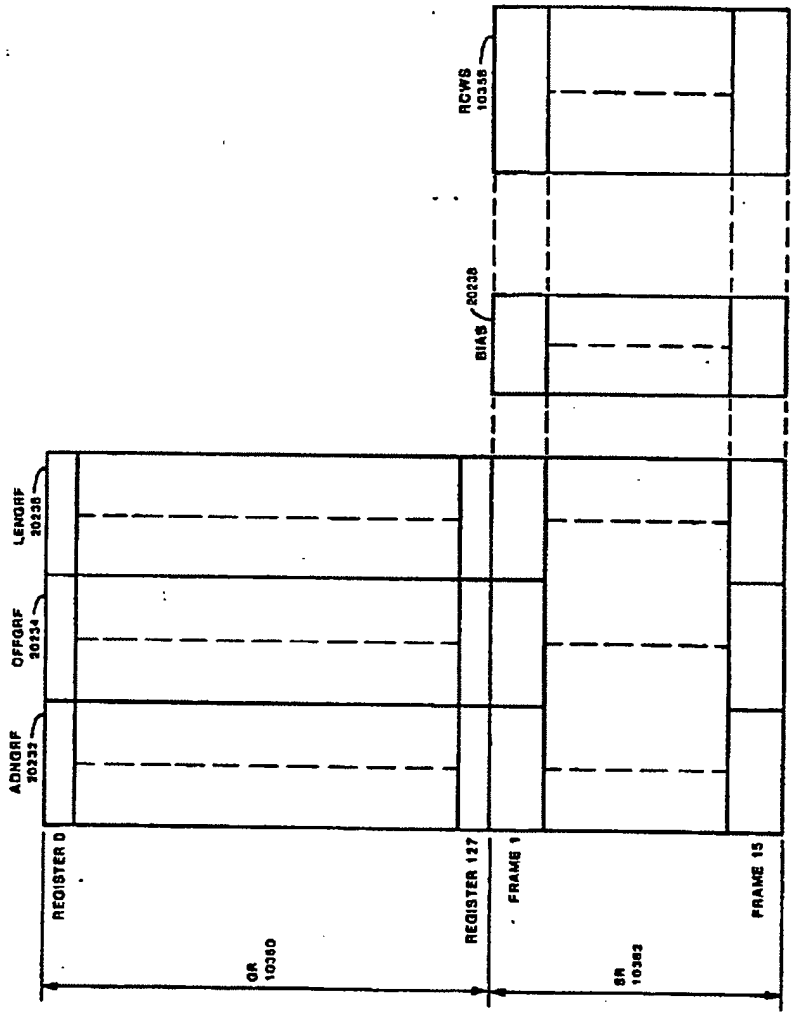


FIG. 253

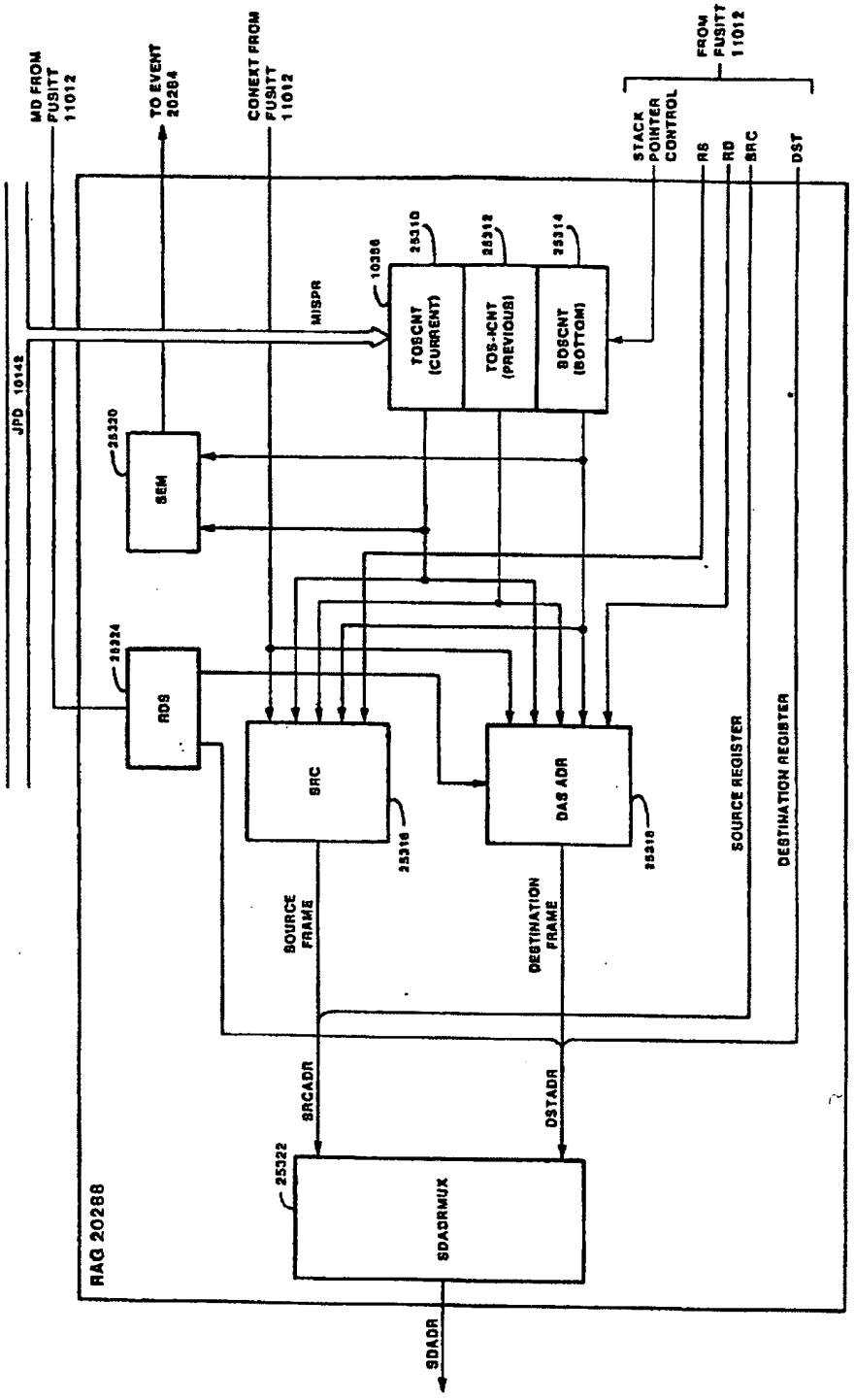


FIG. 253A

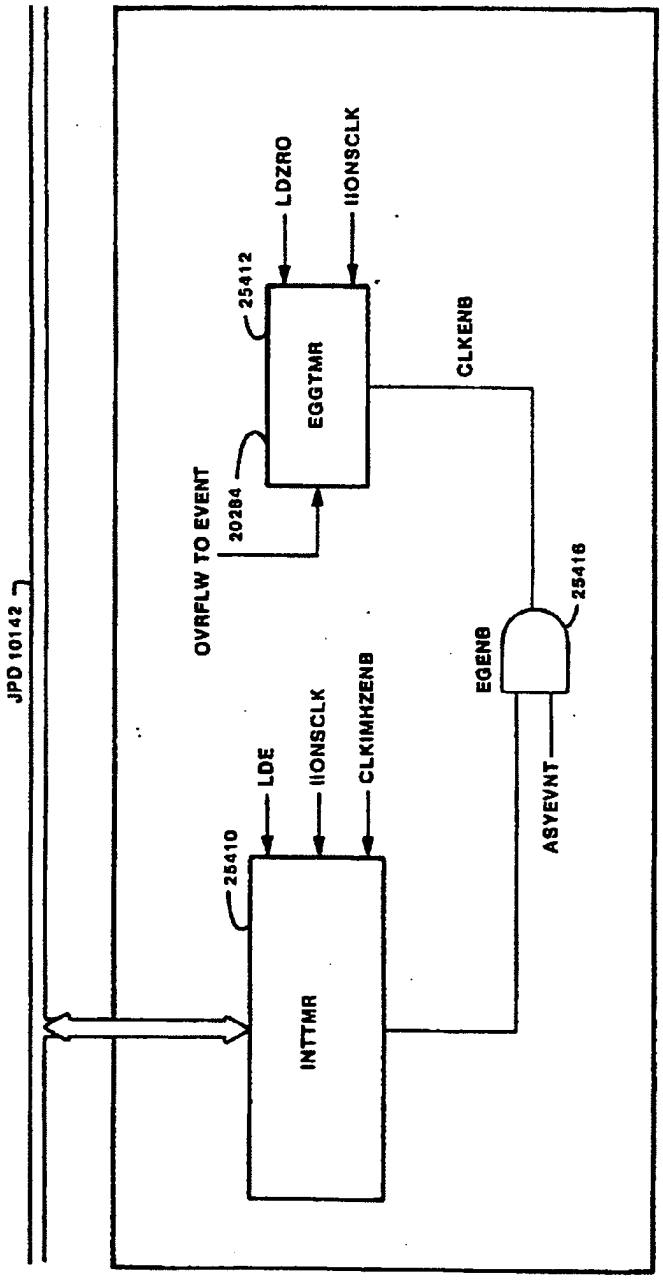


FIG. 254

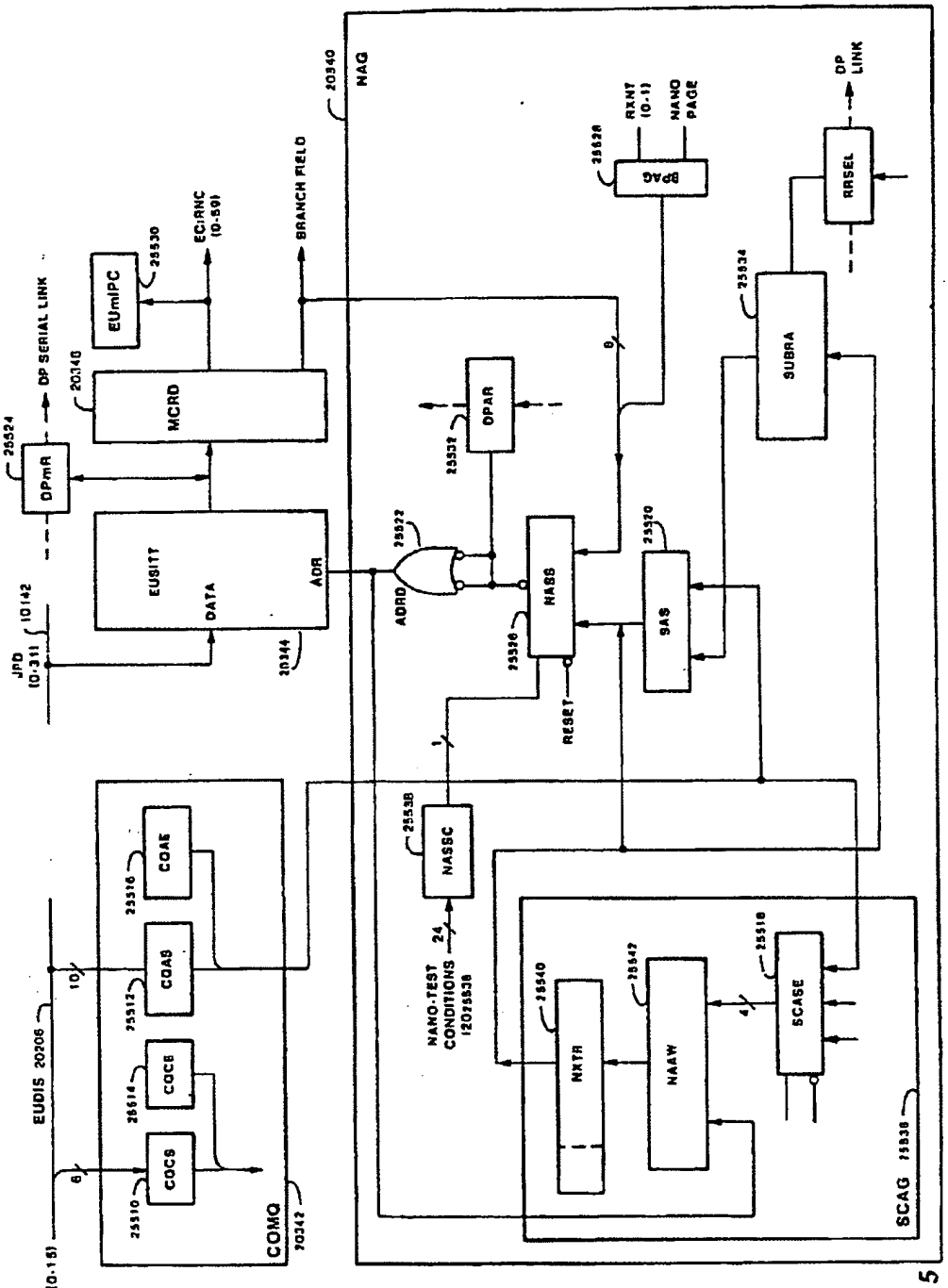


FIG. 255

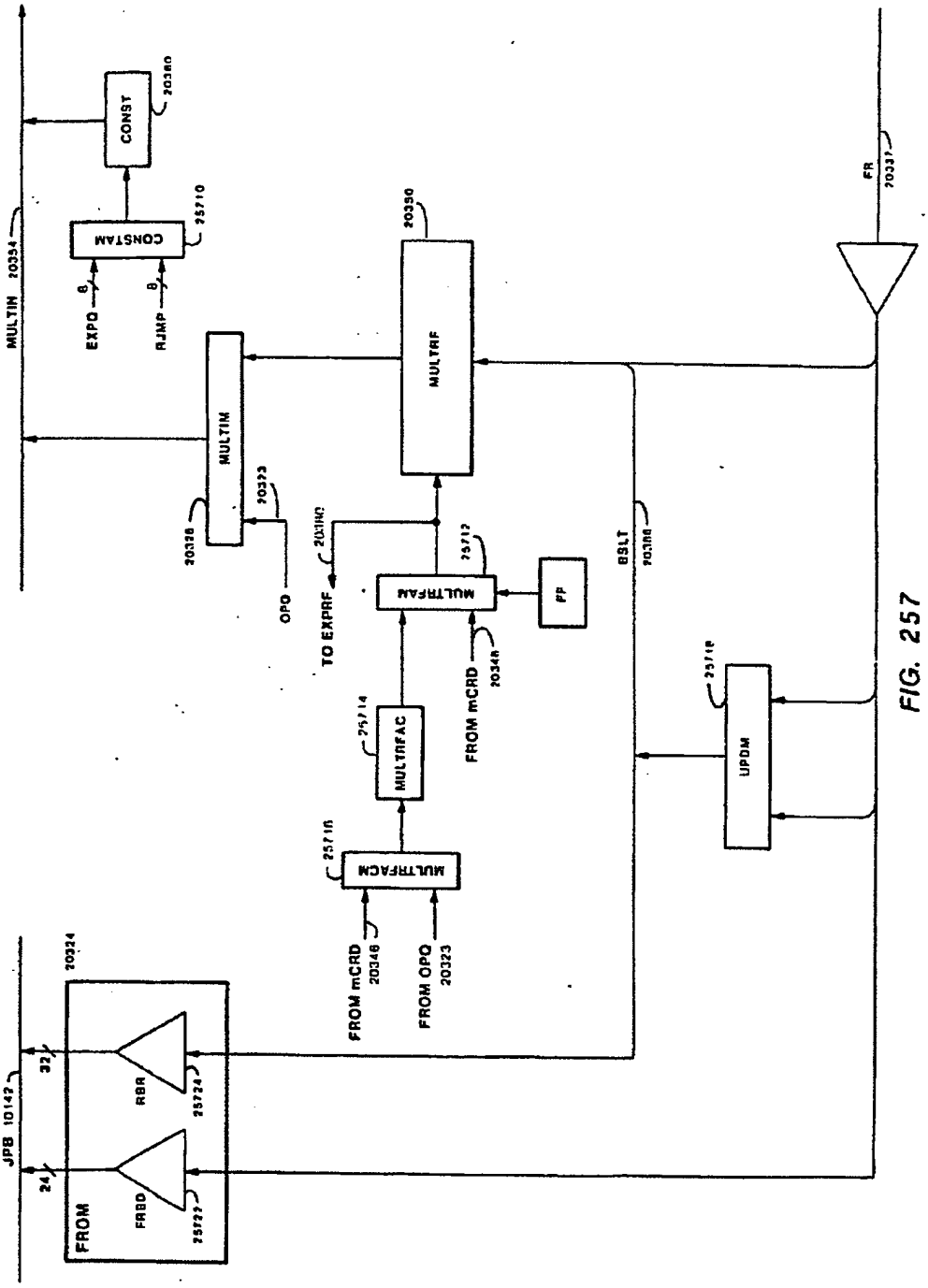


FIG. 257

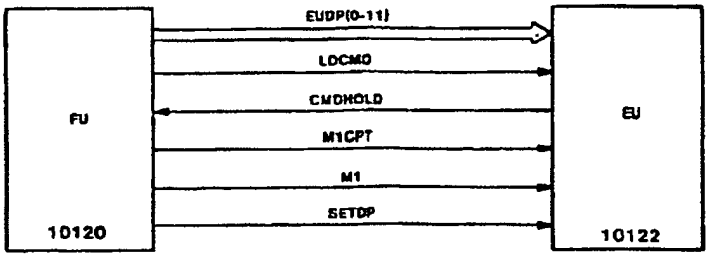


FIG. 260

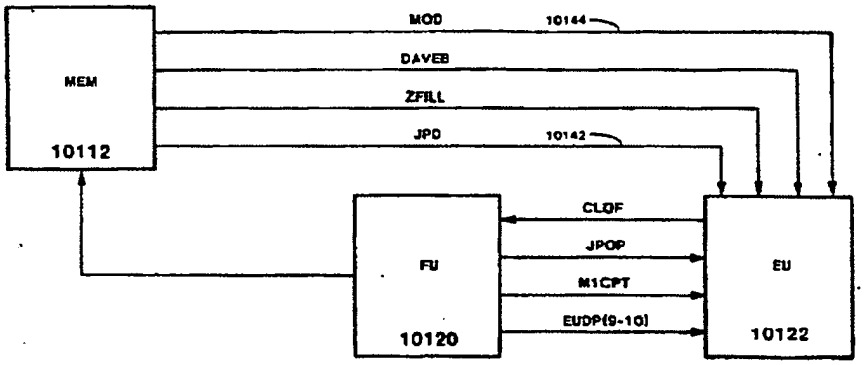


FIG. 261

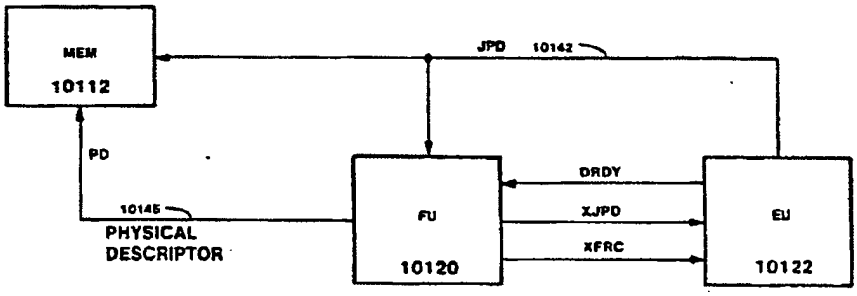


FIG. 262

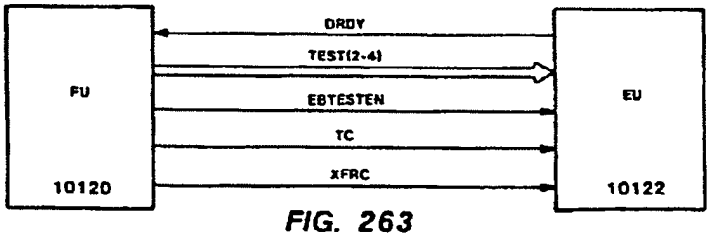


FIG. 263

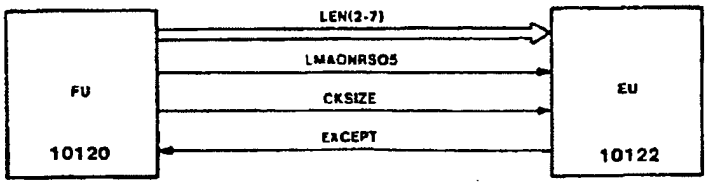


FIG. 264

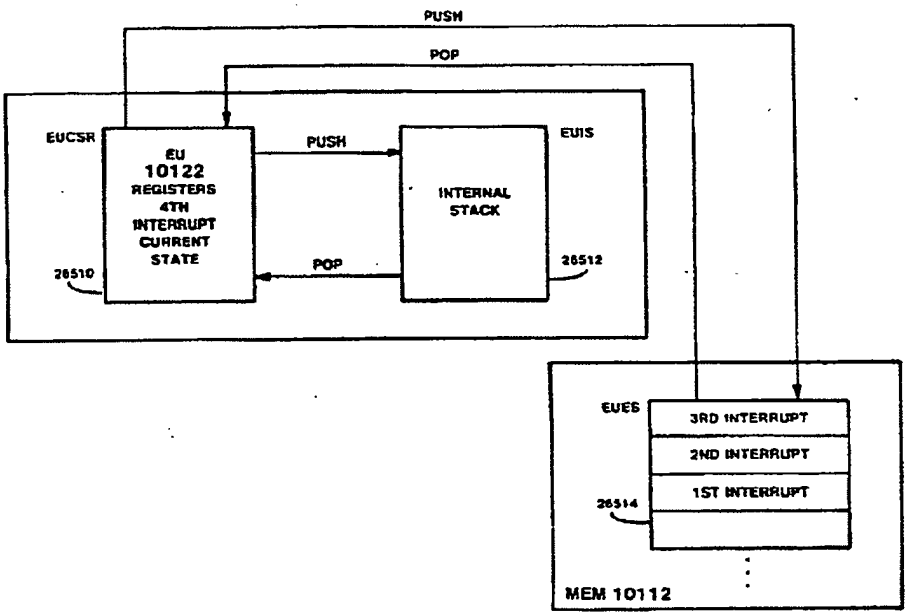


FIG. 265

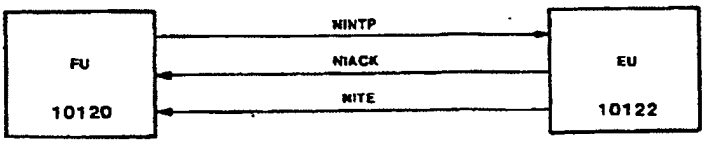


FIG. 266

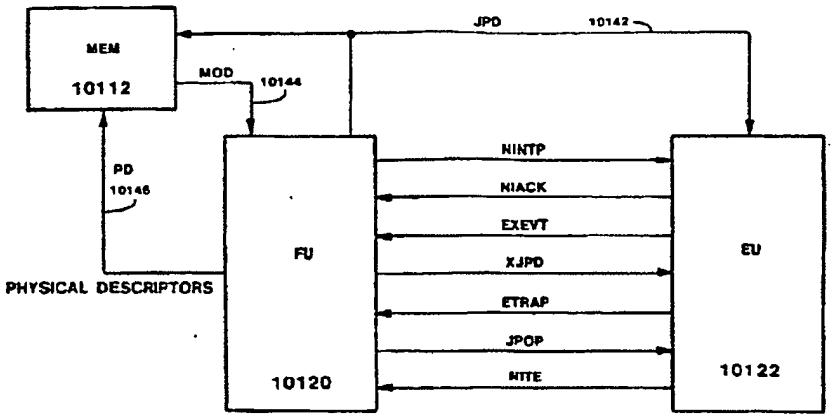


FIG. 267

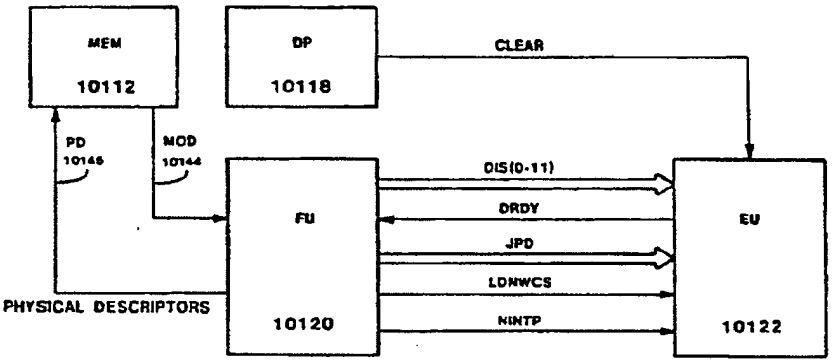


FIG. 268

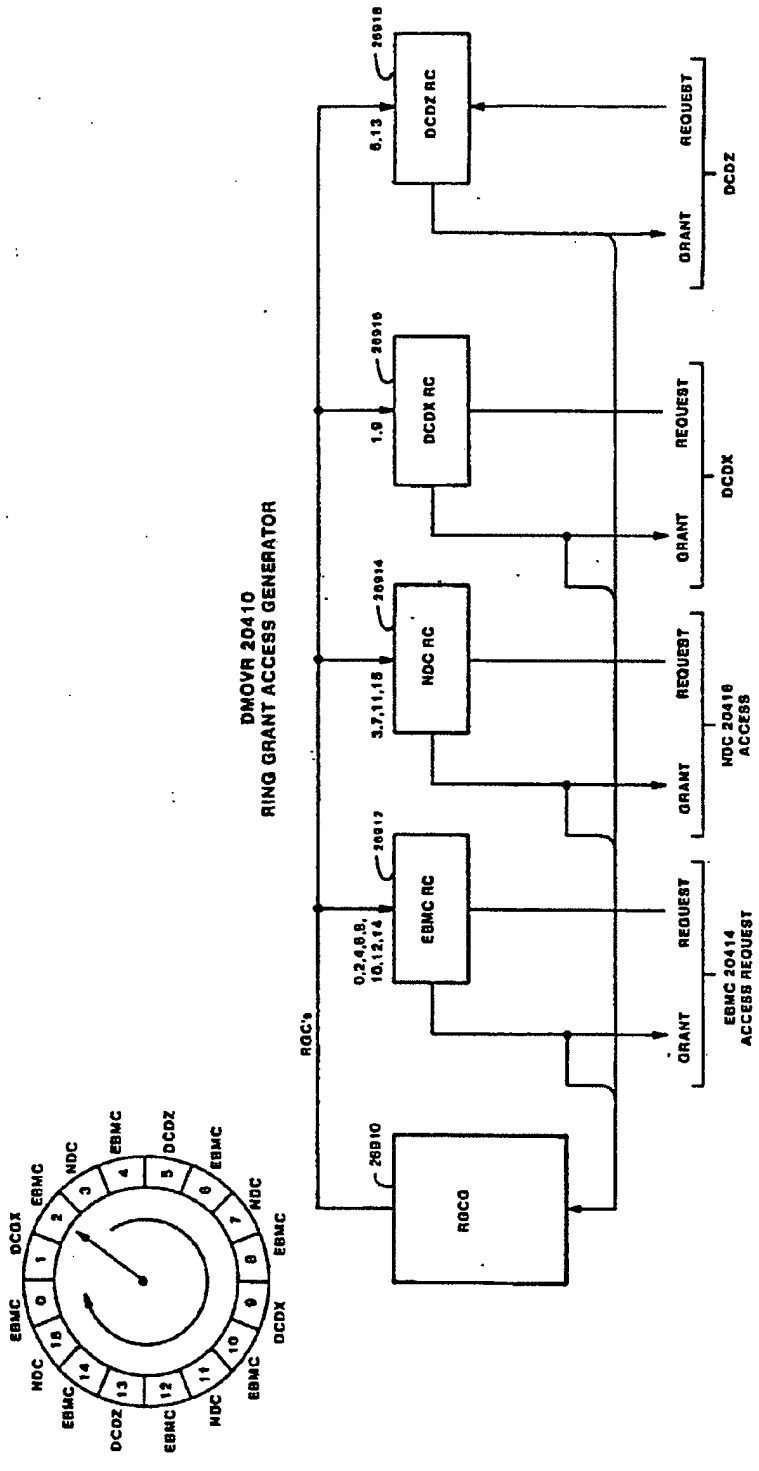


FIG 269

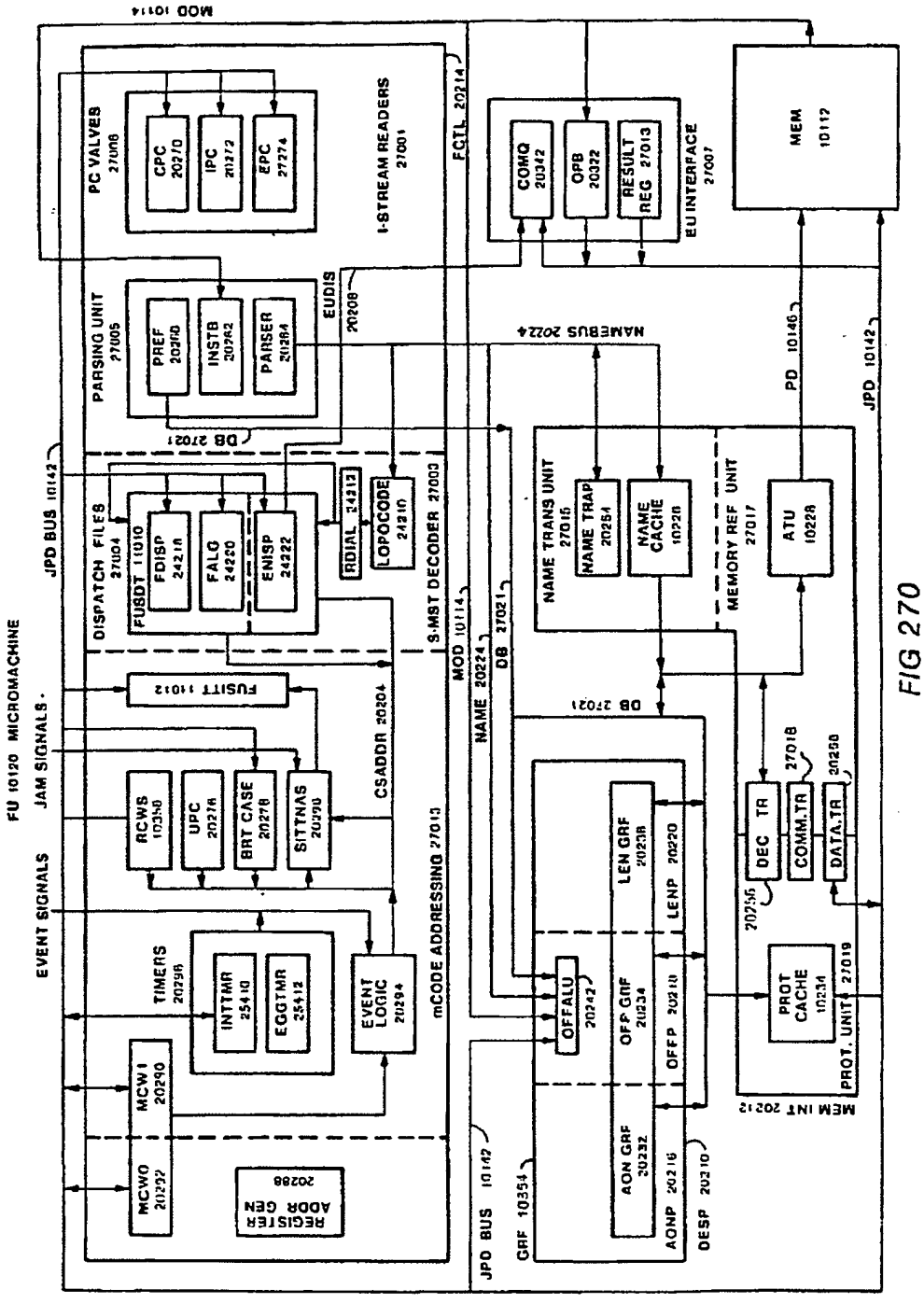


FIG 270

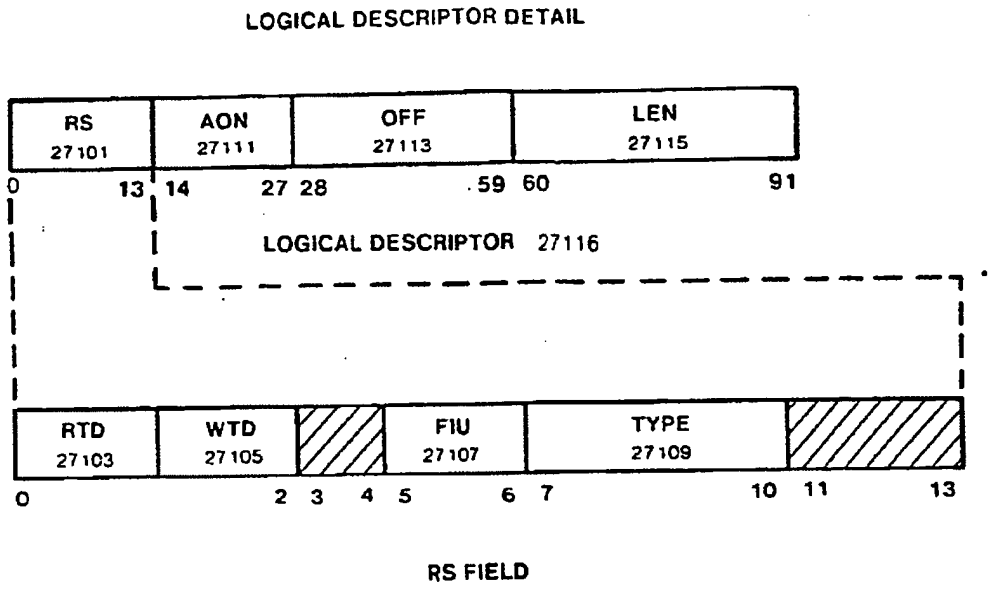


FIG. 271

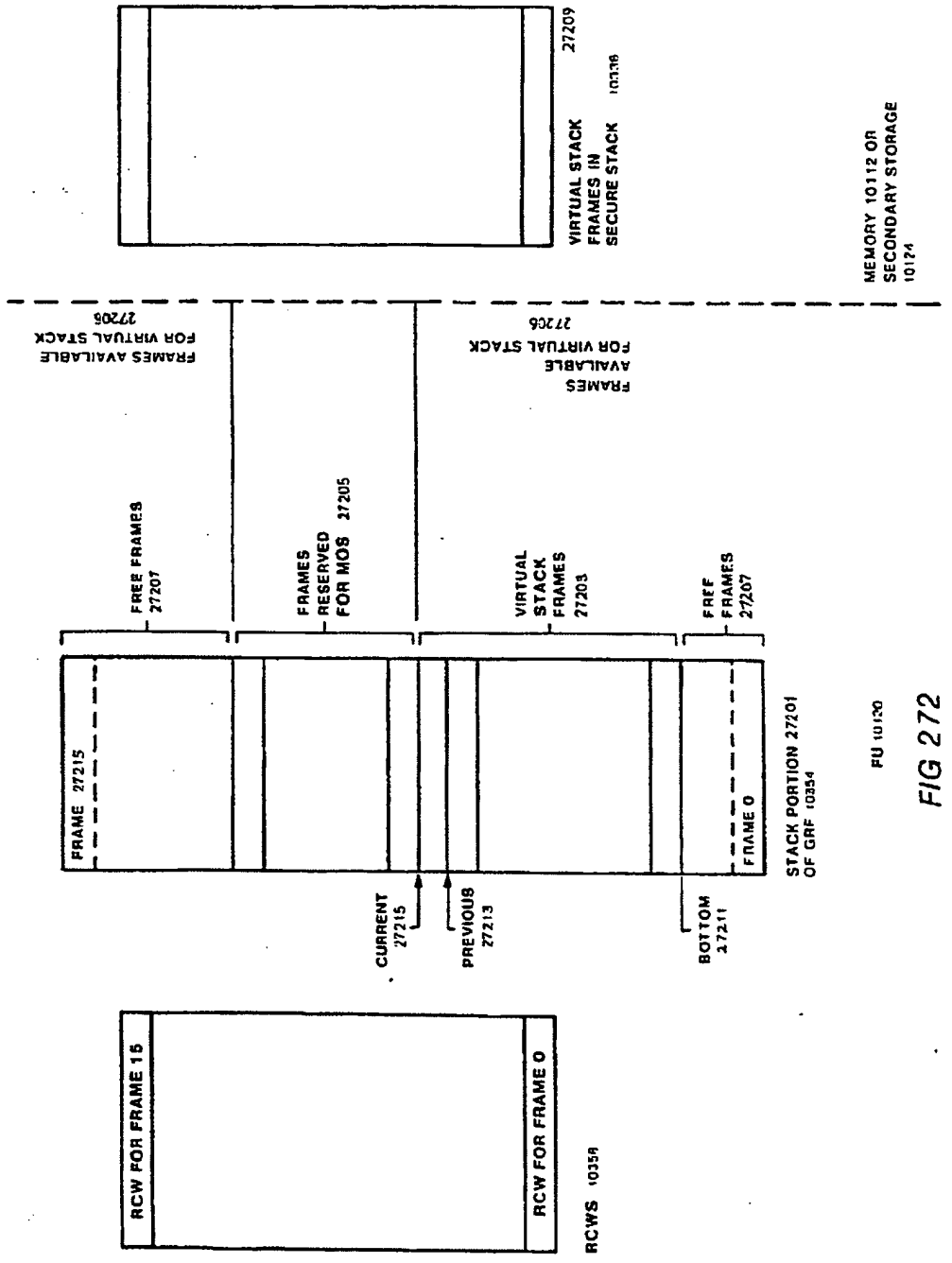


FIG 272

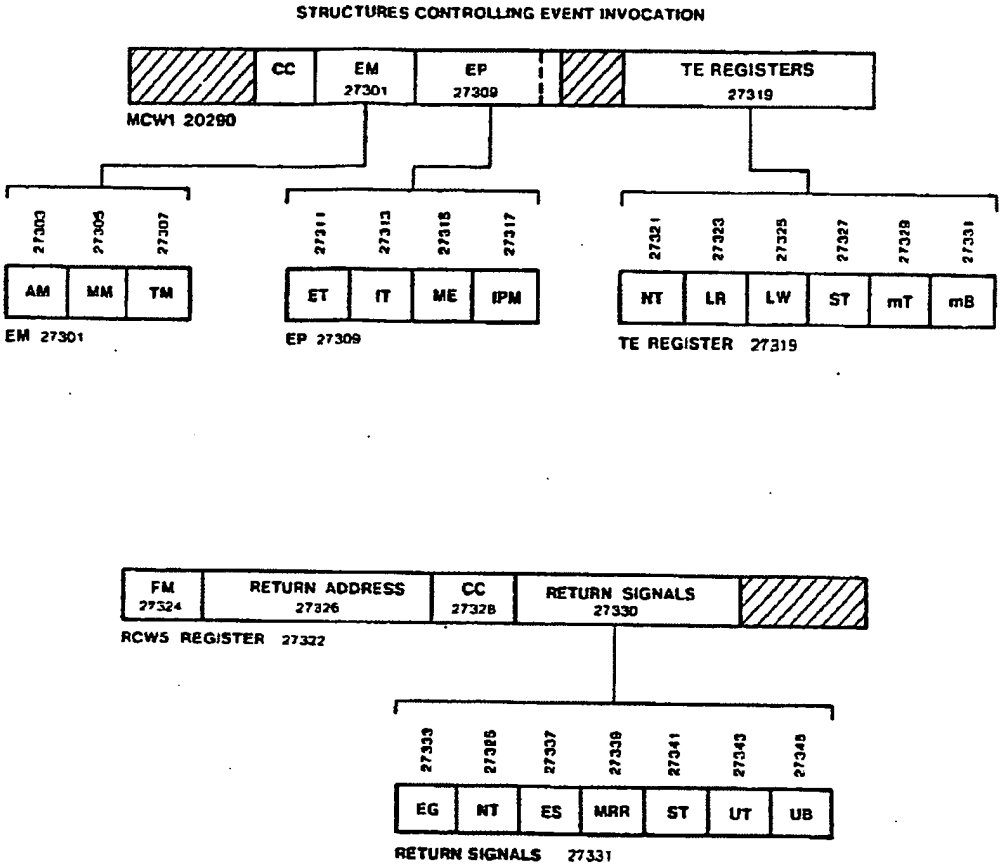


FIG. 273

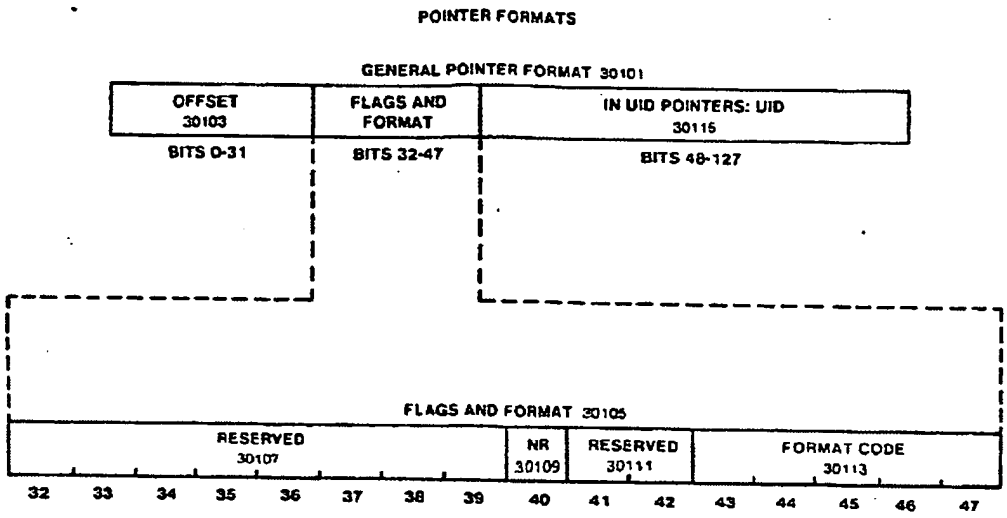


FIG. 301

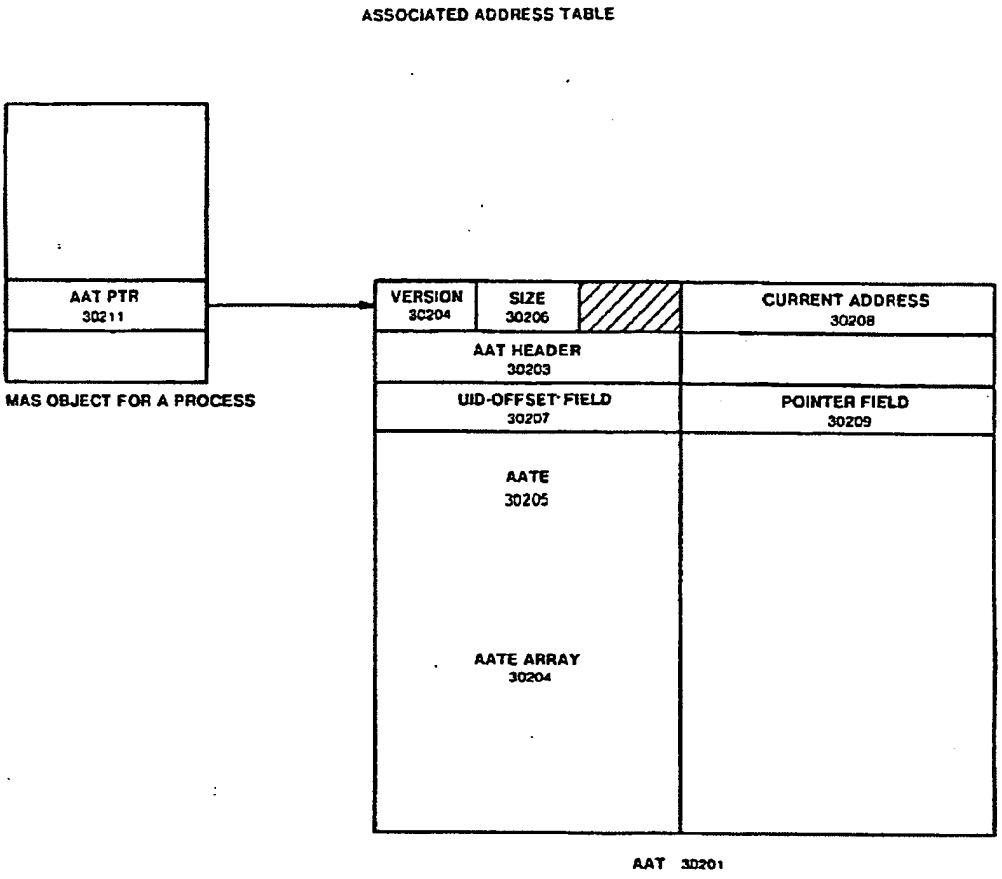


FIG. 302

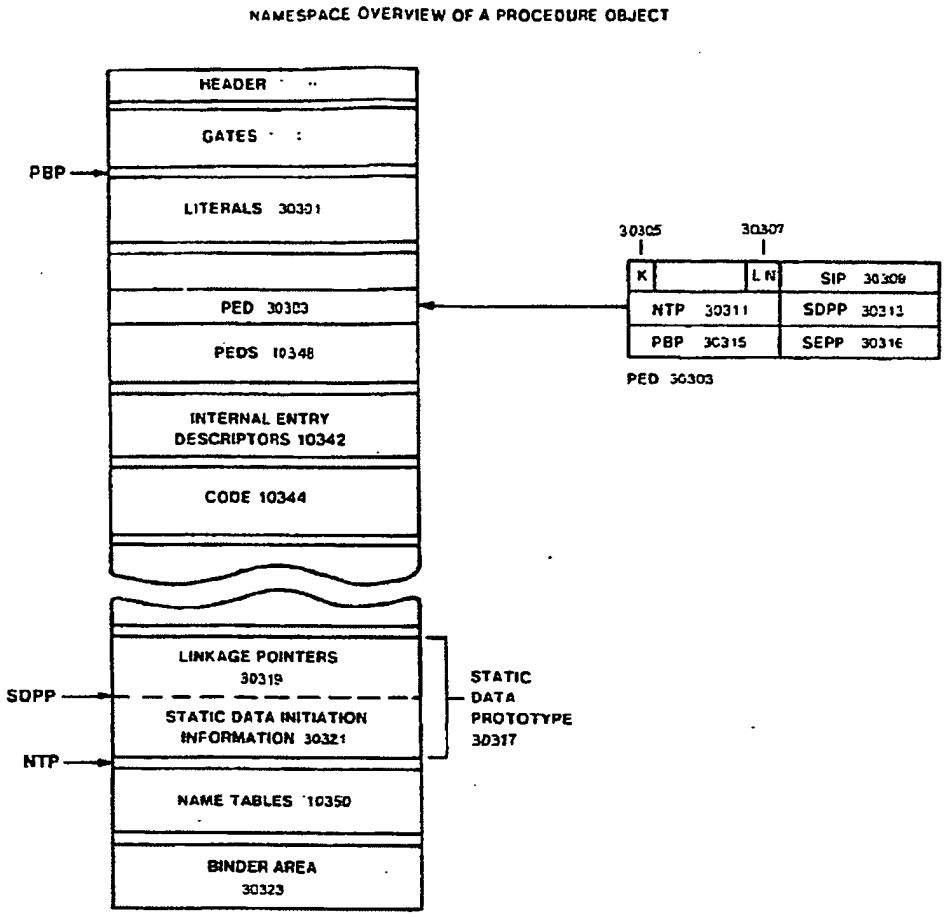


FIG. 303

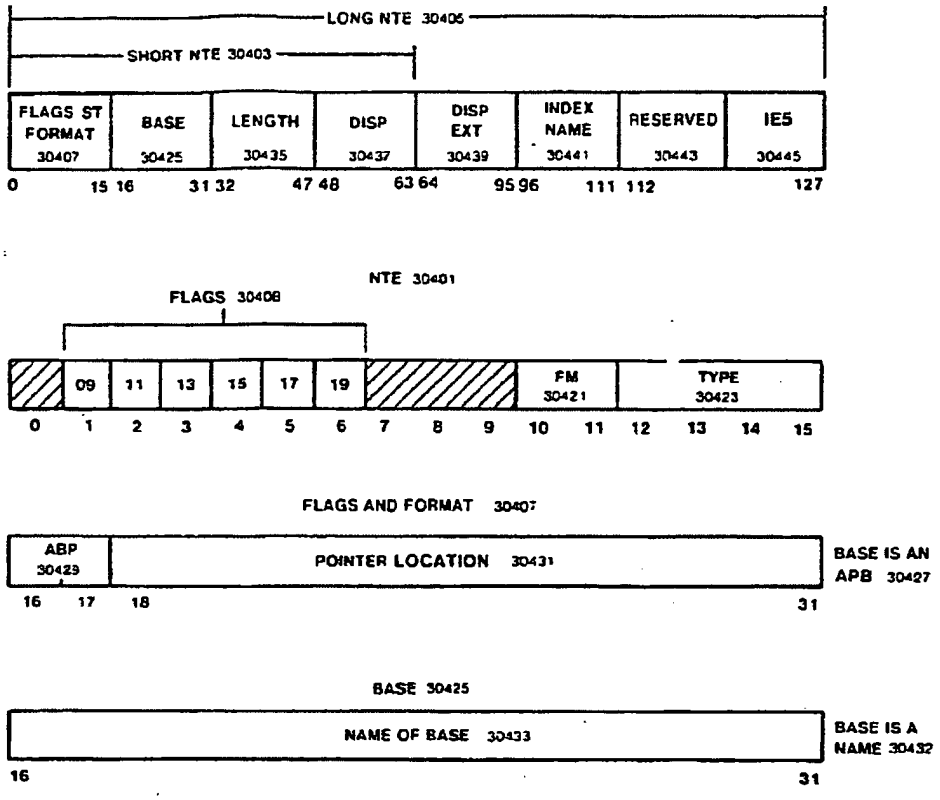


FIG. 304

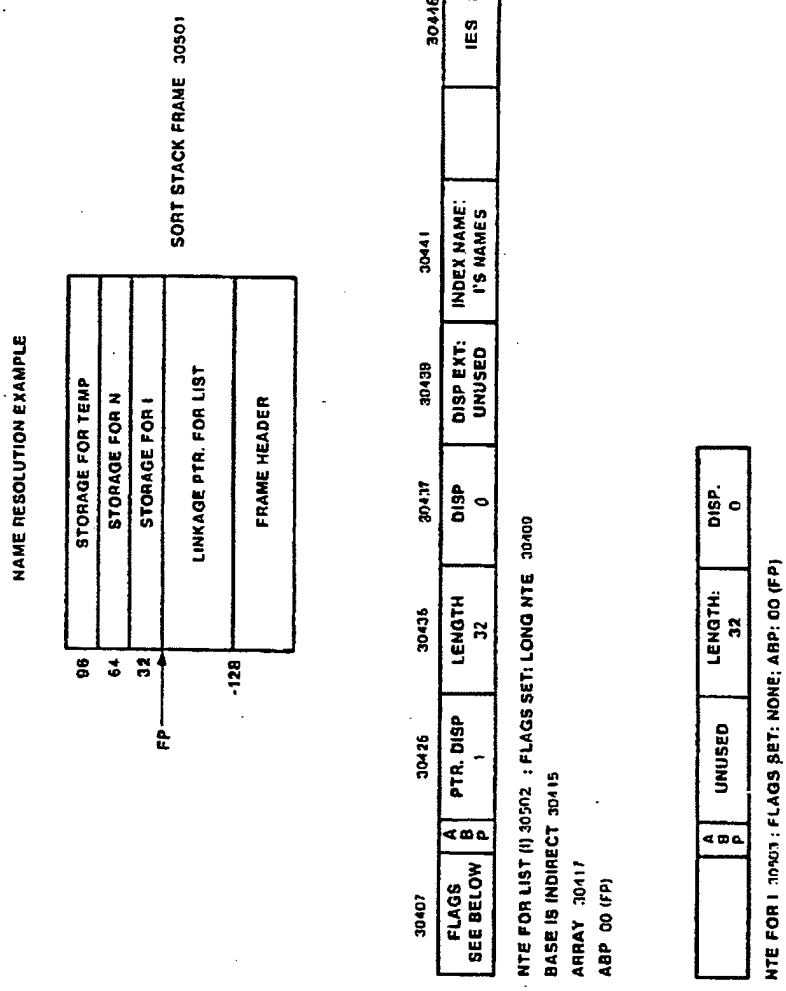


FIG. 305

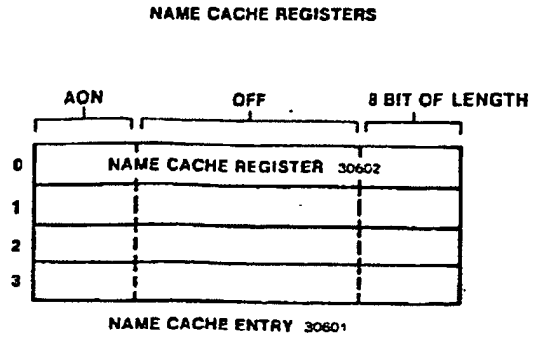


FIG. 306

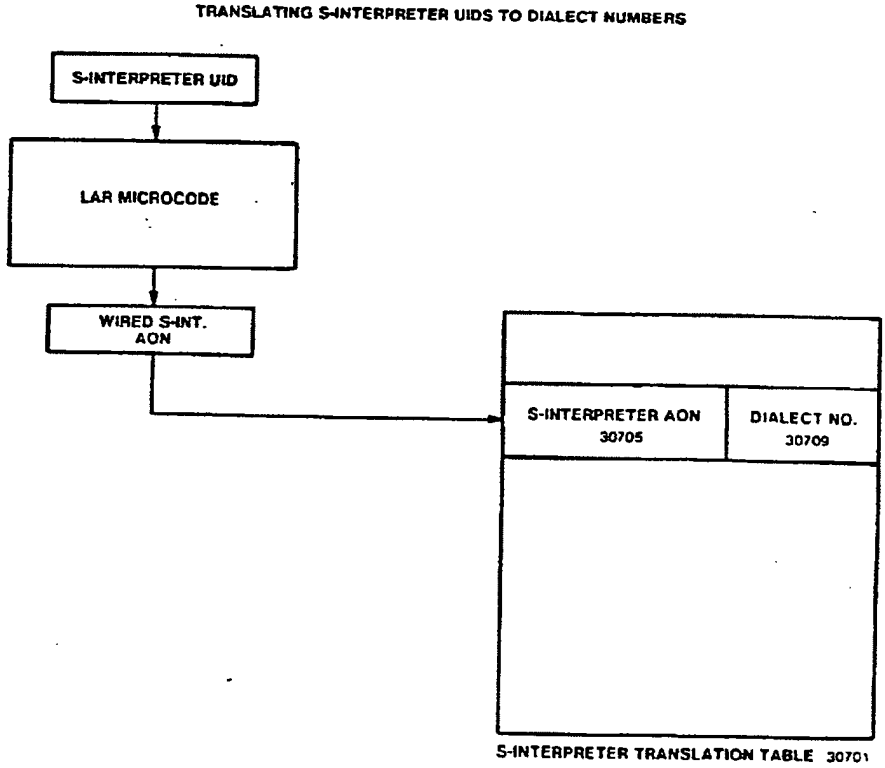


FIG. 307

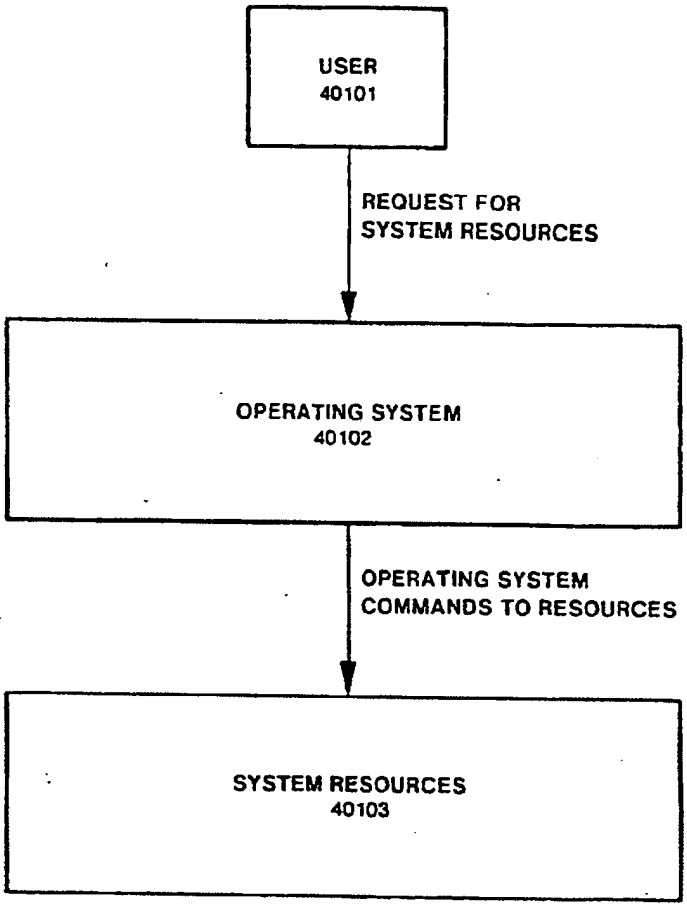


FIG 401

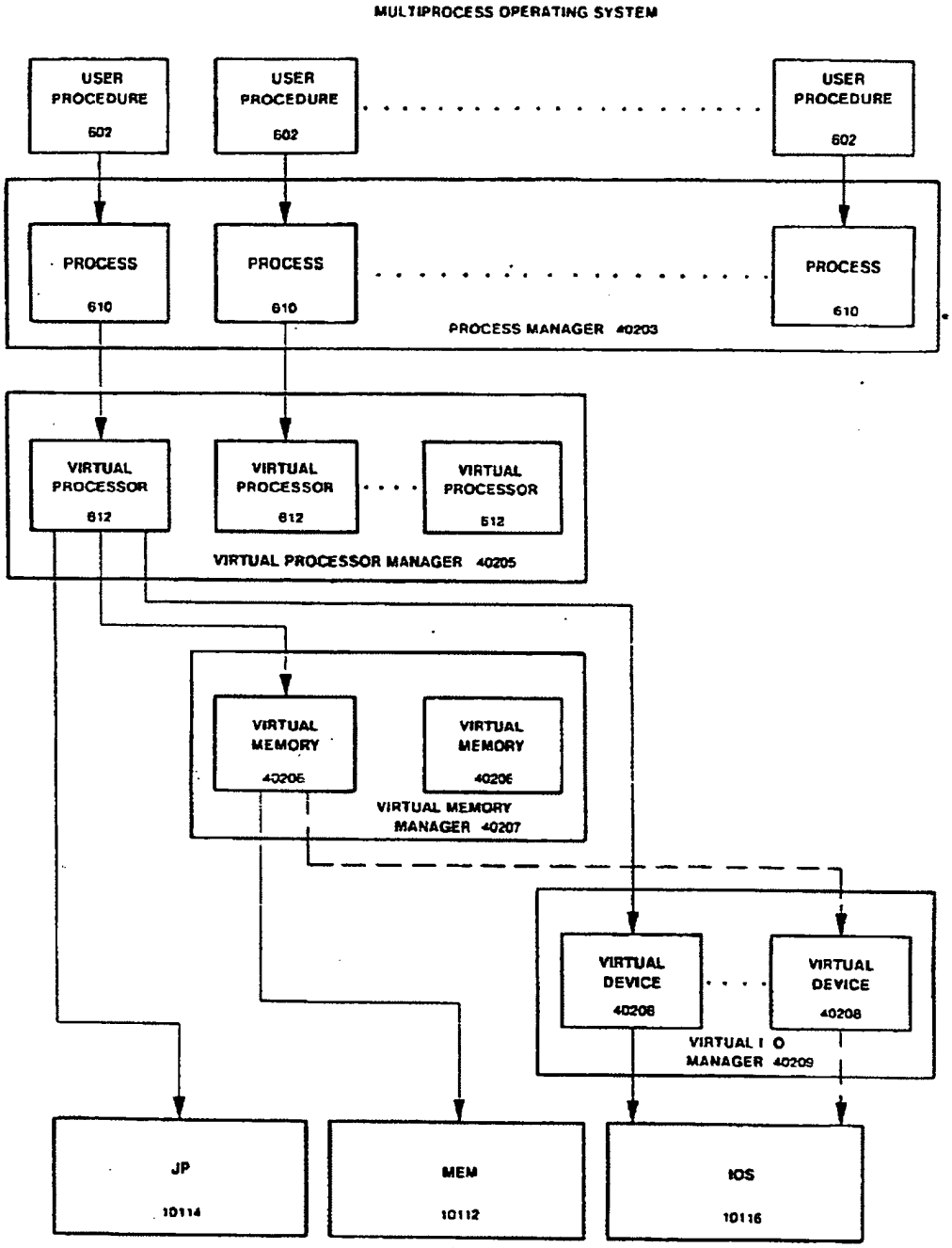


FIG 402

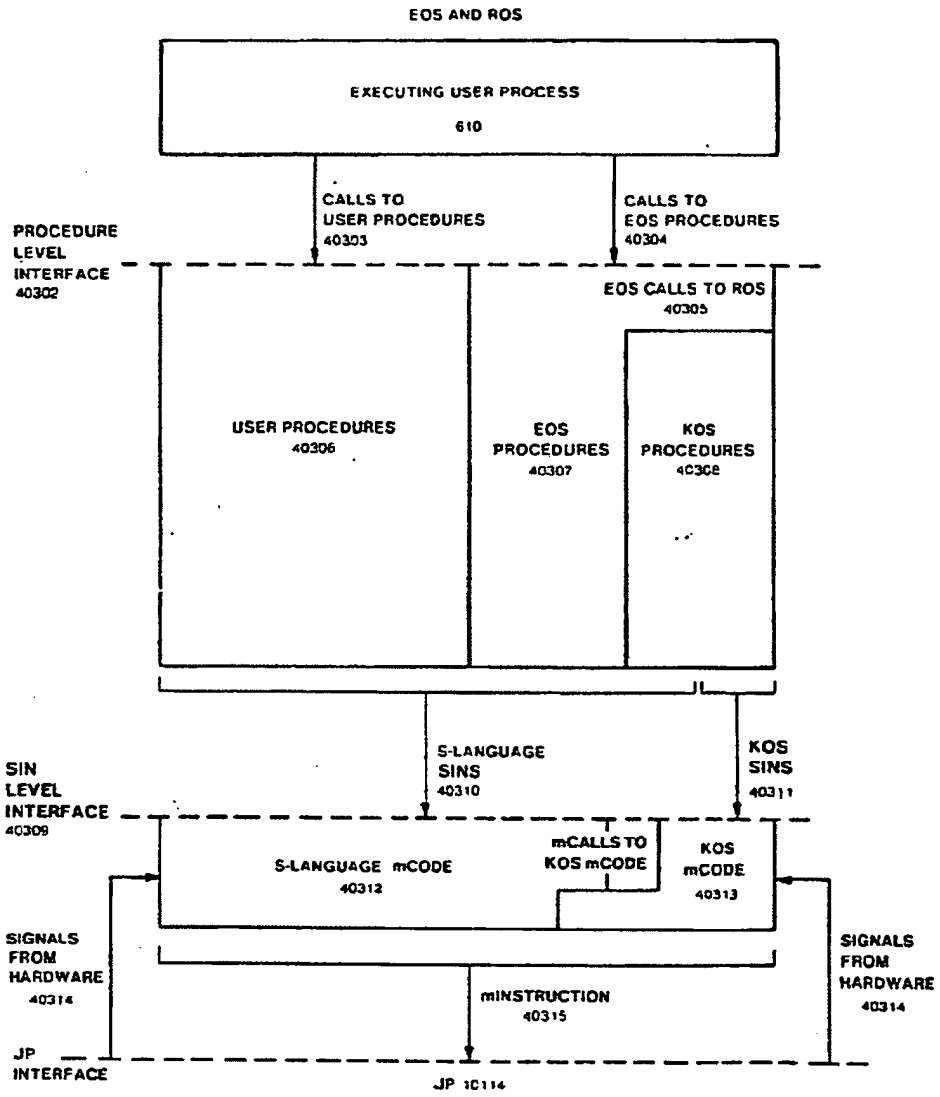


FIG. 403

EOS VIEW OF OBJECTS

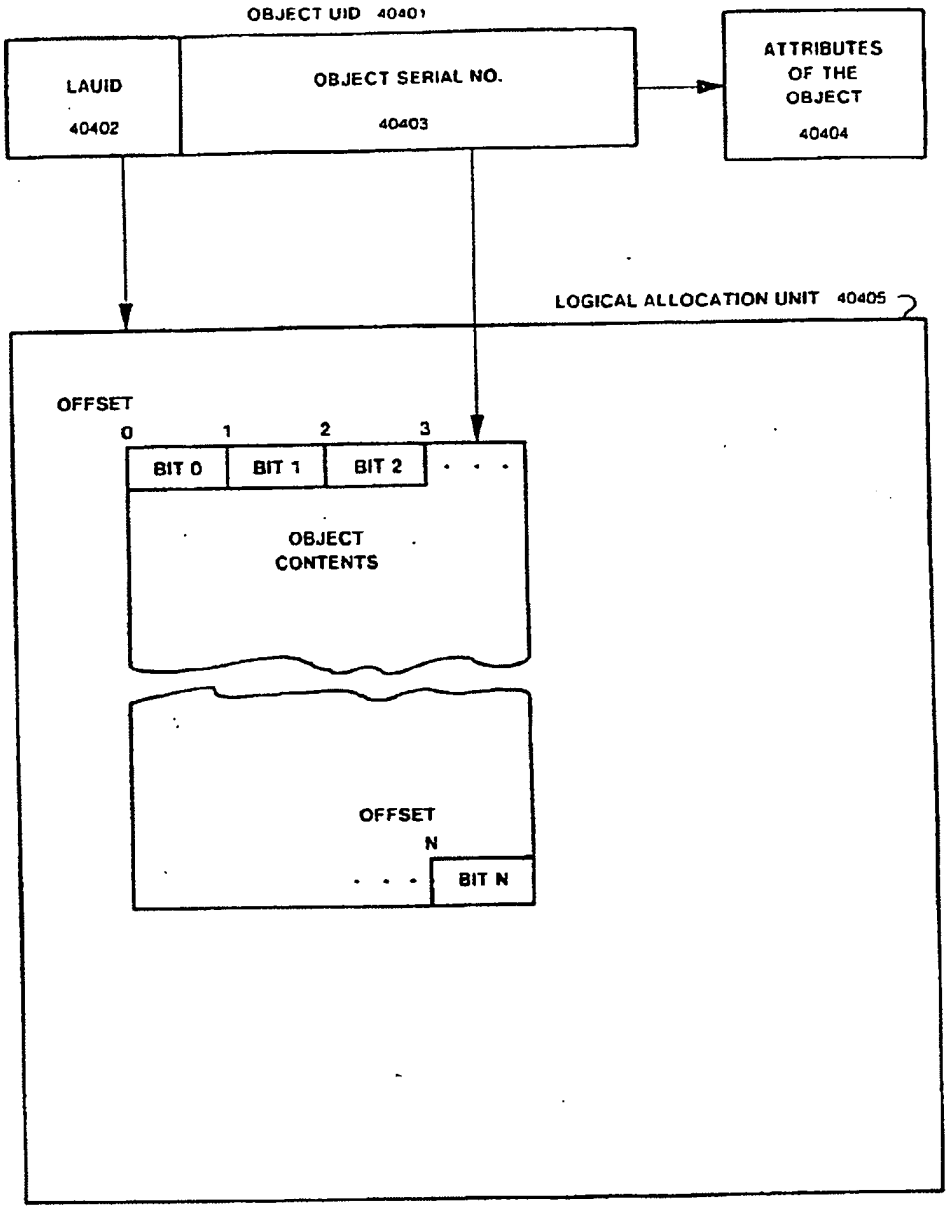


FIG 404

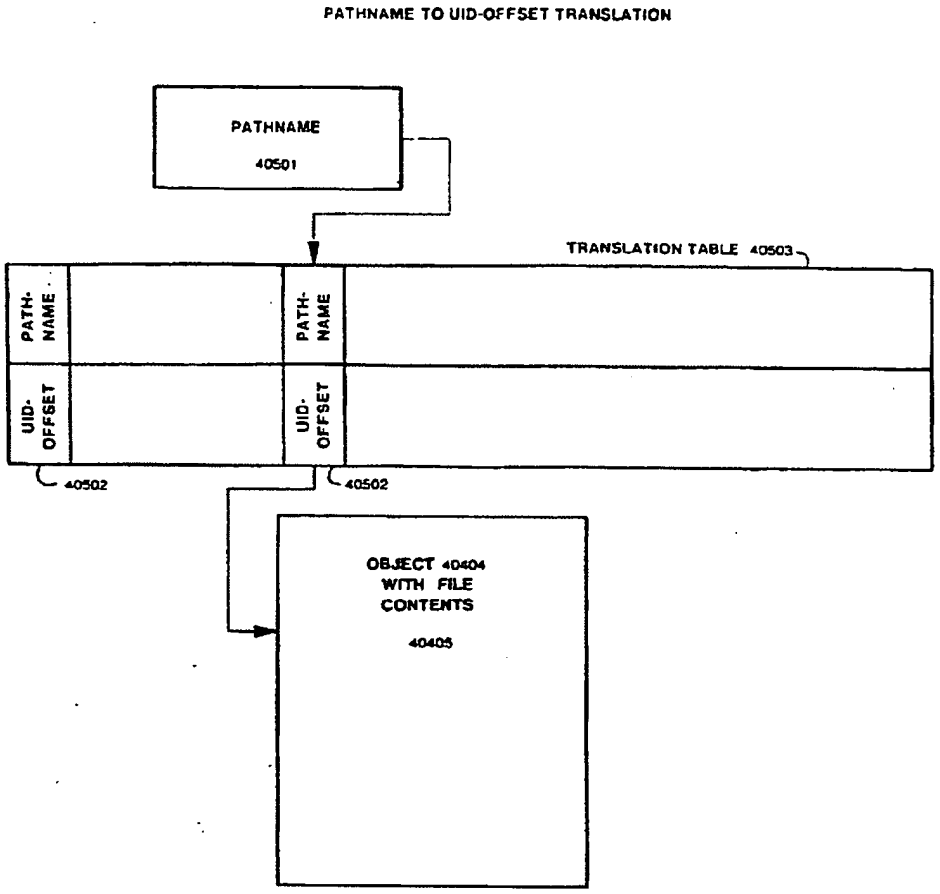


FIG 405

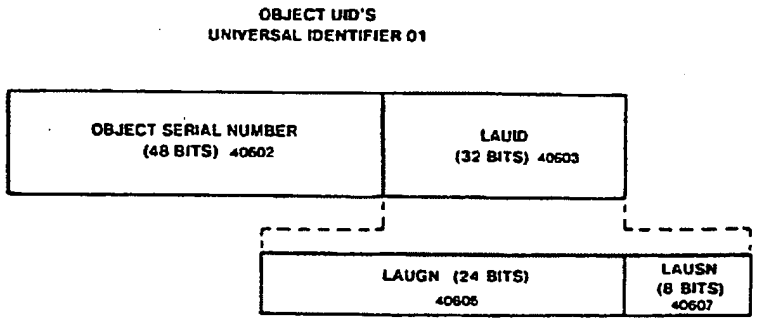


FIG 406

ATU, MHT, AND MEMORY

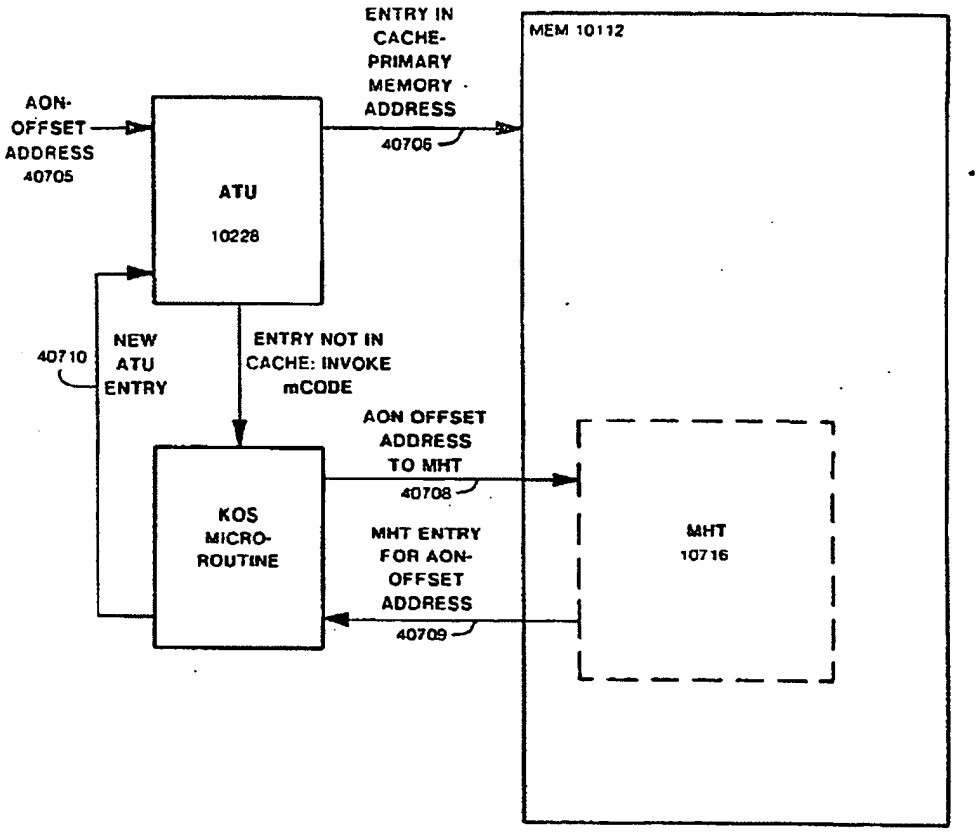


FIG 407

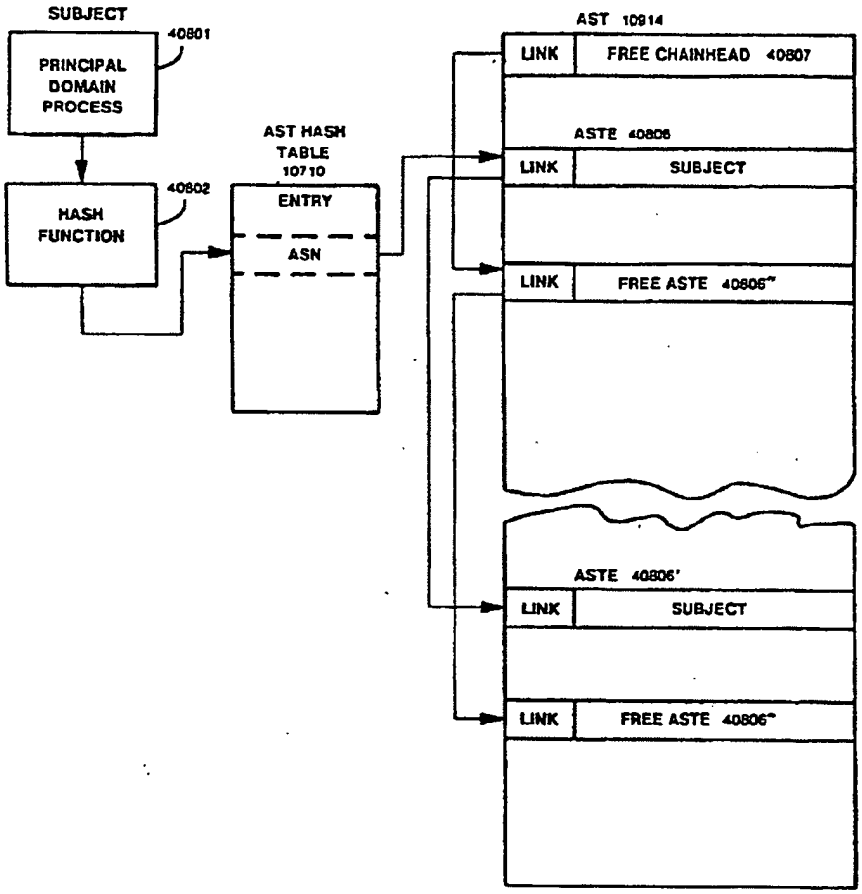


FIG 408

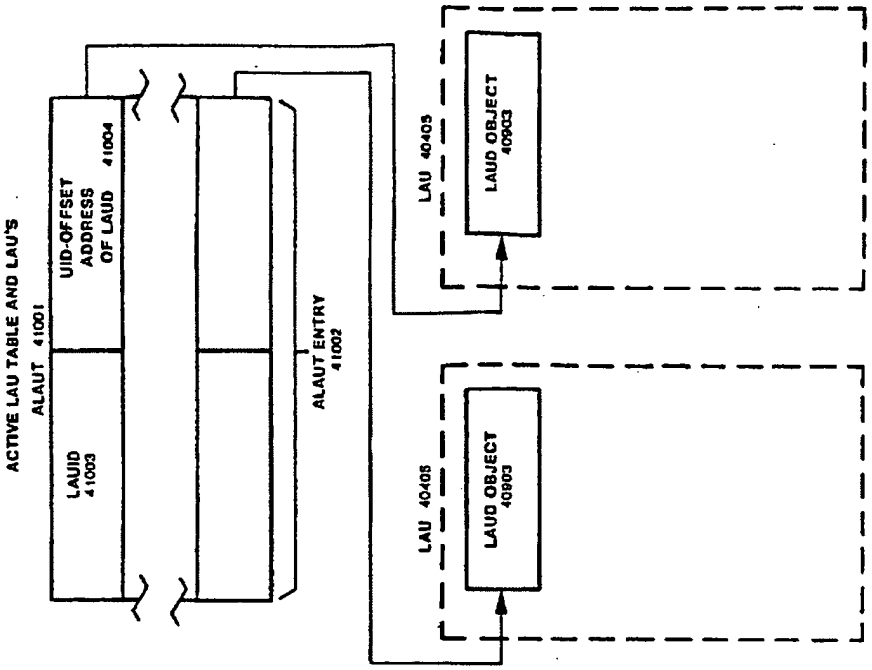


FIG 410

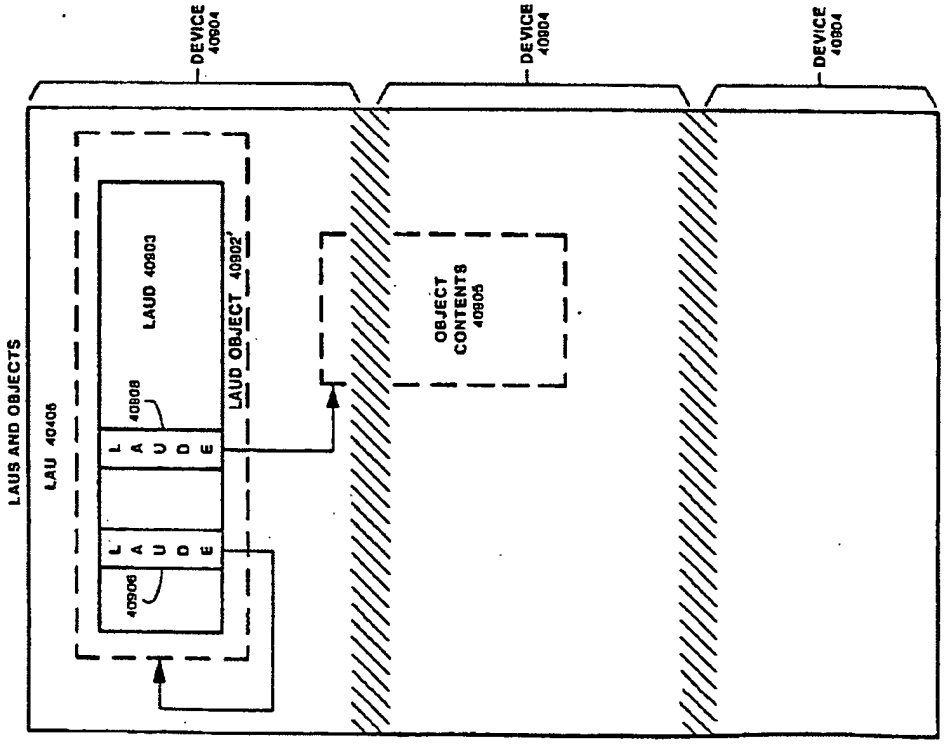


FIG 409

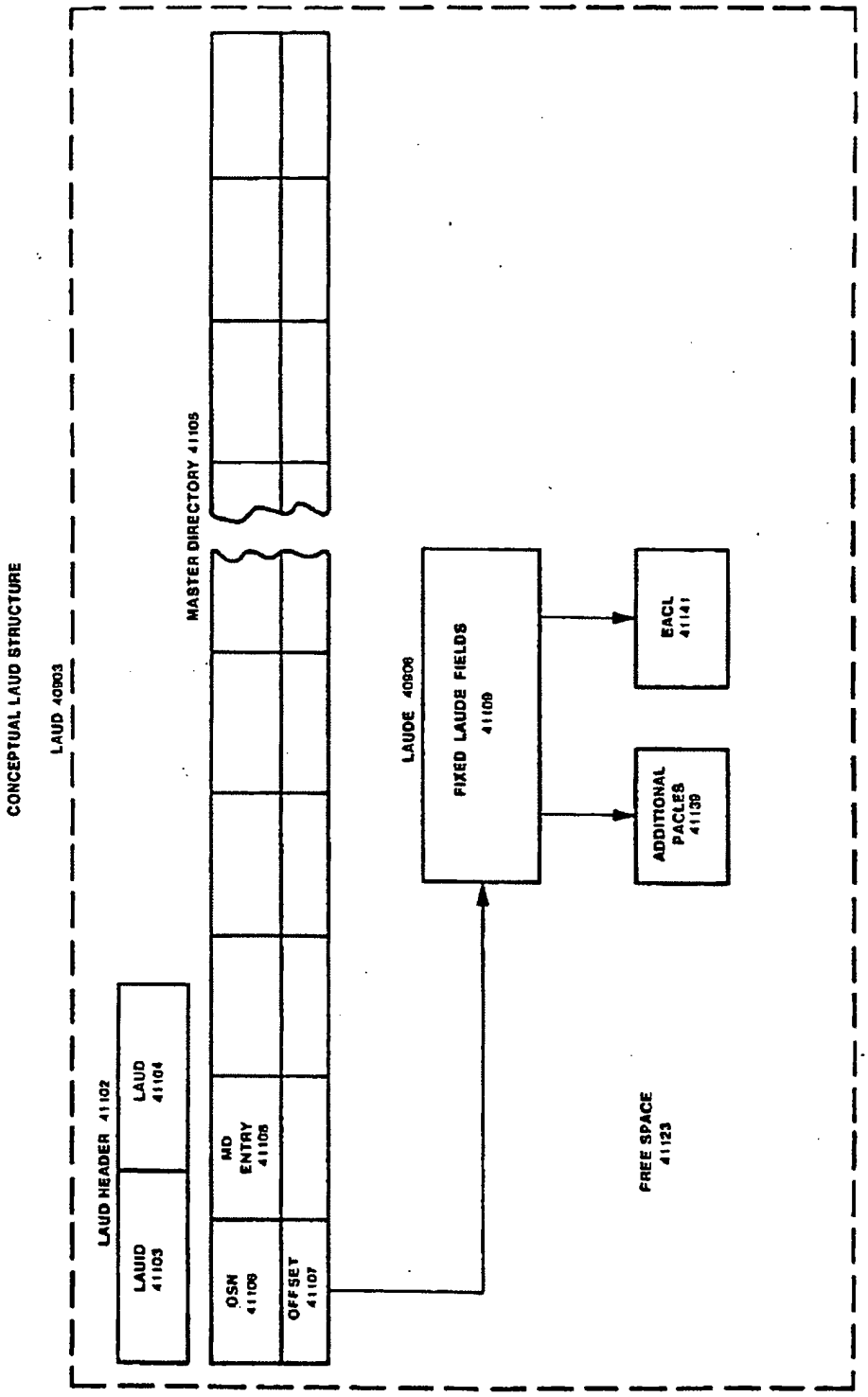


FIG 411

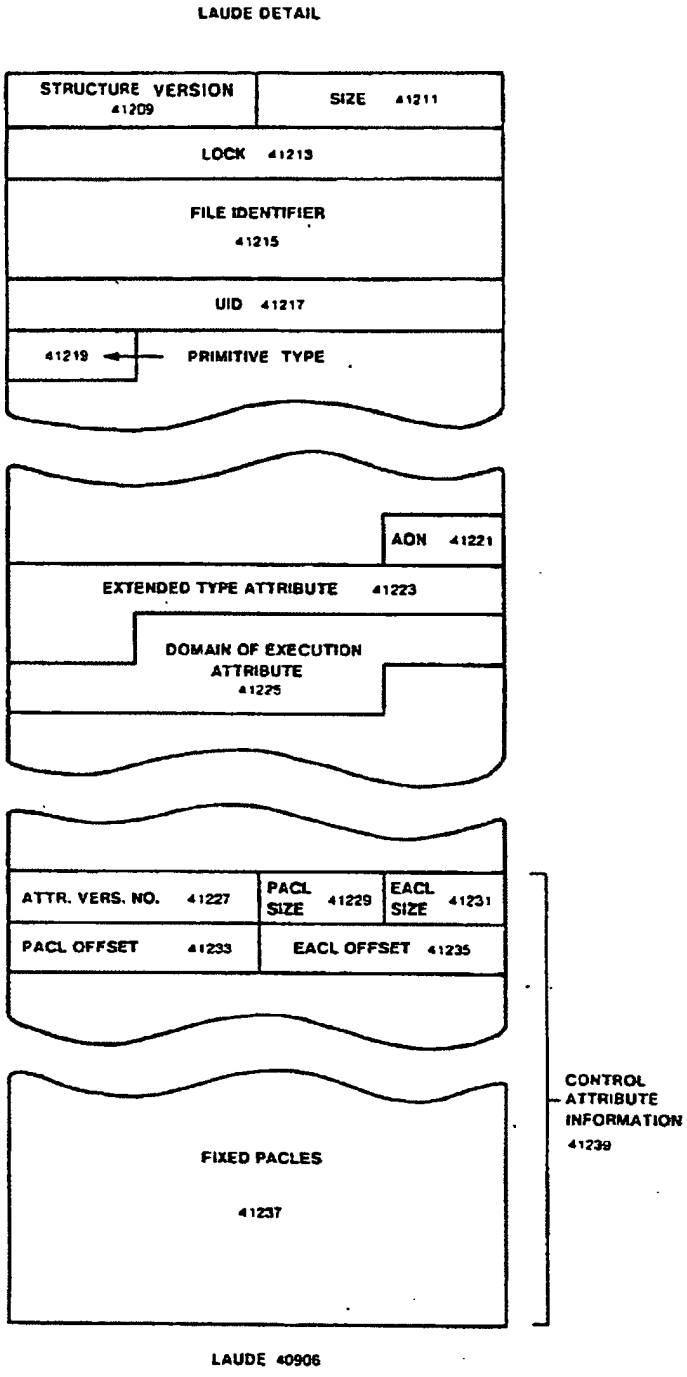


FIG. 412

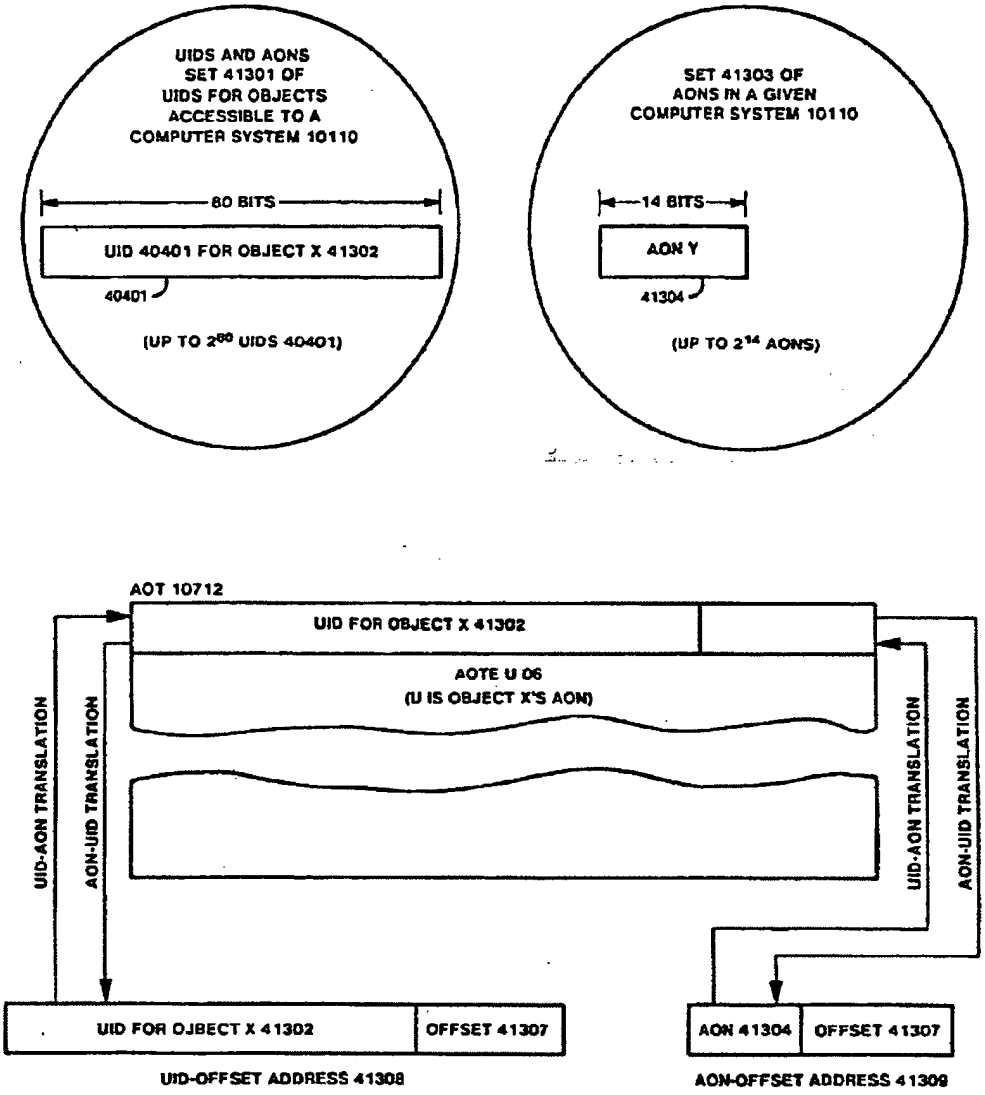


FIG 413

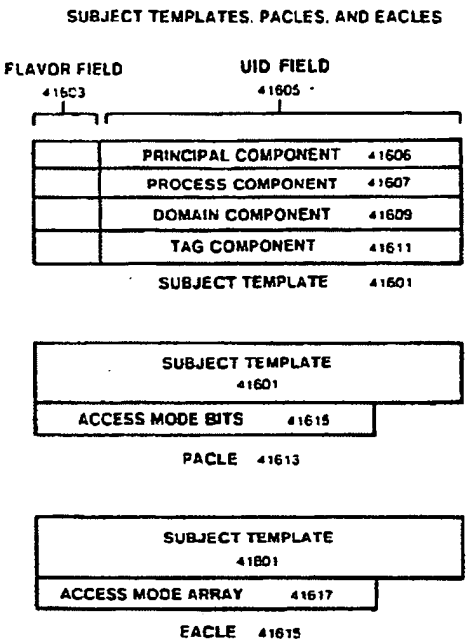


FIG. 416

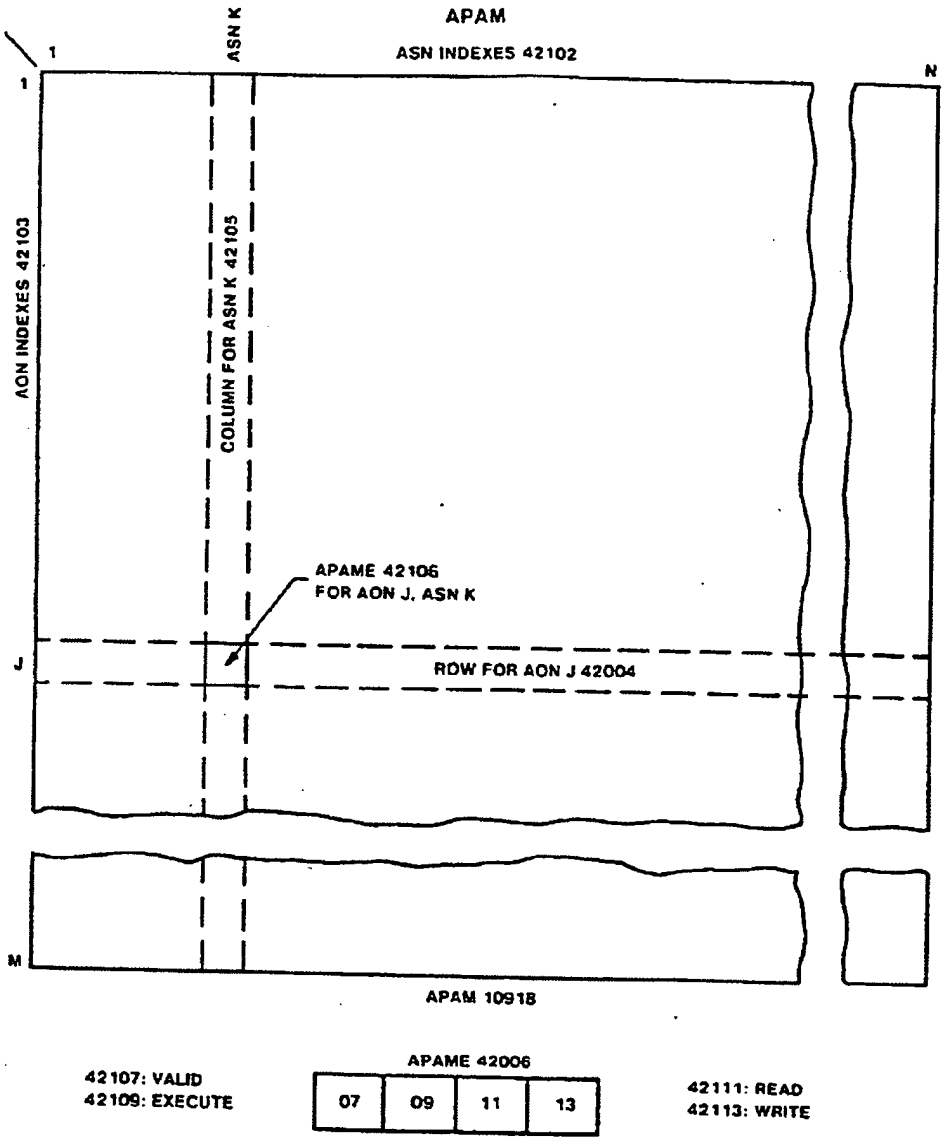


FIG. 421

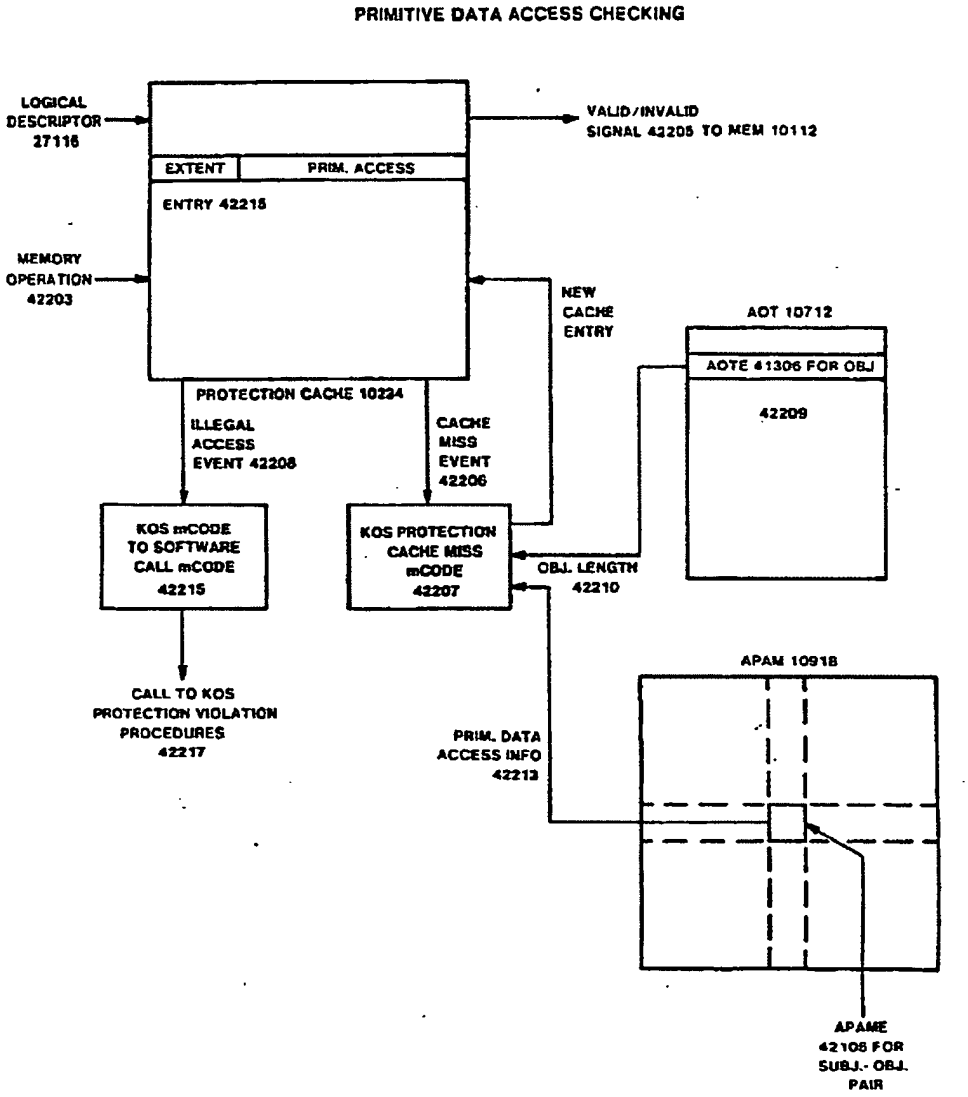


FIG. 422

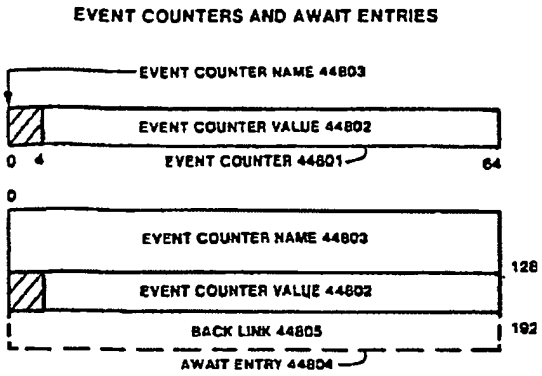


FIG. 448

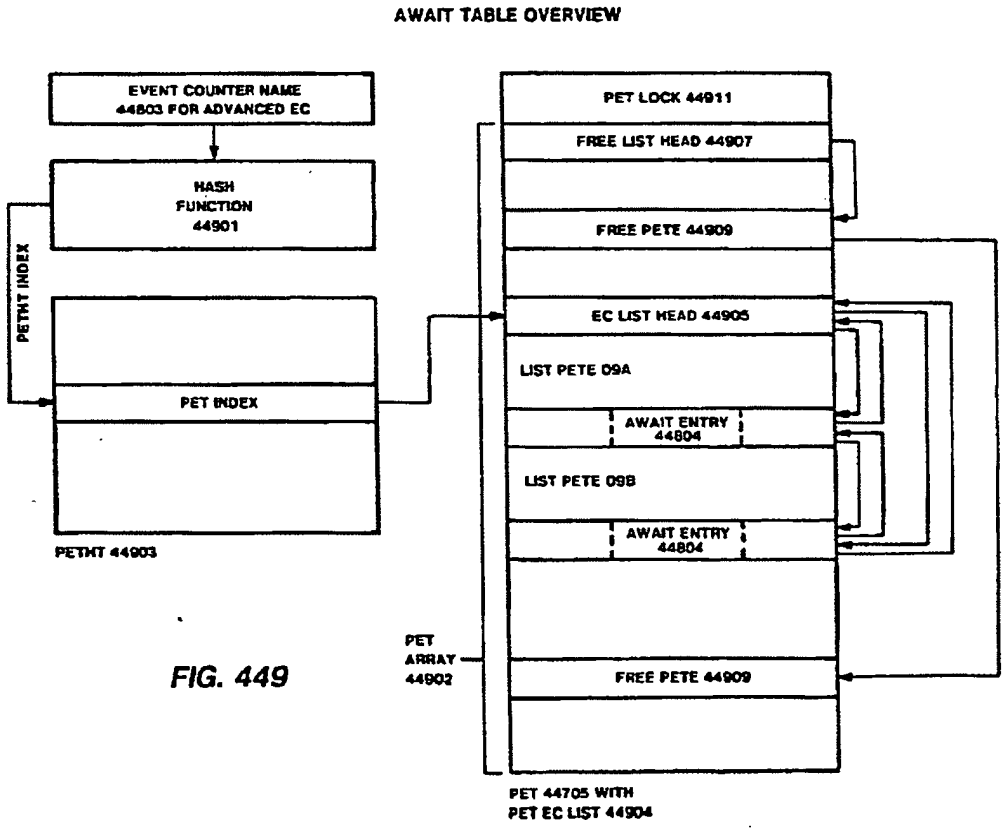


FIG. 449

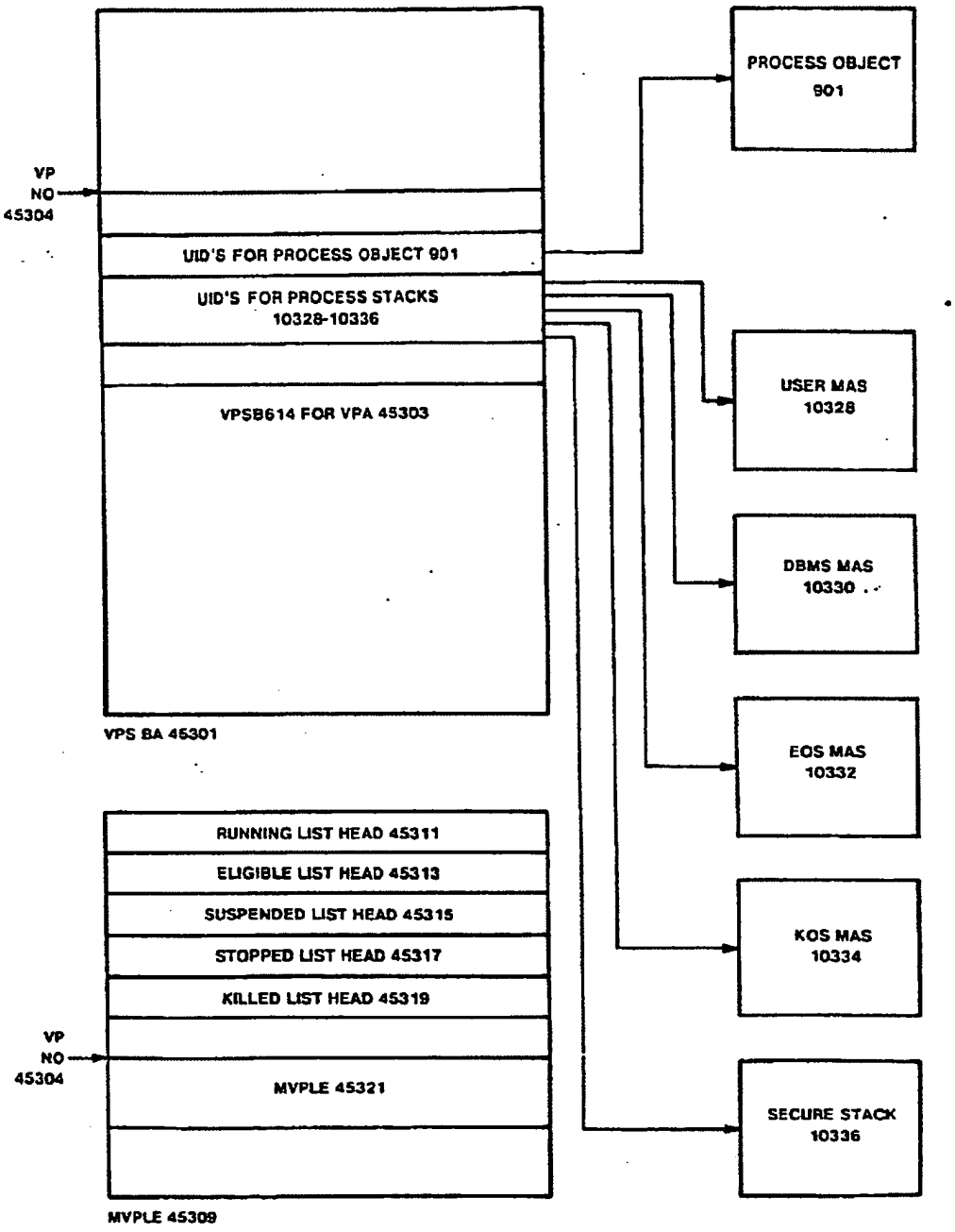


FIG. 453

VIRTUAL PROCESSOR SYNCHRONIZATION

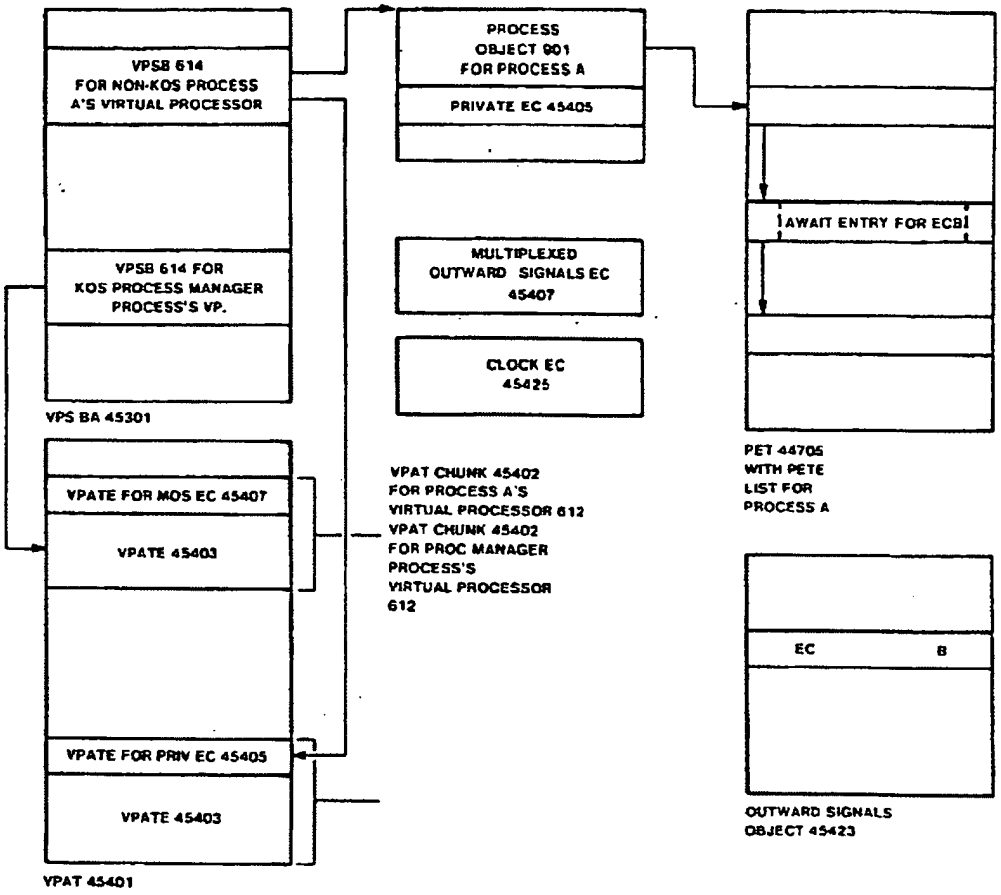


FIG. 454

MAS OBJECT OVERVIEW

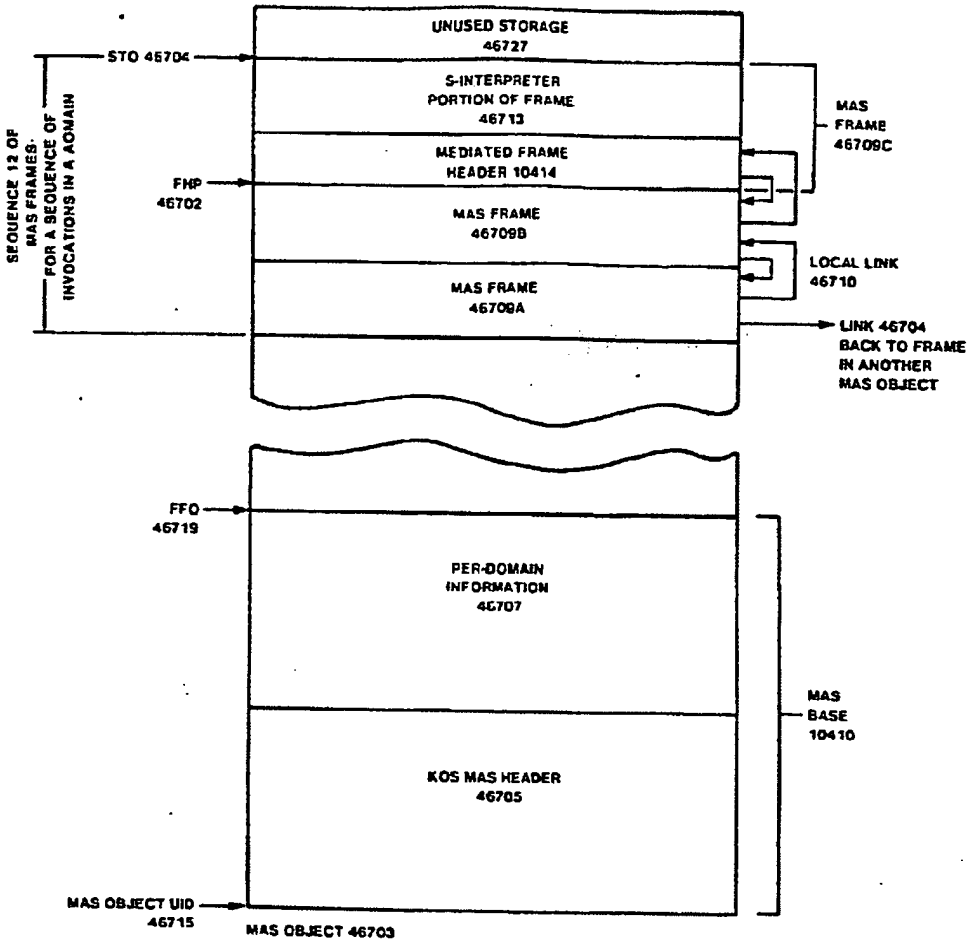


FIG. 467

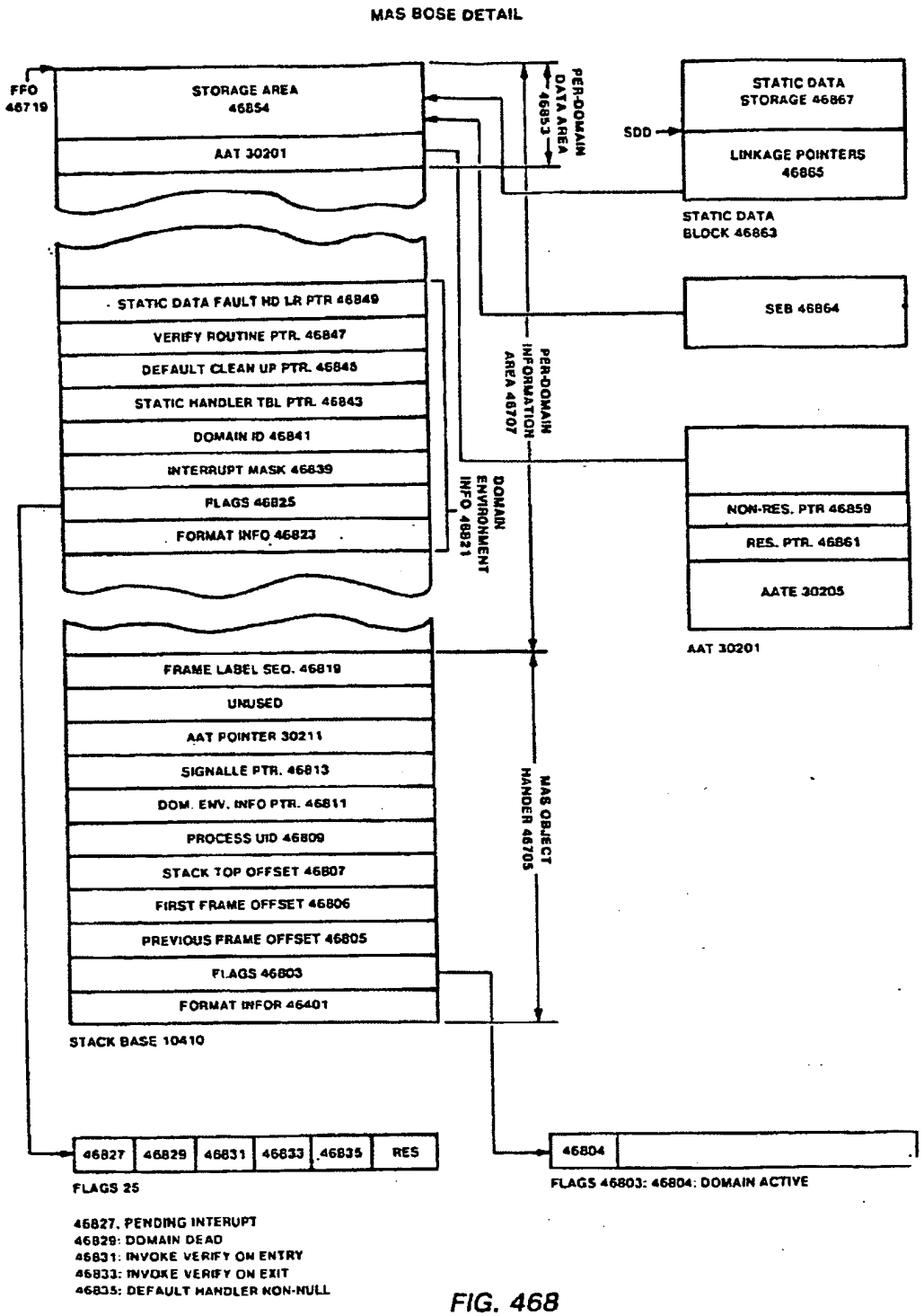


FIG. 468

MAS FRAME DETAIL

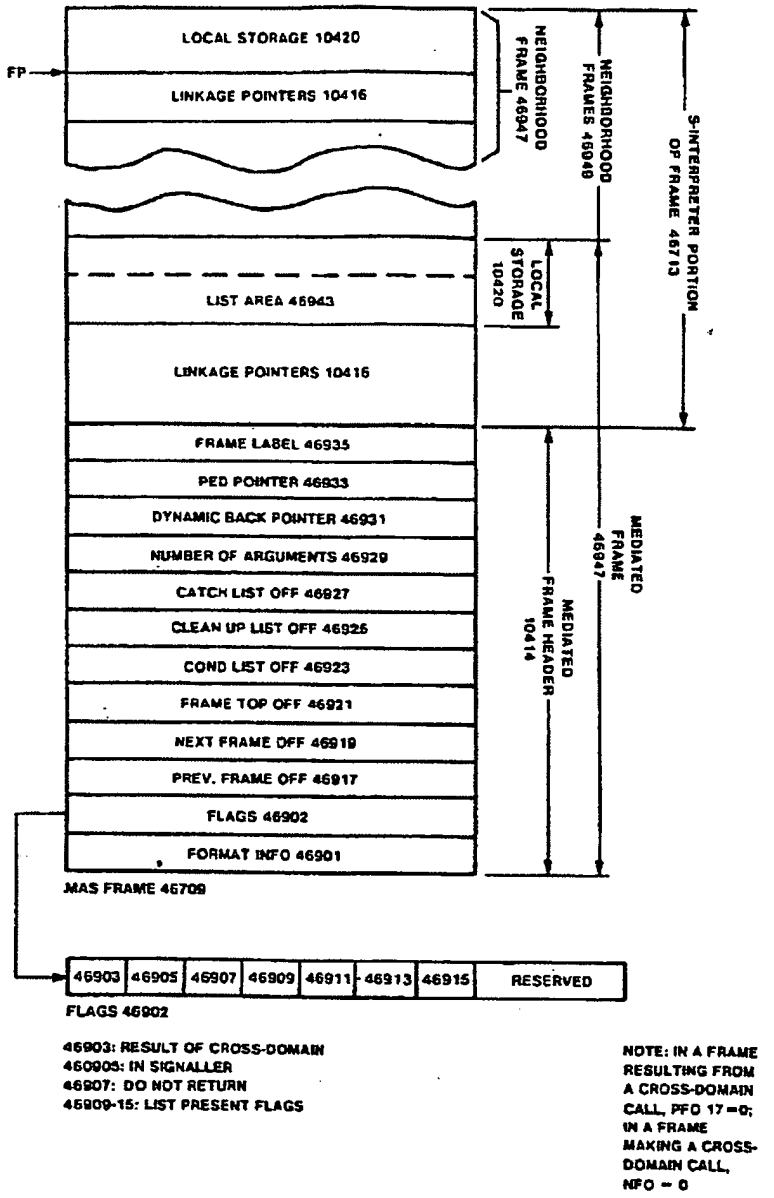


FIG. 469

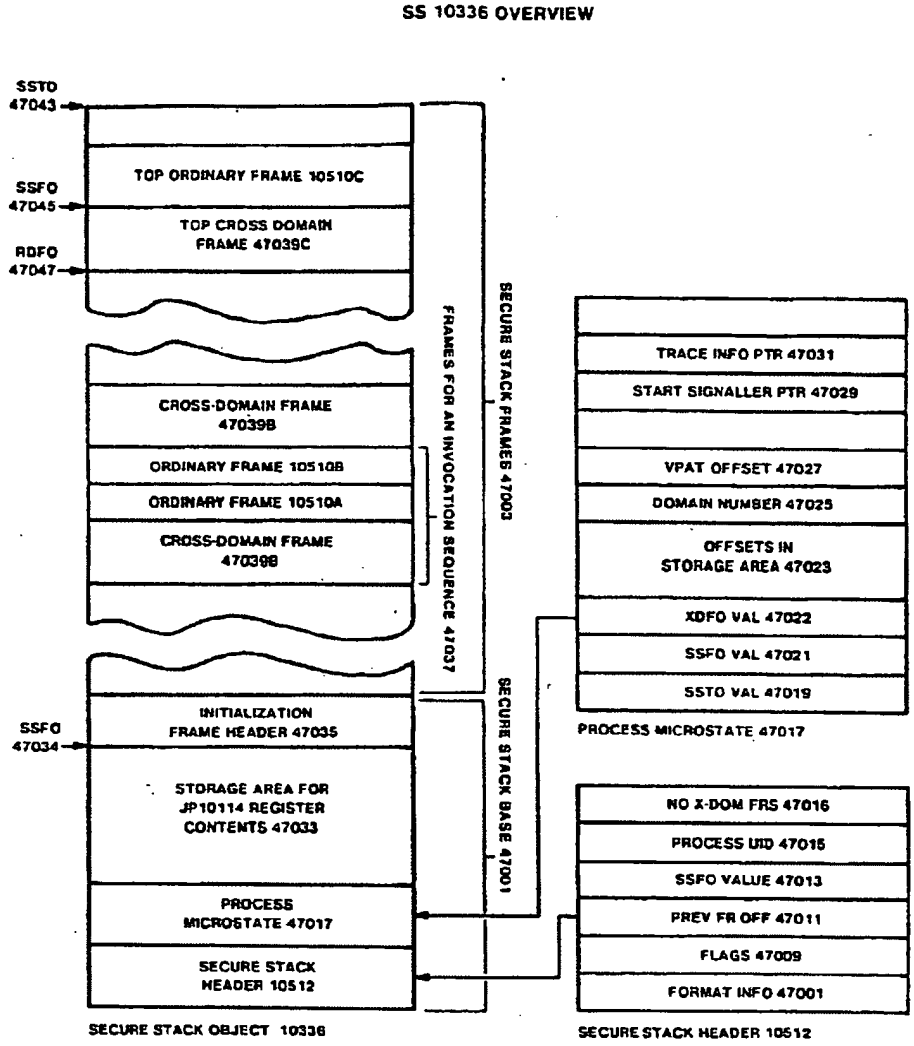


FIG. 470

SECURE STACK 10336 FRAME DETAIL

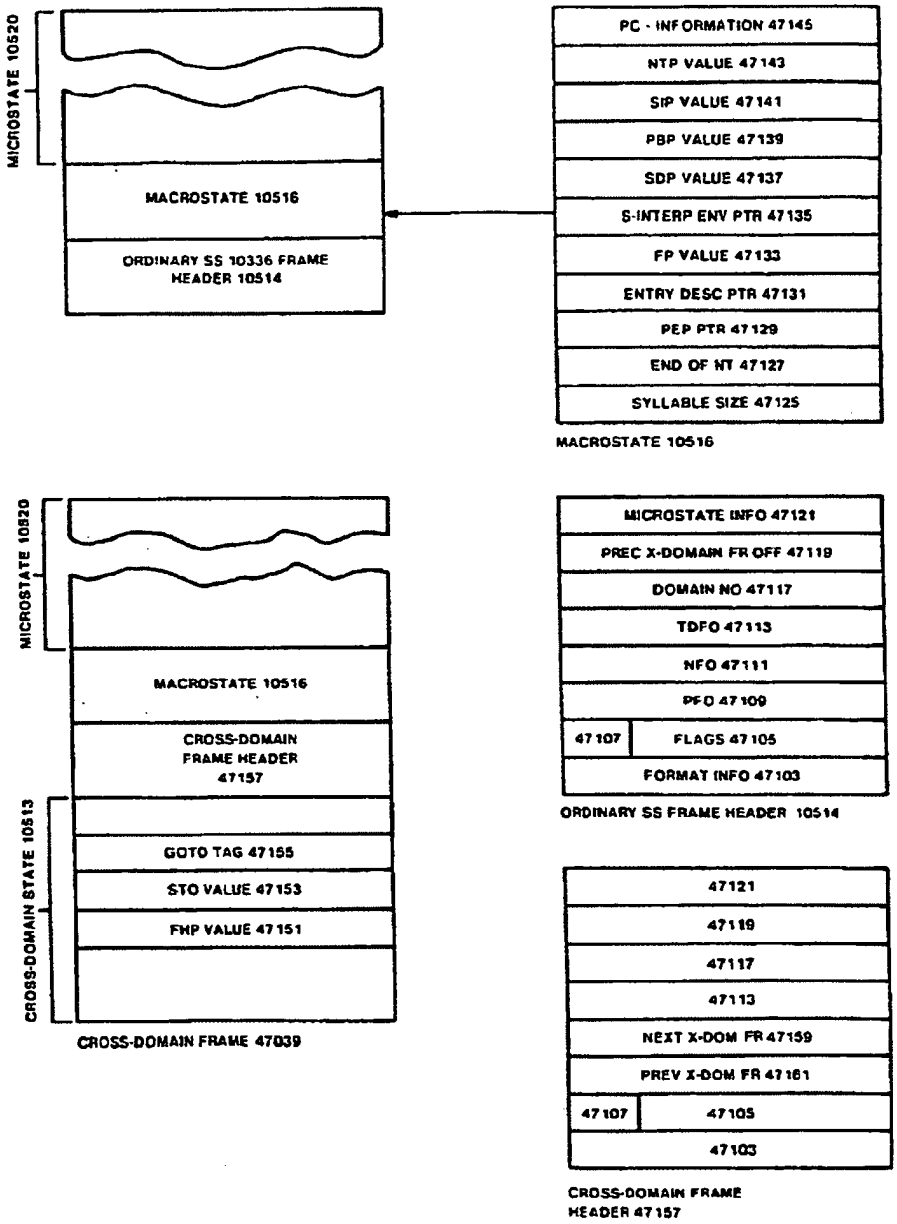


FIG. 471

PROCEDURE OBJECT OVERVIEW

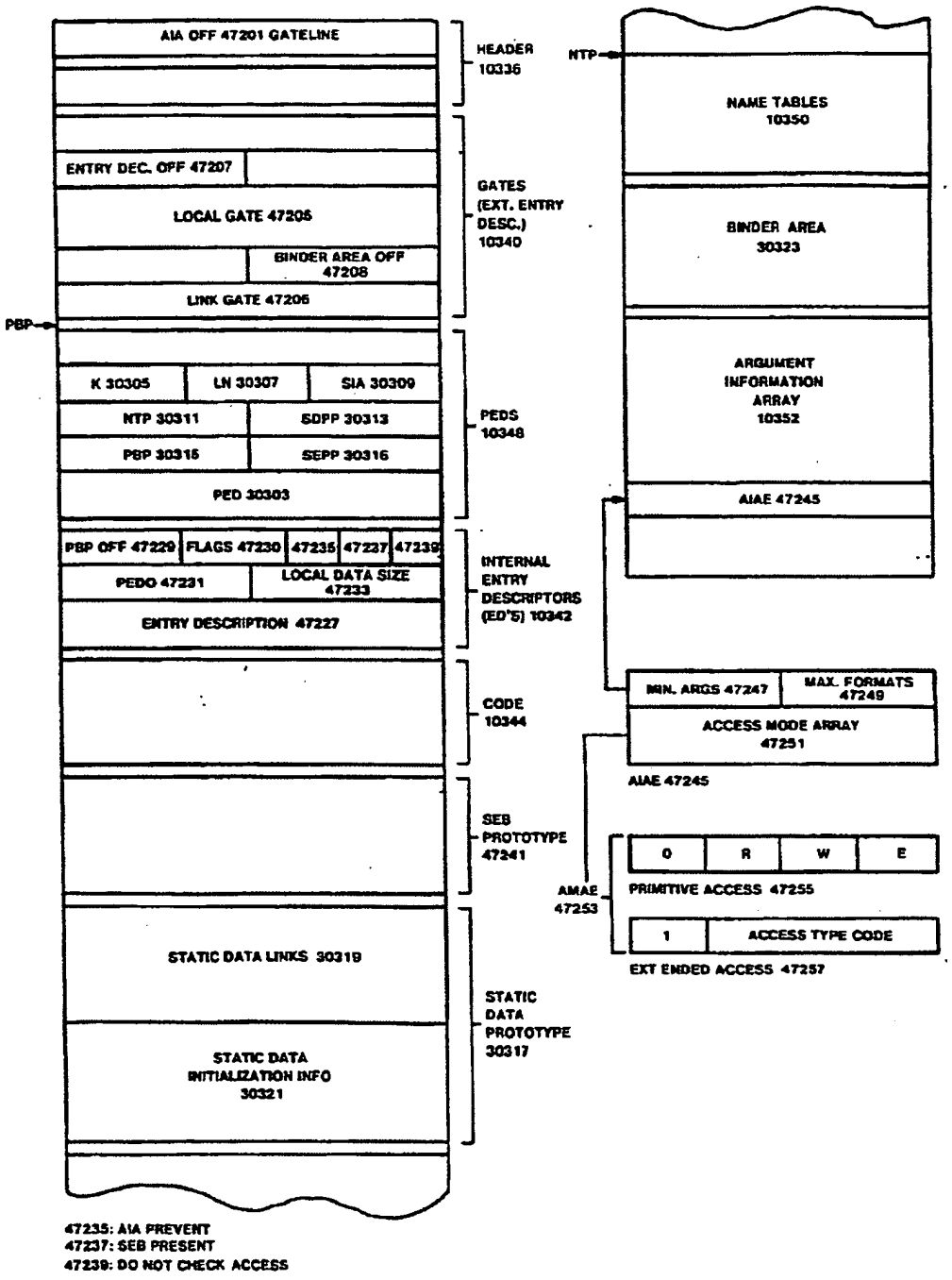


FIG. 472

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : H04L 9/08 — A1

(11) International Publication Number: **WO 99/34553**

(43) International Publication Date: 8 July 1999 (08.07.99)

(21) International Application Number: PCT/US98/27531

(22) International Filing Date: 30 December 1998 (30.12.98)

(30) Priority Data:
09/001,463 31 December 1997 (31.12.97) US

(71) Applicant: V–ONE CORPORATION [US/US]; Suite 300, 20250 Century Boulevard, Germantown, MD 20874 (US).

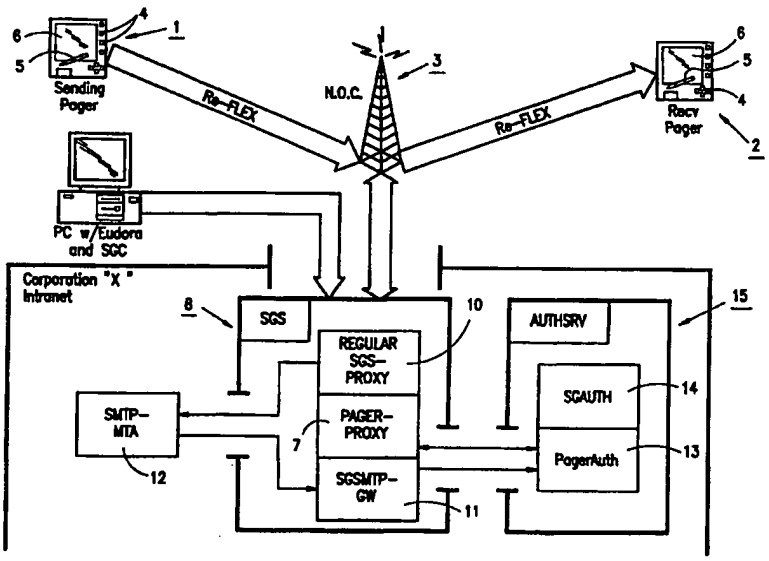
(72) Inventors: WRIGHT, Steven, R.; Apartment 21, 12010 Waterside View Drive, Reston, VA 20194 (US). BROOK, Christopher, T.; 7308 Pomander Lane, Chevy Chase, MD 20815 (US).

(74) Agents: URCIA, Benjamin, E. et al.; Bacon & Thomas, PLLC, 4th floor, 625 Slaters Lane, Alexandria, VA 22314 (US).

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: KEY ENCRYPTION SYSTEM AND METHOD, PAGER UNIT, AND PAGER PROXY FOR A TWO–WAY ALPHANUMERIC PAGER NETWORK



(57) Abstract

A method and system allows encryption services to be added to an existing wireless two–way alphanumeric pager (4) network by providing a pager proxy (7) which is arranged to receive an encrypted message from a sending pager (1) and re–packages it for retransmission to the destination pager (2). The sending pager encrypts the message using a session key, and encrypts the session key so that it can only be recovered by a secret key of the pager proxy. Authentication (13) of the sending pager and proxy server is provided by encryption of the session keys together with identifying data, and authentication of the message is provided by a message authentication code generated by computing a message authentication code based on the session key, identifying data, and the message.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

KEY ENCRYPTION SYSTEM AND METHOD, PAGER UNIT, AND PAGER PROXY FOR A TWO-WAY ALPHANUMERIC PAGER NETWORK

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention relates to a system and method of encrypting messages for transmission and/or receipt by a pager, and in particular to a system and method for which uses a standard two-way wireless pager protocol to send encrypted messages over an existing paging infrastructure. The invention also relates to a pager unit capable of sending and receiving encrypted alphanumeric messages over a wireless pager network, and to a pager proxy server which provides key management functions for enabling transmission of encrypted alphanumeric messages over the wireless pager network.

2. Description of Related Art

Paging systems capable of transmitting simple alphanumeric messages and displaying the messages on a miniature two-way pager are becoming increasingly popular. Such two-way paging systems enable messages like "Meet me at the gym at 6:00" or "I love you" to be both transmitted and received by equipment that is smaller, less complex, and less intrusive than a wireless telephone. The messages are transmitted as packets containing source and destination address data formatted for transmission over

the response channel of a wireless paging network, using a protocol that allows users to respond to messages directly from their pager units without having to use a telephone.

Two-way pagers are currently offered by Motorola and Wireless Access, with national paging services being provided by MTEL, which uses Motorola's Re-FLEX™ paging protocol. The Re-FLEX™ paging protocol allows users to respond to messages using a selection of pre-programmed responses or by formatting a free-form text reply, and in addition includes a TCP/IP protocol stack that allows the user to initiate messages to subscribers on wired networks, including e-mail and fax machine addresses.

The present invention concerns a method and system for encrypting and authenticating messages transmitted over the existing pager system, using the Re-FLEX™ protocol, or over other yet-to-be-implemented paging systems in the U.S. and elsewhere which may or may not use the Re-FLEX™ protocol. Unlike previously proposed arrangements, which either rely on complex encoding schemes and sophisticated hardware at the sending and destination ends of a transmission, over transfer of keys and authentication of keys using a telephone rather than the wireless network, the present invention offers the advantages of (i) providing authenticable key encryption of messages at the source of the transmission and key decryption at the destination, with protection of the communication by keys that are unique to each pager, rather than shared, and yet with no need for a key exchange between the sending and destination pagers, (ii) using existing two-way pager designs and paging system infrastructure, and (iii) providing the encryption capabilities without adding to carrier overhead. The addition of full key encryption and authentication capabilities to an existing pager system without adding to carrier overhead or capital costs distinguishes the system and method of the invention from all previously proposed pager encryption schemes.

An example of a previously proposed pager encryption scheme is described in

U.S. Patent Nos. 5,452,356 and 5,481,255, assigned to Data Critical Corp. Although the term "encryption" is used in these patents, the patents are directed primarily to a data compression and encoding protocol for enabling transmission of large volumes of data over a wireless pager network using modified transmitting and receiving hardware, including separate computers at each end of the transmission. The only discussion of encryption in these patents is a cursory reference to "encryption" as an added security layer provided by utilizing a "commercially available algorithm" (see, *e.g.*, col. 11, lines 15-32 of U.S. Patent No. 5,452,356) during encoding of the files by a computer connected to the pager. Because all encryption and decryption in the Data Critical patents is disclosed as being carried out by software on computers connected directly to the sending and receiving pagers, the only possible ways that true key encryption could be provided for would be to use encryption keys corresponding to decryption keys common to all possible recipients of the message, to use unique keys for each potential recipient but to store the corresponding encryption keys in the sender's computer, or to exchange keys prior to a transmission. While these alternatives might be reasonable in the context of, for example, a medical paging system in which all transmissions are between doctors or trusted medical personnel, none of them are suitable for use in connection with a paging system designed to transmit simple text messages using miniature handheld paging units and which is open to the general public, both because of the hardware intensive nature of the encoding scheme and the problem of key management.

In addition to the wireless pager protocol described in the Data Critical patents the prior art includes a number of patents describing authentication or encryption schemes that are used in connection with wireless paging, but are carried out over a telephone line. The systems described in these patents are more suited to traditional one-way paging environments than with two-way protocols, even though one of the patents issued only recently, and none disclose systems that can be practically applied to the current two-way paging networks.

U.S. Patent No. 5,668,876, for example, discloses a modified pager which provides authentication of a caller. The modified pager calculates a unique response code based on a transmitted challenge code, an input personal identification number, and an internal key. The resulting response code is converted into DTMF tones and transmitted by telephone to a central computer which authenticates the caller. This system does not provide for encryption of messages, or authentication by the receiving party of communications forwarded by the central computer, and yet requires a challenge response form of authentication which requires simultaneous two-way communications, which is currently neither possible nor required by existing two-way wireless pager protocols.

U.S. Patent No. 5,285,496 describes a paging system with two options: the first is to send and receive encrypted messages using private key encryption by transmitting a clear text message over a private communications line to a local client of the pager network where the message is encrypted using a private key, and broadcast over a pager network, and the second is to send the message in clear text by telephone directly to the central control system of the pager network, where the message is encrypted. However, neither of the two options provides for encryption of the original pager transmission, which must be sent in clear text form over a telephone line, and which, in the case where a local client computer is used, provides no way to maintain centralized control. In addition, for the local client computer option, in which the address is encrypted together with the message, the destination pager must decrypt every message sent over the system in order to determine whether a message is addressed to it, which is only possible in pager networks with a very limited number of participants.

In the system described in U.S. Patent No. 5,638,450, on the other hand, reception by a pager of encrypted messages over a radio frequency pager network is made possible by having the pager transmit an encryption key via DTMF tones over a telephone line to a central office, the central office then encrypting the messages before forwarding them

to the recipient. This system does not permit outgoing messages to be encrypted, and provides no way of key encrypting messages between two pagers on the network, and again is not applicable in the context of the present invention.

It will be appreciated that none of the above patents, representing the known pager message protection proposals, describes a system that enables true key encryption and authentication capabilities to be added to a conventional two-way wireless alphanumeric paging system of the type with which the present invention is concerned, using existing pager protocols and equipment, and in which any individual can send a simply alphanumeric message by keying the message into a miniature two-way pager (or choosing from a menu of pre-stored messages), entering a destination address, and pressing a send button, the message then being retrievable by the intended recipient by a simple keystroke on the recipient's pager, with the message being encrypted by a key unique to the sending pager and decrypted by a key unique to the destination pager. In contrast, the present invention not only provides these capabilities, but adds further levels of security by using strong secret or private key based encryption algorithms, with multi-tier authentication of a transmitted packet, while permitting central registration and billing for encryption services and recovery of messages by legal authorities without adding to carrier overhead or increasing the costs of the paging service for users who do not require encryption.

All of the above advantages of the system and method of the invention are made possible through the use of a proxy server to intercept an encrypted message and repackage it for delivery to the intended recipient in a form that the intended recipient is capable of reading, thus eliminating the need for shared keys or key exchange between the sender and ultimate recipient of the message or complex, hardware-intensive encoding schemes, and allowing encrypted messages to be transmitted using existing two-way alphanumeric pager protocols. Because the invention involves key encryption and not encoding of the message, and requires knowledge by the sending and receiving

units of only one or two keys (for example, a private key unique to the pager and a server's public key), encryption being simpler to implement than encoding since it merely involves performing arithmetically combining the message with the key, the present invention can be used with existing pager hardware and protocols, and by avoiding the need for challenge/response authentication, the present invention can be used with existing channels and therefore with the existing pager infrastructure. None of the previously proposed systems and methods has these capabilities.

Not only does the use of a proxy server relieve the sending and receiving pagers of key management functions, but the manner in which the invention utilizes strong encryption capabilities, by separately encrypting the session key, further minimizes the processing resources required by the sending and receiving pagers. Essentially, encryption of the message itself can be carried out with a relatively short session key to minimize usage of the processor, while the relatively short session key can be protected by a strong encryption algorithm. Because the session key is not re-used, key integrity can easily be maintained.

SUMMARY OF THE INVENTION

It is accordingly a first objective of the invention to provide a system of adding full key encryption services to a pager network, allowing key encrypted alphanumeric messages to be sent by any pager unit registered with the encryption service provider to any other registered pager unit via the network, as well as to e-mail addresses, fax machines and other destinations capable of receiving text messages.

It is a second objective of the invention to provide a method of adding full key encryption services to a pager network, allowing key encrypted messages to be sent by any pager unit registered with the encryption service provider to any other registered pager unit via the network, as well as e-mail addresses, fax machines and other

destinations capable of receiving text messages.

It is a third objective of the invention to provide a system which allows encryption of alphanumeric messages by a paging unit for wireless transmission over a paging network in a manner which is transparent to the person sending the message, and which allows decryption and display of the messages by a receiving pager in a manner which is transparent to the person receiving the message.

It is a fourth objective of the invention to provide a method which allows encryption of messages by a paging unit for wireless transmission over a paging network in a manner which is transparent to the person sending the message, and which allows decryption and display of the messages by a receiving pager in a manner which is transparent to the person receiving the message.

It is a fifth objective of the invention to provide a system and method of adding encryption capabilities with centralized key management and unique secret keys for each user, without modification of existing pager network infrastructure or paging transmission protocols.

It is a sixth objective of the invention to provide a system and method of encrypting text messages capable of being transmitted over a pager network, which can be provided as an add-on or option to the services provided by the pager network, and which can be centrally managed using a proxy server connected to the network to provide the encryption services to subscribers who select the encryption option.

It is a seventh objective of the invention to provide a system and method of authenticating messages transmitted in encrypted form over a pager network, without the need for an authentication channel or challenge/response protocol.

It is an eighth objective of the invention to providing a standard alphanumeric pager unit with the capability of encrypting, decrypting, and authenticating messages transmitted using a two-way alphanumeric pager protocol, with minimal or no hardware modification.

It is a ninth objective of the invention to provide a proxy server arrangement which can be connected to the network operations center of a pager network in order to manage transmission of key encrypted messages over the network.

These objectives are achieved, in accordance with the principles of a preferred embodiment of the invention, by using a pager proxy server to carry out decryption of a message encrypted by a session key and received from the sending pager, and to have the pager proxy generate a new session key for re-encryption of the message transmitted to the receiving pager, with the original session key being encrypted at least by a secret key shared by the sending pager and the pager proxy server or by a public key corresponding to a private key of the pager proxy server, and the new session key being encrypted either by a secret key shared by the pager proxy server and the destination pager or a public key corresponding to a private key held by the destination pager, thereby freeing the sending and destination pagers from having to store more than one secret key or of having to carry out a direct exchange of keys, and allowing each pager on the network to be provided with a unique key.

In accordance with the principles of an especially preferred embodiment of the invention, in order to encrypt a message, the sending pager must have hard-coded into memory a unique identification number and a secret key associated with the identification number. When a user is ready to send an encrypted message, he or she begins by entering the message to be sent, after which the user is prompted for an access code to gain access to the encrypted shared key, the encrypted shared is decrypted, and a session key is generated. The message that was entered by the user is then encrypted with the

session key, and the session key is encrypted with the public key of the pager proxy server, or a shared secret key of the sending pager, and appended to the encrypted message for transmission via the network operations center to the pager proxy server.

Pager messages are formatted in accordance with standard pager protocols to include a destination header, which is generally the address or telephone number of the receiving pager, and with an additional space in the header to indicate that the message is encrypted, as will be explained in more detail below. When the network operations center receives a message that is in encrypted form, it forwards it to the encryption service center, which must at least include a pager proxy server, using an appropriate protocol, examples of which include but are not limited to TME-X and TNPP. In the illustrated embodiment, the pager proxy server is included in a gateway server in order to enable the system to package e-mail messages for transmission in encrypted form to pagers on the pager network, or to package pager messages according to an e-mail protocol for transmission over a wired network such as the Internet to an e-mail address, but it will be understood by those skilled in the art that the pager proxy may be operated as a separate unit.

In the illustrated embodiment of the invention, the pager proxy server has the role of verifying the authenticity of the message sent by the sending pager, decrypting the data with its private key or alternatively with a secret key shared with the sending pager to obtain the session key that was generated by the sending pager, and decrypting the message with the session key generated by the sending pager. Once this is accomplished, the server generates a new session key to encrypt the message with, and then encrypts the session key with a secret key shared with the destination pager or with a public key corresponding to the private key of the destination pager, or alternatively with a secret key shared with the destination pager, the two entities being appended together and sent to the recipient pager. The destination pager, after receiving the encrypted message, alerts the user and, when the user is ready to read the encrypted page, prompts him or her

for the access code to begin decryption of the appropriate shared secret key or private key, which is then used to decrypt the session key used to decrypt the message.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a schematic diagram showing the principal elements of a pager encryption system constructed according to the principles of a preferred embodiment of the invention.

Fig. 2 is a schematic illustration summarizing the operation of the two-way pager for sending an encrypted message over a wireless network in accordance with the principles of a preferred embodiment of the invention.

Fig. 3 is a functional block diagram of a module used by a two-way pager to encrypt a message and package it for wireless transmission over a pager network to a network operations center.

Fig. 4 is a functional block diagram of a module used by a pager proxy server to authenticate the sender of an encrypted message, authenticate the message, and extract information from the message which can be used to re-package the message for transmission a destination address.

Fig. 5 is a functional block diagram of a module used by the pager proxy server to repackage a message and send it to the network operations center for transmission for re-transmission over the wireless pager network to a destination pager.

Fig. 6 is a functional block diagram showing the principal elements of a module used by a destination pager to decrypt and display a message received in encrypted form from the network operations center over the wireless paging network.

Fig. 7 is a flowchart of a preferred process corresponding to the functional block diagram of Fig. 3.

Fig. 8 is a flowchart of a preferred process corresponding to the functional block diagram of Fig. 4.

Fig. 9 is a flowchart of a preferred process corresponding to the functional block diagram of Fig. 5.

Fig. 10 is a flowchart of a preferred process corresponding to the functional block diagram of Fig. 6.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

As illustrated in Fig. 1, the system of the preferred embodiment of the invention allows encrypted communications between a sending pager and a receiving pager via a two-way wireless paging system such as M-TEL's system, using two-way alphanumeric pagers such as, but not limited to, the Motorola and Wireless Access pagers. The basic elements of the system are a sending pager 1, a receiving pager 2 which may be identical to the sending pager, and a network operations center (NOC) 3 which provides basic message forwarding and subscription management services for all communications carried by the system.

As is conventional, the sending and receiving or destination pagers (or pager units) 1 and 2 include function and data entry keys 4, and/or a stylus 5 or other data entry device, for allowing a user to input and send alphanumeric messages, and an LCD or other device 6 which allows received alphanumeric messages to be displayed. The pagers can also provide other functions such an alarm function to alert the user that a message has been received, and includes a microprocessor and circuitry capable of formatting an

input message and transmitting it to the network operations center according to an appropriate protocols, including but not limited to the ReFLEX™ protocol. The sending and receiving or destination pagers also include a memory for storing a unique user identification number (UID) that identifies a particular pager for addressing purposes, and other information such as a password that can be used to prevent unauthorized users from accessing the transmission or message display functions of the pager, as well as an addressing mode (AM) generator that is used in the pager protocol to indicate the type of addressing used by the paging system, and a timer that can be used to generate a message number.

In order to be used with the system and method of the illustrated embodiment of the invention, the pager memory must also have stored therein at least a private key of the pager unit, a corresponding public key of the pager unit, and a public key corresponding to a private key of the server, for encrypting either the message itself or a session key used to encrypt the message, and software capable of running on the included processor for performing an encryption algorithm and a decryption algorithm. In addition, according to the preferred embodiment of the invention illustrated in Figs. 2-10, the pager must be capable of generating a session key for each message to be transmitted, storing a private key unique to the pager which is used to authenticate the pager, and computing a message authentication code which is used to authenticate the message being transmitted or received.

It will be appreciated by those skilled in the art, however, that whenever a public key or private key is required, a shared secret key could be substituted using an appropriate algorithm, and that while the use of session keys is highly advantageous, the session key could also be eliminated in favor of public-private key encryption. In addition, while the illustrated system provides both encryption and decryption capabilities in at least two pagers, so that each pager can send or receive messages, the system and method of the invention could also be applied to systems in which some or

all of the pagers have reception capabilities only, *i.e.*, in which some or all of the pagers are designed to allow the pagers to receive encrypted messages originating from e-mail addresses and/or two-way pagers, but not to originate messages. Conceivably, the system and method of the invention could even be applied to systems in which at least some of the pagers are capable of sending encrypted messages, but not receiving and decrypting them, although such a system would seem to make little commercial sense. In any case, it will be appreciated that the system and method illustrated in Figs. 2-10 are intended as being illustrative in nature only, and should not be interpreted as being limitative of the scope of the invention.

As indicated above, the number of keys required of a pager to encrypt and decrypt messages is at most two, so that the key storage requirements are minimal. The encryption algorithms themselves simply involve a series of mathematical steps, and are well within the capabilities of the microprocessors used in the conventional pagers, as are message authentication code generating techniques such as CRC or SHA1. The session key used in the preferred embodiment to encrypt the message itself consists, in a practical implementation, of just sixteen characters (128 bits), and thus encryption of the alphanumeric message using RC4 or a similar stream cipher or other algorithm which makes use of a shared secret key can be accomplished without a large amount of processing resources, while strong overall protection of the transmission is still provided because the more processor intensive encryption algorithms are reserved for encryption of the relatively small session key rather than the alphanumeric message itself. Of course, the session key is not limited to a particular bit size, and it is possible for example to use 256 bit session keys, or longer or shorter session keys as desired.

In the preferred embodiment, encryption of the session key is carried out by RSA (1024 bits) but other stronger private key algorithms such as ECC PK1 (~2500 bits) can also be used, as well as shared secret key-based encryption methods such as RC4. The public-private key encryption algorithms are preferred not only because of the strong

encryption provided, but also because the permit authentication of the sender, as explained below, but legal or other considerations may also affect the choice of encryption algorithm, and thus the system of the invention is designed to permit the use of different mutually exclusive encryption algorithms by the sending and destination pagers.

The sending pager 1 illustrated in Fig. 1 transmits messages to the network operations center 3 in the form of a packet that includes a clear text applications header that tells the center to forward the text to the pager proxy server 7, which is conveniently though not essentially included in a gateway 8 capable of network communications as well as the pager encryption and decryption functions required by the present invention. Forwarding of the packet to the pager proxy or gateway server preferably involves use of a network data transfer protocol such as TME-X, although the manner in which the packet is forwarded to the proxy will depend on the wireless protocol used by the pager network and the capabilities of the network operations center. TME-X is a preferred transfer protocol for use with Re-FLEX encoded packets because of the presence of a TCP/IP stack in the standard format packets that allows the Re-FLEX™ protocol to communicate directly with computer networks.

The gateway 8 may include a general purpose proxy server 10 such as the one described in U.S. Patent No. 5,602,918, entitled "Application Level Security System And Method," and also in U.S. Patent Application Ser. No. 08/917,341, filed August 26, 1997, entitled "Multi-Access Virtual Private Network," both of which are incorporated herein by reference. The two patent documents describe a system currently available from V-One Corporation of Germantown, Maryland under the name SmartGate™ (SG in the figures) which is especially suitable for use with the pager proxy of the present invention, although the pager proxy server of the invention could also be used with other gateway servers, or without any network connection capabilities.

As illustrated, gateway 8 also includes a dedicated e-mail server or gateway 11, and e-mail protocol message transfer agent (MTA) 12 for transferring messages from the gateway server 10 to the e-mail gateway. Both the e-mail gateway 11 and pager proxy 7 may be physically incorporated in the gateway server or provided on independent or separate computers, and are connected to a pager authentication module 13 which may be physically incorporated into a general purpose gateway authentication module 14 of a separate authentication server 15, combined with the gateway server, or may be provided as an independent unit.

Computers on the network with capabilities of communicating with the general purpose proxy server are represented in Fig. 1 by computer 16, and include gateway client software that permits the computer to establish a secured communications path to the gateway server, as well as an e-mail program which packages messages in an appropriate format such as that provided by the SMTP protocol for transmission over the secured communications path established by the gateway client software. An example of an e-mail program is "Eudora™," although the use of standard protocols such as SMTP and Re-FLEX™ allows any e-mail program to communicate with the gateway and thence with the pager network, so that the system of the invention is not limited to use in connection with any particular e-mail program, the conventional pager network already being equipped to handle e-mail transmissions to or from the wireless network. The invention may be considered to apply equally to pager-to-pager communications, pager-to-email communications, and email-to-pager communications. In addition, it is possible that the invention could be adapted to communications originating from a fax machine, in which case the clear packet transmitted by the fax machine over a telephone line would be processed by a facsimile proxy for packaging and encryption by the pager proxy, and messages addressed to the fax machine would be decrypted by the pager proxy and forwarded to the facsimile proxy for transmission as clear text over a telephone line, the principles of the invention still being applicable to the encryption and decryption of the messages by the pager proxy and sending or receiving pagers.

Turning to the specific embodiment illustrated in Figs. 2-10, the system and method of the invention take the form of modifications to the header of the transmission packet sent by the sending pager 1 and/or the pager proxy 7. Essentially in order to send messages over the paging system, the sending pager and pager proxy, (or pager proxy alone in the case of a message originating from computer 16 or a source of clear text messages such as a facsimile machine) generates a header which includes the information necessary to enable processing by the recipient of the packet, and in the case of the pager proxy, for forwarding of a repackaged packet to a destination address. The header should at least include the session key encrypted message, the encrypted session key, a sender identification number, and a destination header or address, but because the header format will vary if a protocol other than Re-FLEX™ is used, it should be appreciated that the other information contained in the illustrated header, and the position of the information, can be varied without departing from the scope of the invention, and the invention is intended to encompass headers formatted for other alphanumeric wireless paging protocols, as well as for encryption algorithms and authentication protocols other than the specific algorithms and protocols indicated.

Fig. 2 illustrates the format of the preferred header, which is divided into three fields. It is to be understood that while the illustration refers to the communication between the sending pager and the pager proxy, the same header will be used for the communication between the pager proxy and the destination pager, with appropriate substitutions of addresses and keys as explained in more detail below. As shown in Fig. 2, the first field is a clear text field that contains the encryption method indicator EM, pager addressing mode (AM), and user identification number (UID) (sometimes referred to as a PIN, but not to be confused with the password entered by the user to access pager functions), while the second field contains the encrypted session key (SESKey1) and various data referred to as "header data" (HdrData) including the destination header or address (DH) and a message authentication code (MAC), the information in the second field being encrypted by the unique private key of the sending pager (pv.sender) in order

to authenticate the sender, and by a public key corresponding to a private key held by the server (pb.server) in order to protect the contents of this field. The third field contains the message encrypted by the session key.

The various fields illustrated in Fig. 2 may be formatted in any convenient manner permitted or required by the protocol used to package the data in the fields for transmission, but in the illustrated example most or all of the data in at least fields one and two can conveniently be in hexadecimal format. Whenever the drawings illustrate a hexadecimal number, the number ## will be preceded by a "0x" to form 0x##.

The encryption method indicator EM indicates which of the possible encryption methods handled by the server is being used to encrypt the session key and other information in field 2, so that the session key can be recovered and used to decrypt the encrypted message in field 3. As indicated above, possible encryption methods include the RC4 secret key encryption method, which requires the parties to the communication to have a shared secret key that is used for both encryption or decryption, and the RSA public key encryption method, which is the method illustrated in Fig. 2. The indicator itself is simply a number assigned to the encryption method. While any given pager will generally have only a single encryption method stored in memory, it is possible for the pager proxy to be arranged to handle multiple different methods and thus need to have an indication of the type of encryption method, to accommodate different pager systems or legal requirements, particularly if international pager traffic is involved.

The addressing mode (AM) indicates the type of address involved. For example, in the U.S., pager addressing modes are assigned one application header, while e-mail addressing modes are assigned another application header. This indicator may not be necessary in all protocols since the destination header may be unique to a specific type of address, but is included in field 1 as part of the Re-FLEX™ protocol.

The user identification number (UID) included in clear text in field 1 and in encrypted form in field 2, is the unique address assigned to the pager, and is used to indicate the source of the message so as to enable the pager proxy to retrieve the appropriate public decryption key (pb.sender), and for use in authentication of the sender and for display by a receiving pager. Preferably, this number is hard-coded into memory so that it cannot easily be altered, and in current U.S. paging systems is in the form of a ten digit number.

The header data (HdrData) of the second field includes an application header (AH), which included in a field having variable length and string value, the address mode and destination header (AM/DH), the user identification number (UID), which is the same as the one included in field 1, and a message number (MSGNO) and message authentication code (MAC). In addition, e-mail address protocols require a byte indicative of address length to be added where the address mode indicates an e-mail address.

For purposes of the present invention, the message number can be any arbitrary number, although the use of a time-related reference, as allowed by the Re-FLEX protocol, is useful for account tracking or billing purposes, and in addition can be used to ensure that received message is not a recording of a message sent earlier and intercepted by an unauthorized party. For example, the message number has previously been defined as the number of seconds since January 1, 1970.

The message authentication code is a checksum used to verify that the recovered message is identical to the original message, and may be computed using an error correction code function such the cyclic recovery code (CRC) function, with CRCs being used in the illustrated embodiment or, alternatively, by computing a hash or one-way combination of the header data with the message and the session key, using an algorithm such as SHA1. By combining the message with other data to obtain the message

authentication code in a way that can only recreated if the data used to recreate the code is the same as the data originally used to generate the code, the code can be used to authenticate the message, *i.e.*, to verify that the message has not been altered since the time when the code was first generated, as will be described in more detail below. It will be appreciated that the exact form of the message authentication code is not a part of the present invention, and that any message authentication code may be used so long as it can be used to authenticate the message in the manner described below.

The three blocks above the header data in Fig. 2 indicate the source of the data for the various fields. The manner in which the data is combined to form the fields is described in more detail in connection with Figs. 3-10, but the sources of the data may be summarized as (i) information entered by the user, which consists of the message (MSG) and the recipient address which forms the destination header, (ii) information stored in memory, including private and public keys of the pager, a public key of the pager proxy server, an access code which is to be compared with an access code input by the user, the encryption method indicator (EM), the user identification number (UID), and the application header, and (iii) information generated at runtime, *i.e.*, during assembly of the packet header, including the session key (SESKey), the message number (MSGNO), the addressing mode (AM), and the message authentication code (MAC).

The details of the manner in which the data shown in Fig. 2 is assembled by sending pager 1 to form the header shown in Fig. 2 is illustrated in the functional block diagram of Fig. 3, as well as the flowchart of Fig. 7. As illustrated in Fig. 3, the pager 1 includes a user input 20 connected to keys 4 or stylus 5, which supplies the destination header (DH) to a functional block 21 which assembles the header data (HdrData), and to a functional block 22 which computes the message authentication code (MAC). In addition, the user input 20 supplies the message to functional block 28, the output of which is field 3 of the header.

Pager 1 also includes a memory 24 which stores the encryption method (EM), the application header (AH), the user identification number (UID) and the encryption method identifier (EM), which are supplied directly to functional block 23 for inclusion in field 1, the user identification number and application header being also supplied to functional block 21 for inclusion in the header data, which in turn is supplied to functional block 22 for inclusion in the message authentication code. The address mode (AM), which is associated with the destination header (DH) in the header data is generated by an address mode generator 25 which can be in the form of a look-up table, device that reads a particular identifying bit in the destination header, or other device, and the message number can be generated by a counter, timer, or other device 26 depending on the nature of the message number. Finally, the session key (SESKey1) for this embodiment of the invention is an eight character string generated by a random or pseudorandom number generator 27, which supplies the session key to functional block 28 for use in encrypting the message (MSG), to functional block 22 for inclusion in the message authentication code, and to functional block 29 for encryption together with the header data by the private key of the sender. The output of functional block 29 is supplied to functional block 30 for encryption by the public key of the server, the output of block 30 serving as field 2 of the header for the packet transmitted by the sending pager.

It will be appreciated by those skilled in the art that any of the functional blocks and data or number generators illustrated in Fig. 3, or in Figs 4-6, may be implemented either by hardware or software, and that while distinguishable by function, the functions may be carried out using common subroutines, hardware, or software.

Turning to Fig. 4, the pager proxy 7 includes a database of public keys corresponding to the unique public keys of pagers registered with the encryption service provider that operates the proxy server. The database is accessed by functional block 31 according to the clear text user identification number (UID) present in the header of a packet forwarded to the pager proxy by the network operations center. Field 2 is

decrypted by functional block 32 using the private key of the server (pv.server) and by functional block 33 using the public key of the sender (pb.sender) to recover the session key, and the user identification number (UID) recovered from field 2 is compared by functional block 34 with the user identification number of field 1 to verify the authenticity of field 2 and recover the session key (SESKey1). A functional block 35 then uses the session key to decrypt the message (MSG).

The message recovered by the pager proxy is authenticated in functional block 37, by comparing the message authentication code recovered from field 2 with the output of a functional block 36 that computes the message authentication code based on the destination header (DH), application header (AH), user identification number (UID), message number (MSGNO), and session key (SESKey1) recovered from field 2, and the message recovered from field 3. The message, session key, and header data (HdrData) are then made available by functional block 38 to an encryption or repackaging module, illustrated in Fig. 5, for repackaging in a way that will enable decryption by a destination pager.

As shown in Fig. 5, the application header (AH) and message number (MSGNO) received from functional block 38 is provided to functional blocks 41 and 42 for inclusion in the header data and message authentication code, while the address mode (AM) and encryption method (EM) obtained from field 1 of the packet received from the sender is passed to functional block 43 or regenerated for inclusion as clear text in the packet header. In order to permit decryption and authentication of the repackaged header by the receiving pager, however, the destination header (DH) and user identification number (UID) are swapped, so that the original destination header is supplied by the pager proxy to functional blocks 41, 42, and 43 as the user identification number (UID), and the original user identification number are supplied to functional blocks 41 and 42 as the destination header (DH). Functional block 42 generates a message authentication code based on the new destination header (DH), application header (AH), user

identification number (UID), message number (MSGNO), while a new session key (SESKey2) is generated by functional block 44 in the same manner as functional block 27 shown in Fig. 3, and the resulting message authentication code (MAC) together with the new session key and header data from functional block 41 are encrypted by functional block 45 using the private key of the server (pv.server) before being sealed by functional block 46 using the public key of the destination pager (pb.recipient) and included in the header as field 2. Functional block 47 receives the message and new session key and re-encrypts the message using the new session key and an algorithm such as RC4 to generate field 3, fields 1-3 being assembled into a packet 50 for transmission to the destination pager 2 via the network operations center 3.

Again, those skilled in the art will appreciate that all of the functional blocks illustrated as being present in the proxy server and/or proxy authentication module may be implemented as software, hardware, or a combination of software and hardware, and may be varied depending on the encryption method and requirements of the pager protocol.

In addition, those skilled in the art will appreciate that the illustrated embodiment could be modified by eliminating the session key and instead using public key encryption of the message. Alternatively, instead of having the pager proxy perform any decryption of the message, the original session key could simply be re-encrypted by the pager proxy using at least the public key of the destination pager as described above, or a secret key shared with the destination pager, in which the encrypted message would simply be forwarded to the destination pager unit with the session key re-encrypted so that it can be recovered by the destination pager. While neither of these options is currently preferred because elimination of the session key leaves transmissions vulnerable to recording, and elimination of message decryption by the pager proxy makes message authentication more difficult, they should nevertheless be considered to be within the scope of the invention.

Turning to Fig. 6, the destination pager 2 includes functional blocks mirroring those of the server for decrypting messages and authenticating packets received from the pager proxy 7 via the network operations center 3. These include functional block 51 for retrieving the server public key (pb.server) from memory, functional blocks 52 and 53 for decrypting the field 2 using the recipient private key (pv.recipient) and the server public key, functional block 54 for comparing the user identification number recovered from field 2 with the user identification number in field 1, functional block 56 for decrypting the message (MSG) using the session key (SESKey2) recovered from field 2, and functional blocks 57 and 58 for generating a message authentication code and comparing it with the message authentication code recovered from field 2. It will be noted that functional block 57 may also be used to generate a message authentication code for an outgoing message, avoiding duplication of the hardware or software which performs this function.

Finally, destination pager 2 includes a functional block 59 for displaying the message (MSG) and destination header (DH) corresponding to the user identification number of the sending pager, and for alerting the user as necessary that a message has been received. The display is identical to that used for an unencrypted message, and thus the decryption operation is entirely transparent to the user.

The method steps that implement the functions illustrated in Figs. 3-6 are as follows:

First, as shown in Fig. 7, upon input of a message and destination address by the user of a pager (step 100), which may follow the input and verification of a password (not shown), a message number, address mode, and session key are generated (step 110) and the encryption method identifier, application header, user identification number, server public key, and sender private key are retrieved from memory (step 120). The encryption method identifier, address mode, and user identification number are included in field 1 (step 130), a message authentication code based on the destination header, application

header, user identification number, message number, message, and session key is computed (step 140), and the application header, user identification number, destination header, message number, message authentication code, and session key are encrypted by the private key of the sending pager (step 150) and then by the public key of the pager proxy (step 160) to obtain field 2 of the packet header. Finally, the message is encrypted by the session key (step 170) to obtain field 3, and the packet header is transmitted via the network operations center to the pager proxy (step 180).

Upon receipt by the pager proxy, as shown in Fig. 8, the public key of the sending pager is retrieved based on the user identification number in field 1 (step 200), and field 2 of the packet is decrypted by the private key of the server (step 210) and then by the public key of the sending pager (step 220) based on the encryption method identified by the identifier in field 1. Authentication of the sender is provided by comparing the user identification number recovered from field 2 with the user identification number in field 1 (step 230), the message included in field 3 is decrypted using the session key recovered from field 2 (step 240), and authentication of the message is provided by generating a message authentication code based on the destination header, application header, user identification number, message number, and session key recovered from field 2 together with the decrypted message (step 250), and by then comparing the computed message authentication code with the message authentication code recovered from field 2 (step 260).

As illustrated in Fig. 9, after authenticating the information contained in field 2, the proxy server generates a new session key (step 300), encrypts the message using the new session key (step 310), assigns the original user identification as the new destination header and the original destination header as the new user identification number, computes a new message authentication code (step 330), encrypts the address header, message number, new user identification number, new destination header, new session key, and new message authentication code using the private key of the server (step 340),

encrypts the result of step 340 using the public key of the destination pager (step 350), and assembles the header and packet for RF transmission to the destination pager via the network operations center (step 360).

As illustrated in Fig. 10, upon receipt by the destination pager, as shown in Fig. 8, the public key of the pager proxy server is retrieved based on the user identification number in field 1 (step 400), and field 2 of the packet is decrypted by the private key of the destination pager (step 410) and then by the public key of the pager proxy server (step 420) based on the encryption method identified by the identifier in field 1. Authentication of the sender is provided by comparing the user identification number recovered from field 2 with the user identification number in field 1 (step 430), the message included in field 3 is decrypted using the session key recovered from field 2 (step 440), and authentication of the message is provided by computing a message authentication code based on the destination header, application header, user identification number, message number, and session key recovered from field 2 together with the decrypted message (step 450), and by then comparing the computed message authentication code with the message authentication code recovered from field 2 (step 460). Finally, after authentication of the user identification number and message, the user is alerted that a message has been received and the decrypted message and information contained in the destination header are displayed at the request of the user (step 470).

Having thus described a preferred embodiment of the invention in sufficient detail to enable those skilled in the art to practice the invention, it is nevertheless anticipated that numerous variations and modifications of the invention will occur to those skilled in the art, and it is intended that all such variations and modifications be included within the scope of the invention. For example, although the preferred embodiment of the invention has the pager proxy re-package the message by first decrypting it, and then re-encrypting it using a new session key, it is also within the scope of the invention to have the pager proxy decrypt only the session key and re-encrypt the same session key using

the public key or shared secret key of the destination pager. Accordingly, it is intended that the above description not be taken as limiting, but rather that it be defined solely by the appended claims.

I claim:

1. A system for adding encryption services to an existing pager network, the pager network including a network operations center which provides a means for receiving an alphanumeric message from any of a plurality of handheld pager units and forwarding the alphanumeric message to another of the plurality of handheld pager units, at least two of said pager units comprising:

means for inputting an alphanumeric message and a destination address;

means for including the alphanumeric message in a packet for transmission to the destination address by wireless transmission via the network operations center;

means for receiving an alphanumeric message from the network operations center; and

means for displaying the alphanumeric message received from the network operations center,

wherein the system for adding encryption services comprises:

means in at least one of said pager units for encrypting a message and transmitting the encrypted message via the network operations center to another of said pager units;

means in said another one of said pager units for decrypting and displaying the encrypted message; and

a pager proxy server including means for receiving a packet containing the encrypted message that has been sent to the network operations center, decrypting at least a portion of the packet, and re-encrypting said portion of the packet for delivery to said another of said pager units via said network operations center.

2. A system as claimed in claim 1, wherein said means for encrypting the message comprises means for encrypting the message by a secret key.

3. A system as claimed in claim 2, wherein said secret key is a first session key generated by a sending pager unit, said sending pager unit further comprising means for encrypting said first session key by a public key corresponding to a private key held by the pager proxy server so that the session key can be recovered only by the paging proxy server.

4. A system as claimed in claim 3, wherein said sending pager unit further comprises means for encrypting at least the first session key by a private key of the sending pager unit, and wherein said pager proxy server includes means for retrieving a public key corresponding to the private key of the sending pager unit for use as a first level authentication of the sending pager unit.

5. A system as claimed in claim 4, further comprising means for appending a unique user identification number of the sending pager unit to the header in clear text form, said user identification number being hard-coded into the sending pager unit.

6. A system as claimed in claim 5, wherein said means for encrypting at least the session key by a private key of the sending pager unit also encrypts the user identification number of the sending pager unit, and said paging proxy server includes means for decrypting the encrypted user identification number together with the first session key and comparing it with the clear text user identification number in order to authenticate the contents of the field containing the encrypted user identification number and first session key.

7. A system as claimed in claim 4, wherein the sending pager unit further comprises means for generating a first message authentication code based on various header data and the message and encrypting the various information together with the session key and the first message authentication code using the private key of the sending pager unit, and wherein the pager proxy server further comprises means for decrypting the various header

data, first message authentication code, and session key using a public key corresponding to the private key of the sending pager unit, decrypting the message using the session key, generating a second message authentication code based on the message and various header data, and comparing the first message authentication code with the second message authentication code in order to authenticate the message.

8. A system as claimed in claim 7, wherein said message authentication code is an error correction code function.

9. A system as claimed in claim 7, wherein said various header data includes at least a user identification number of the sending pager and a destination header corresponding to the input address of the destination pager.

10. A system as claimed in claim 9, wherein said various header data further includes a message number and application header.

11. A system as claimed in claim 4, wherein the sending pager further comprises means for adding an encryption method identifier in clear text to the packet header.

12. A system as claimed in claim 4, wherein an encryption algorithm used to encrypt the first session key is a public-private key encryption algorithm.

13. A system as claimed in claim 4, wherein said secret key is a first session key generated by a sending pager unit and said first session key is encrypted by a stream cipher that uses a shared secret key.

14. A system as claimed in claim 2, wherein said sending pager unit further comprises means for generating an address mode and appending the address mode in clear text to the packet header.

15. A system as claimed in claim 14, wherein said address mode indicates an address type selected from the group consisting of pager address types and e-mail address types, and wherein the pager proxy server is connected to a computer network gateway server and includes means for re-packaging said message in an e-mail packet and transmitting the e-mail packet via said computer network server to an e-mail address.

16. A system as claimed in claim 15, further comprising means for receiving e-mail packets from said computer network gateway server, and re-packaging said e-mail packets for transmission to the destination pager unit via said network operation center, and means for repackaging packets received from the network operations center for forwarding to an e-mail server.

17. A system as claimed in claim 1, wherein said means included in the pager proxy server for decrypting at least a portion of the packet includes means for decrypting, using a secret key, a portion of the packet containing a first session key used by a sending pager unit to encrypt said portion of the packet.

18. A system as claimed in claim 17, wherein said pager proxy server further includes means for decrypting said message using said first session key, means for generating a second session key, and means for re-encrypting the message using the second session key.

19. A system as claimed in claim 18, wherein said means for re-encrypting said portion of the packet includes means for encrypting the second session key by a secret key.

20. A system as claimed in claim 19, wherein said means for encrypting said portion of the packet by a secret key includes means for re-encrypting the second session key by a public key corresponding to a private key of a destination pager unit.

21. A system as claimed in claim 20, wherein said means for encrypting said portion of the packet by a secret key further includes means for, before re-encrypting the second session key by the public key corresponding to a private key of the destination pager, encrypting the second session key and various additional data by a private key of the pager proxy server.

22. A system as claimed in claim 21, wherein said additional data includes a second user identification number, said second user identification number corresponding to a first destination header included in said decrypted portion of the packet received from the sending pager unit, and wherein said destination paging unit includes means for comparing said second user identification number encrypted with said second session key to a clear text version of the second user identification number received from the pager proxy server in order to authenticate the pager proxy server.

23. A system as claimed in claim 22, wherein said additional data includes a second destination header corresponding to the first user identification number, and wherein said second pager unit includes means for displaying information included in said second destination header in order to indicate an address of the sending pager unit.

24. A system as claimed in claim 22, wherein said additional data includes a second destination header corresponding to the first user identification number, a message number recovered from said decrypted portion of the packet received from the sending pager unit, and an application number.

25. A system as claimed in claim 22, wherein said pager proxy server further comprises means for generating a message authentication code based on said message, said second session key, and said additional data, and said destination pager unit includes means for recovering said additional data and computing a message authentication code based on the additional data, said second session key, and said message in order to authenticate said message.

26. An encryption method according to which encryption services may be added to an existing two-way wireless pager network, the pager network including a network operations center which provides a means for receiving an alphanumeric message from any of a plurality of handheld pager units and forwarding the alphanumeric message to another of the plurality of handheld pager units, comprising the steps of:

causing one of said pager units to perform the steps of encrypting a message, including the encrypted message in a wireless transmission packet, and transmitting the encrypted message from said one of said pager units to a pager proxy server via the network operations center;

causing the pager proxy server to perform the steps of receiving the encrypted message and repackaging it for transmission to another of said pager units via the network operations center; and

causing said another of said pager units to perform the steps of decrypting and displaying the encrypted message.

27. A method as claimed in claim 26, wherein the step of encrypting the message comprises the step of encrypting the message by a secret key corresponding to a secret key of the pager proxy server so that the session key can only be recovered by the paging proxy server.

28. A method as claimed in claim 26, wherein said secret key is a first session key generated by a sending pager unit, and wherein said sending pager unit further performs the step of encrypting said first session key by a public key corresponding to a private key held by the pager proxy server.

29. A method as claimed in claim 27, wherein said sending pager unit further performs the step of encrypting at least the first session key by a private key of the sending pager unit, and wherein said pager proxy server performs the step of retrieving a public key corresponding to the private key of the sending pager unit for use as a first

level authentication of the sending pager unit.

30. A method as claimed in claim 29, further comprising of the step of appending a unique user identification number of the sending pager unit to the header of the transmission to the paging proxy server in clear text form, said user identification number being hard-coded into the sending pager unit.

31. A method as claimed in claim 30, wherein said step of encrypting at least the session key by a private key of the sending pager unit includes the step of encrypting the user identification number of the sending pager unit, and said paging proxy server further performs the steps of decrypting the encrypted user identification number together with the first session key and comparing it with the clear text user identification number in order to authenticate the contents of the field containing the encrypted user identification number and first session key.

32. A method as claimed in claim 29, wherein the sending pager unit further performs the step of computing a first message authentication code based on various header data and the message and encrypting the various information together with the session key and the first message authentication code using the private key of the sending pager unit, and wherein the pager proxy server further performs the steps of decrypting the various header data, first message authentication code, and session key using a public key corresponding to the private key of the sending pager unit, decrypting the message using the session key, generating a second message authentication code based on the message and various header data, and comparing the first message authentication code with the second message authentication code in order to authenticate the message.

33. A method as claimed in claim 32, wherein said message authentication code is an error correction code function.

34. A method as claimed in claim 32, wherein said various header data includes at least the user identification number of the sending pager and a destination header corresponding to the input address of the destination pager.

35. A method as claimed in claim 34, wherein said various header data further includes a message number and application header.

36. A method as claimed in claim 34, wherein the sending pager further performs the step of adding an encryption method identifier in clear text to the packet header.

37. A method as claimed in claim 29, wherein an encryption algorithm used to encrypt the first session key is a public-private key encryption algorithm.

38. A method as claimed in claim 27, wherein said secret key is a first session key generated by a sending pager unit and said first session key is encrypted by a stream cipher that uses a shared secret key.

39. A method as claimed in claim 37, wherein said sending pager unit further performs the step of generating an address mode and appending the address mode in clear text to the packet header.

40. A method as claimed in claim 39, wherein said address mode indicates an address type selected from the group consisting of pager address types and e-mail address types, and wherein the pager proxy server is connected to a computer network gateway server and further performs the step of re-packaging said message in an e-mail packet and transmitting the e-mail packet via said computer network server to an e-mail address.

41. A method as claimed in claim 40, further performs the steps of receiving e-mail packets from said computer network gateway server, and re-packaging said e-mail

packets for transmission to the destination pager unit via said network operation center.

42. A method as claimed in claim 26, wherein said step of repackaging the encrypted message for transmission includes the step of causing the pager proxy server to encrypt, using a secret key, a portion of the packet containing a first session key used by a sending pager unit to encrypt said portion of the packet.

43. A method as claimed in claim 42, wherein said pager proxy server further performs the steps of decrypting said message using said first session key, generating a second session key, and re-encrypting the message using the second session key.

44. A method as claimed in claim 43, wherein said pager proxy server further performs the step of encrypting the second session key by a secret key.

45. A method as claimed in claim 44, wherein said step of encrypting said portion of the packet by a secret key includes the step of re-encrypting the second session key by a public key corresponding to a private key of a destination pager unit.

46. A method as claimed in claim 45, wherein said step of encrypting said portion of the packet by a secret key further includes the step of, before re-encrypting the second session key by the public key corresponding to a private key of the destination pager, encrypting the second session key and various additional data by a private key of the pager proxy server.

47. A method as claimed in claim 46, wherein said additional data includes a second user identification number, said second user identification number corresponding to a first destination header included in said decrypted portion of the packet received from the sending pager unit, and wherein said destination paging unit perform the step of comparing said second user identification number encrypted with said second session key

to a clear text version of the second user identification number received from the pager proxy server in order to authenticate the pager proxy server.

48. A method as claimed in claim 47, wherein said additional data includes a second destination header corresponding to the first user identification number, and wherein said second pager unit performs the step of displaying information included in said second destination header in order to indicate an address of the sending pager unit.

49. A method as claimed in claim 47, wherein said additional data includes a second destination header corresponding to the first user identification number, a message number recovered from said decrypted portion of the packet received from the sending pager unit, and an application number.

50. A method as claimed in claim 47, wherein said pager proxy server further performs the step of computing a message authentication code based on said message, said second session key, and said additional data, and said destination pager unit further performs the step of recovering said additional data and computing a message authentication code based on the additional data, said second session key, and said message in order to authenticate said message.

51. A two-way alphanumeric pager unit, comprising:

means for inputting a message and a destination address;

means for generating a session key;

means for encrypting the message using the session key;

means for protecting the session key so that it can only be recovered by a pager proxy server;

means for transmitting the message via a wireless pager network to the pager proxy server;

means for receiving an encrypted message transmitted via the wireless pager network from the pager proxy server;

means for decrypting an encrypted session key appended to the message;

means for decrypting the encrypted message transmitted from the pager proxy server using the decrypted session key; and

means for displaying the message.

52. A pager unit as claimed in claim 51, wherein said means for protecting the session key comprises means for encrypting the session key by a secret key.

53. A pager unit as claimed in claim 52, wherein said secret key is a first session key generated by the pager unit, said sending pager unit further comprising means for encrypting said first session key by a public key corresponding to a private key held by the pager proxy server.

54. A pager unit as claimed in claim 53, further comprising means for appending a unique user identification number of the pager unit to the header in clear text form, said user identification number being hard-coded into the pager unit.

55. A pager unit as claimed in claim 54, wherein said means for encrypting at least the session key by a secret key also encrypts the user identification number of the sending pager unit, said encrypted user identification number being compared by the pager proxy server with a clear text version of the user identification number transmitted with a packet header in order to authenticate the pager unit.

56. A pager unit as claimed in claim 55, wherein the pager unit further comprises means for computing a message authentication code based on various header data and the message, and means for encrypting the various information together with the session key and the message authentication code using a private key of the sending pager unit in order to provide a means for authentication by the pager proxy of the message.

57. A pager unit as claimed in claim 56, wherein said message authentication code is an error correction code function.

58. A pager unit as claimed in claim 57, wherein said various header data includes at least the user identification number of the pager unit and a destination header corresponding to the input address of a destination pager.

59. A pager unit as claimed in claim 58, wherein said various header data further includes a message number and application header.

60. A pager unit as claimed in claim 52, wherein the pager unit further comprises means for adding an encryption method identifier in clear text to a packet header.

61. A pager unit as claimed in claim 60, wherein an encryption algorithm used to encrypt the first session key is a public-private key encryption algorithm.

62. A pager unit as claimed in claim 60, wherein said secret key is a first session key generated by a sending pager unit and said first session key is encrypted by a stream cipher that uses a shared secret key.

63. A pager unit as claimed in claim 62, wherein said pager unit further comprises means for generating an address mode and appending the address mode in clear text to the packet header.

64. A pager unit as claimed in claim 62, wherein said address mode is selected from the group consisting of pager address types and e-mail address types, and wherein the pager proxy server is connected to a computer network server and includes means for re-packaging said message in an e-mail packet and transmitting the e-mail packet via said computer network server to an e-mail address.

65. A pager proxy server, comprising:

means for receiving a message encrypted by a session key, the session key being encrypted and appended to the encrypted message, from a network operations center of a pager network;

means for recovering the session key using a secret key of the server;

means for authenticating the sender of the message; and

means for re-transmitting the message encrypted by a session key in a manner which enables decryption of the message only by a holder of a second secret key.

66. A server as claimed in claim 65, wherein said means for re-transmitting the message comprises means for decrypting the message using the first session key, re-encrypting the message using a second session key, and encrypting the second session key.

67. A server as claimed in claim 66, wherein said first secret key is a private key held by the pager proxy server.

68. A server as claimed in claim 67, further comprising means for retrieving a public key corresponding to a private key of a sending pager unit for use as a first level authentication of the sending pager unit.

69. A server as claimed in claim 68, further comprising means for decrypting the a user identification number of the sending pager unit together with the session key and comparing it with a clear text user identification number in order to authenticate the contents of the field containing the encrypted user identification number and session key.

70. A server as claimed in claim 69, further comprising means for decrypting various header data, a first message authentication code, and a session key using a public key corresponding to the private key of the sending pager unit, decrypting the message using

the session key, generating a second message authentication code based on the message and various header data, and comparing the first message authentication code with the second message authentication code in order to authenticate the message.

71. A server as claimed in claim 70, wherein said message authentication code is an error correction code function.

72. A server as claimed in claim 70, wherein said various header data includes at least the user identification number of the sending pager and a destination header corresponding to the input address of the destination pager.

73. A server as claimed in claim 72, wherein said various header data further includes a message number and application header.

74. A server as claimed in claim 73, wherein said encryption method is a public-private key encryption algorithm.

75. A server as claimed in claim 73, wherein said encryption method is RC4 secret key encryption.

76. A server as claimed in claim 72, further comprising means for receiving e-mail packets from said computer network server, and re-packaging said e-mail packets for transmission to the destination pager unit via said network operation center.

77. A system for adding encryption services to an existing pager network, the pager network including a network operations center which provides a means for receiving an alphanumeric message from any of a plurality of handheld pager units and forwarding the alphanumeric message to another of the plurality of handheld pager units, at least one of said pager units comprising:

means for inputting an alphanumeric message and a destination address;

means for including the alphanumeric message in a packet for transmission to the destination address by wireless transmission via the network operations center;

means for receiving an alphanumeric message from the network operations center; and

means for displaying the alphanumeric message received from the network operations center,

wherein the system for adding encryption services comprises:

means in at least one of said pager units for decrypting and displaying an encrypted message; and

a pager proxy server including means for receiving a packet containing the encrypted message, decrypting at least a portion of the packet, and re-encrypting said portion of the packet for delivery to said at least one of said pager units via said network operations center.

78. An alphanumeric pager unit, comprising:

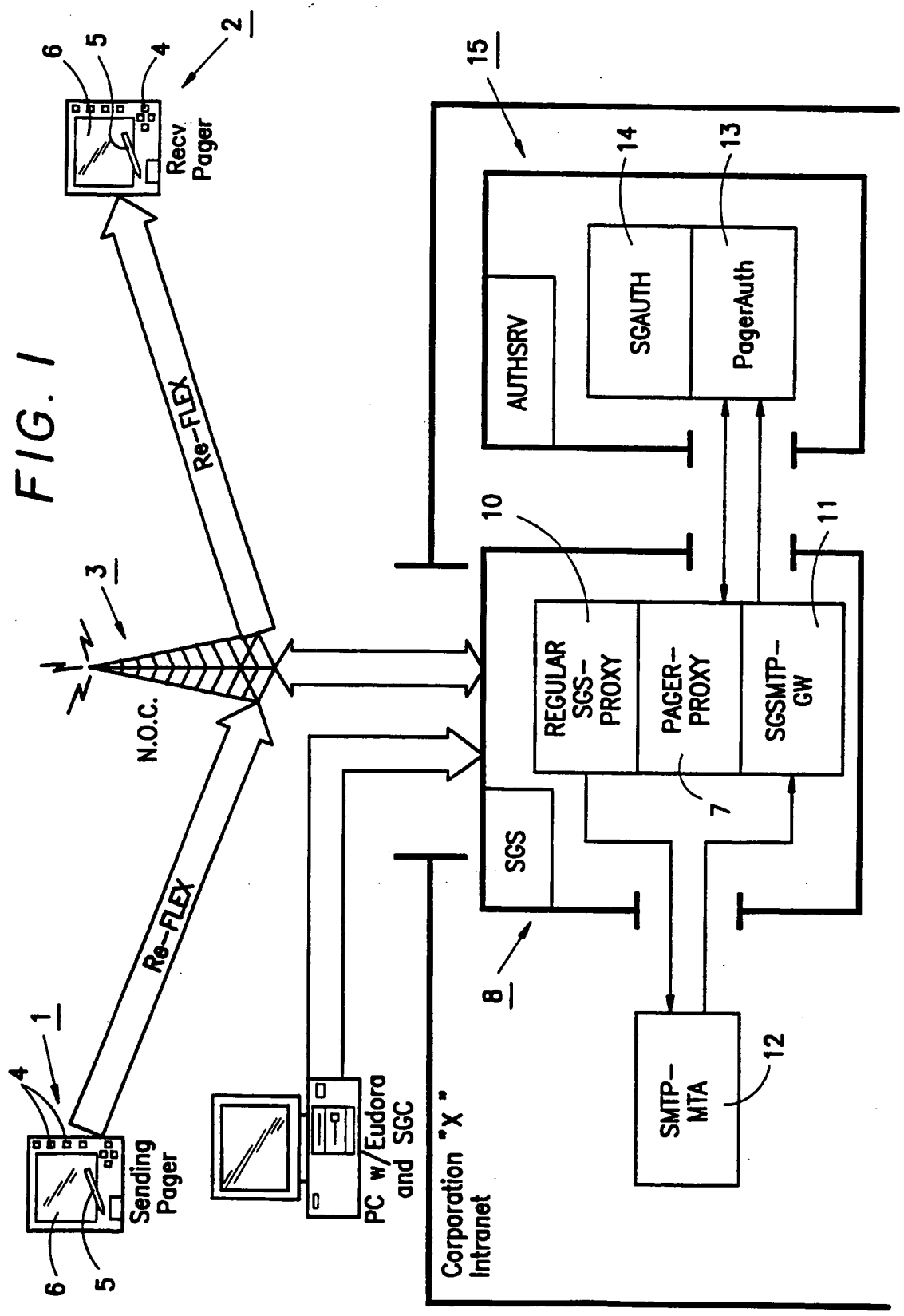
means for receiving an encrypted message transmitted via a wireless pager network from a pager proxy server;

means for decrypting an encrypted session key appended to the message;

means for decrypting the encrypted message transmitted from the pager proxy server using the decrypted session key; and

means for displaying the message.

FIG. 1



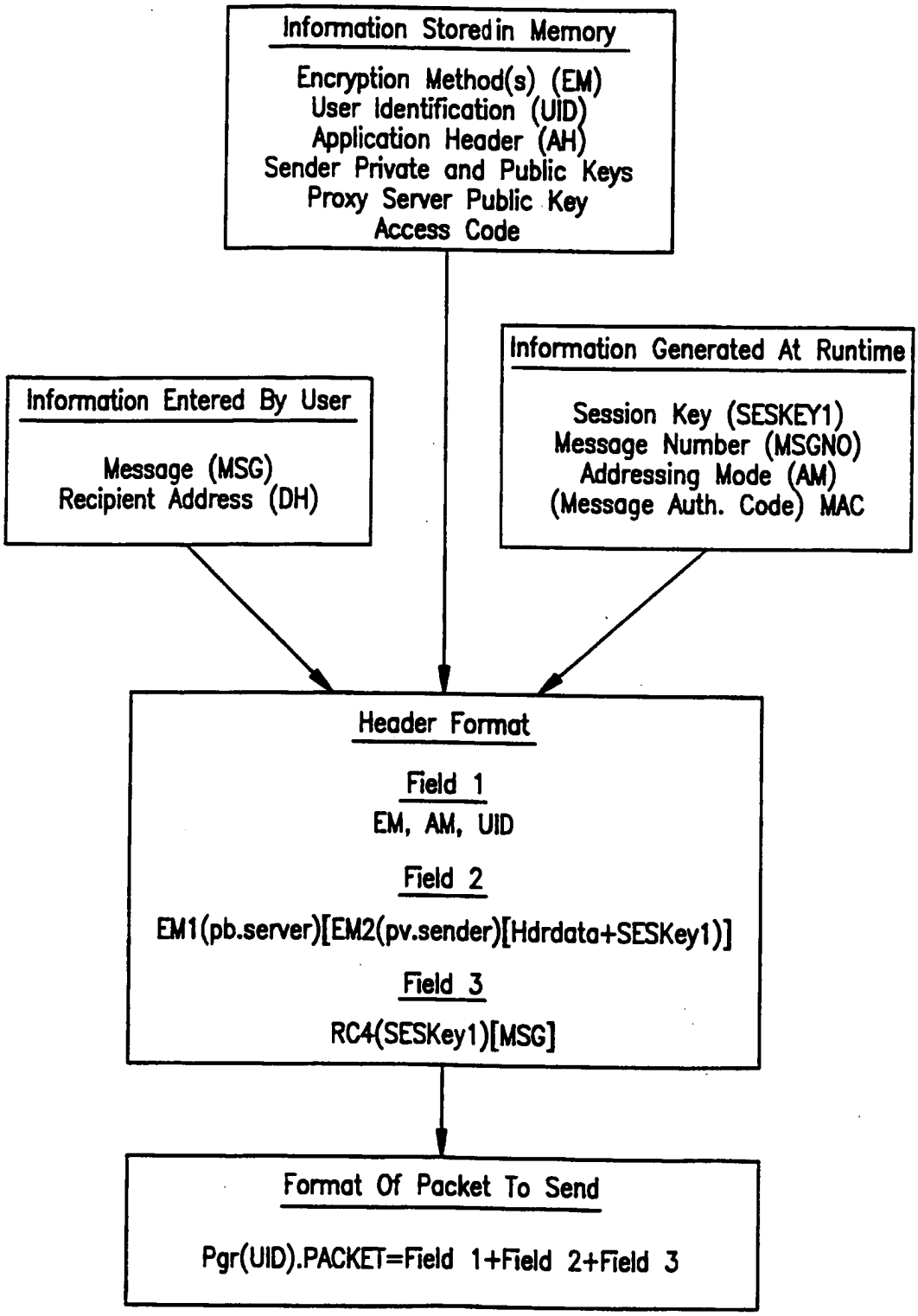
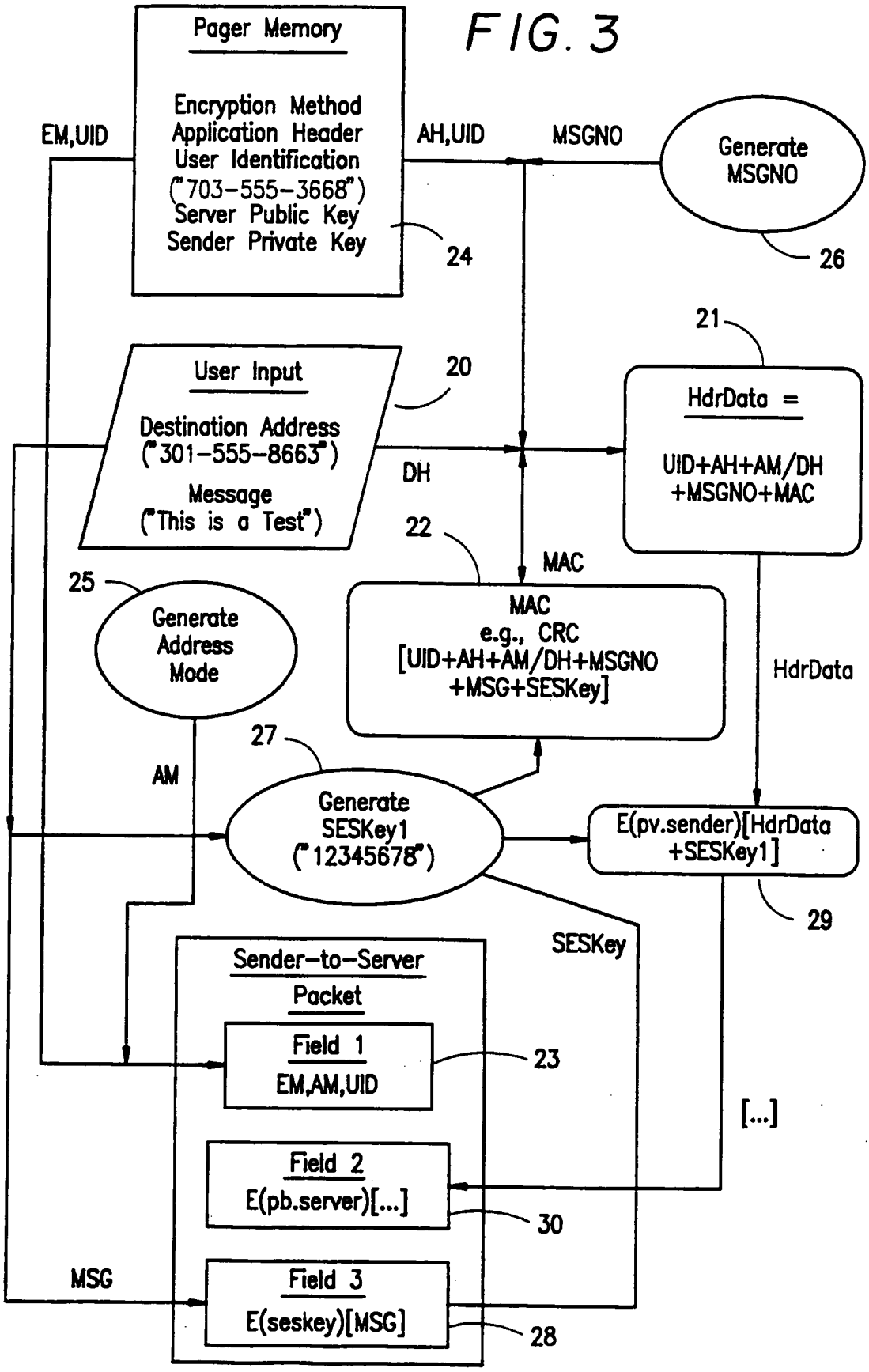


FIG. 2

FIG. 3



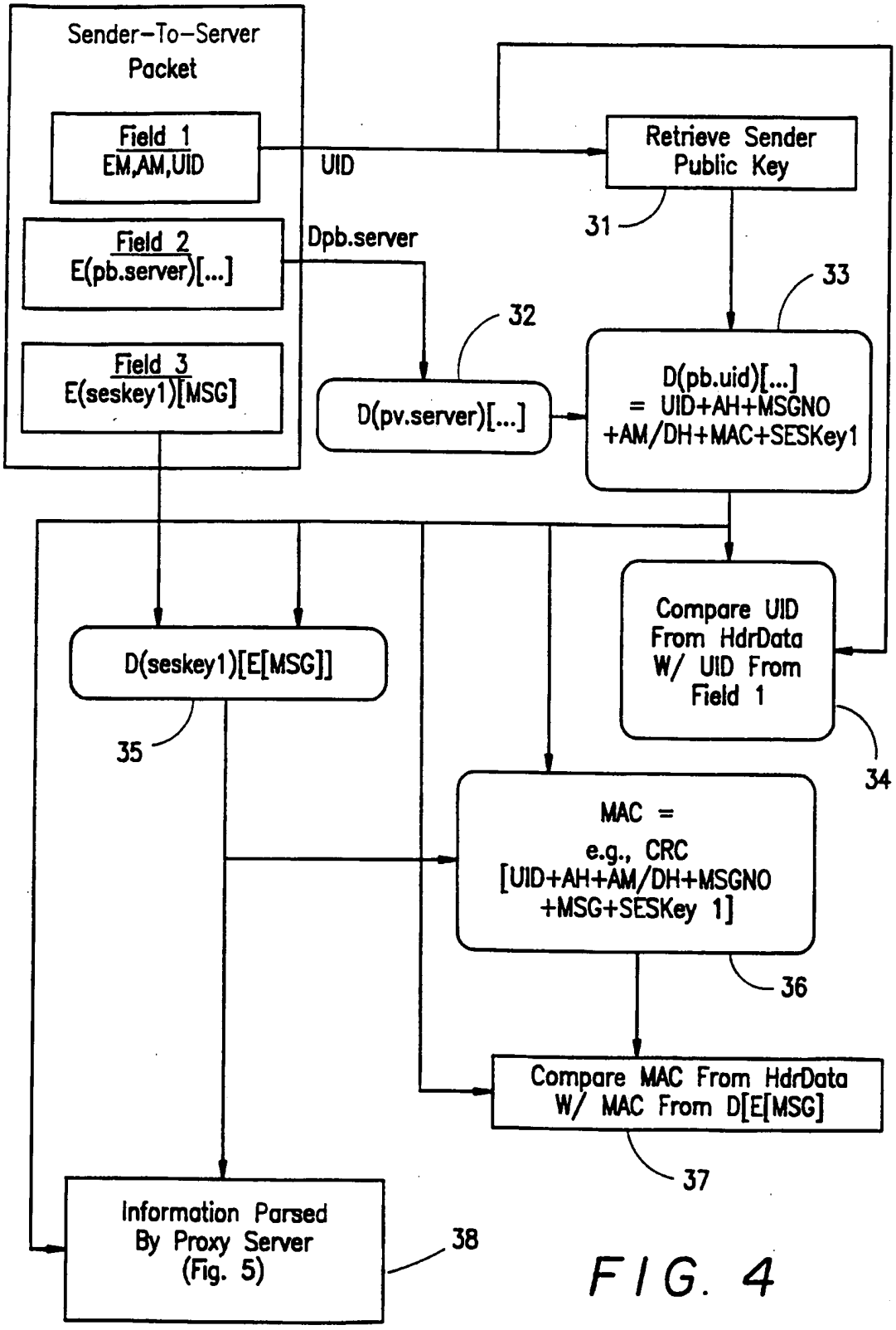


FIG. 4

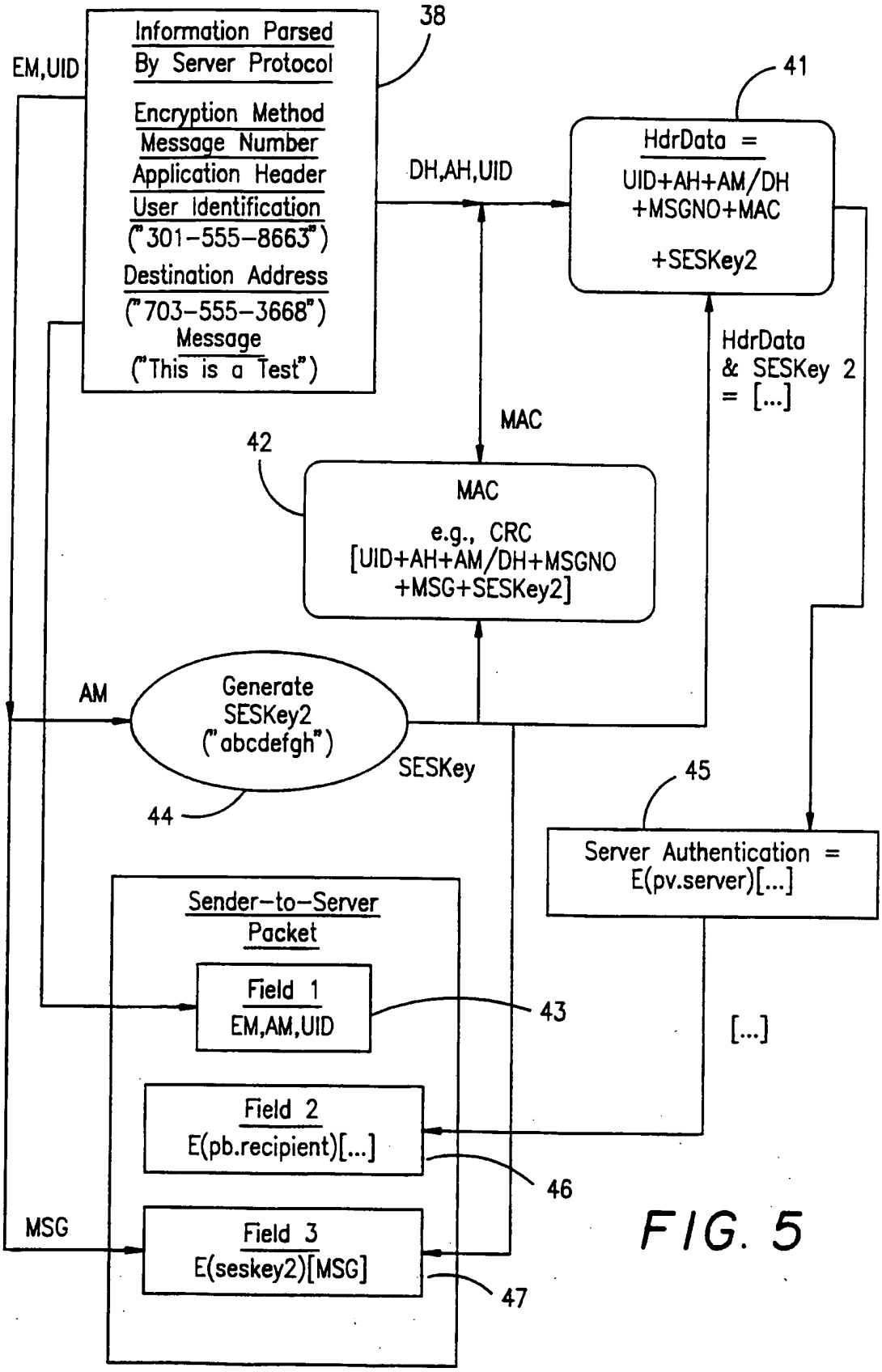


FIG. 5

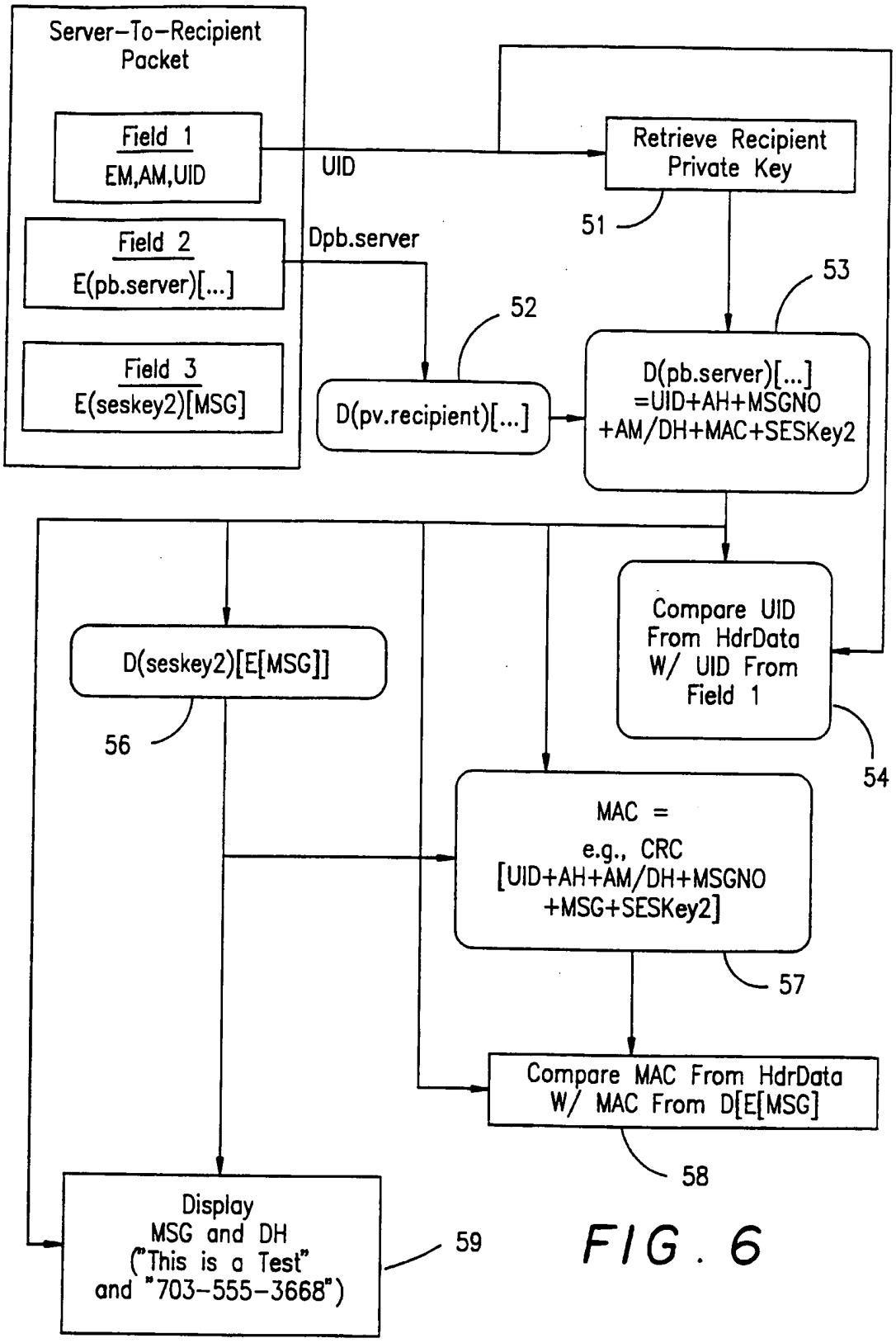


FIG. 6

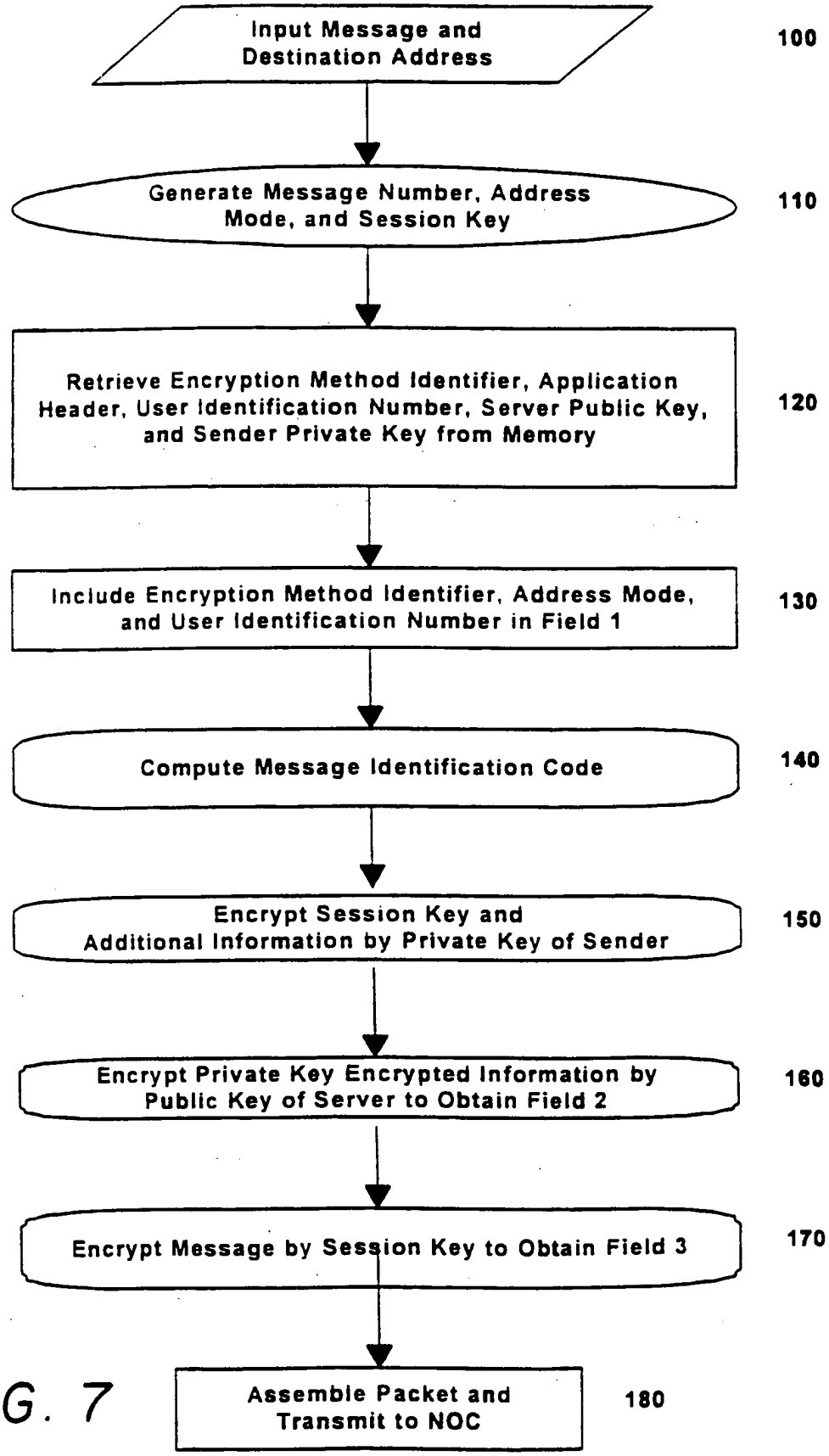
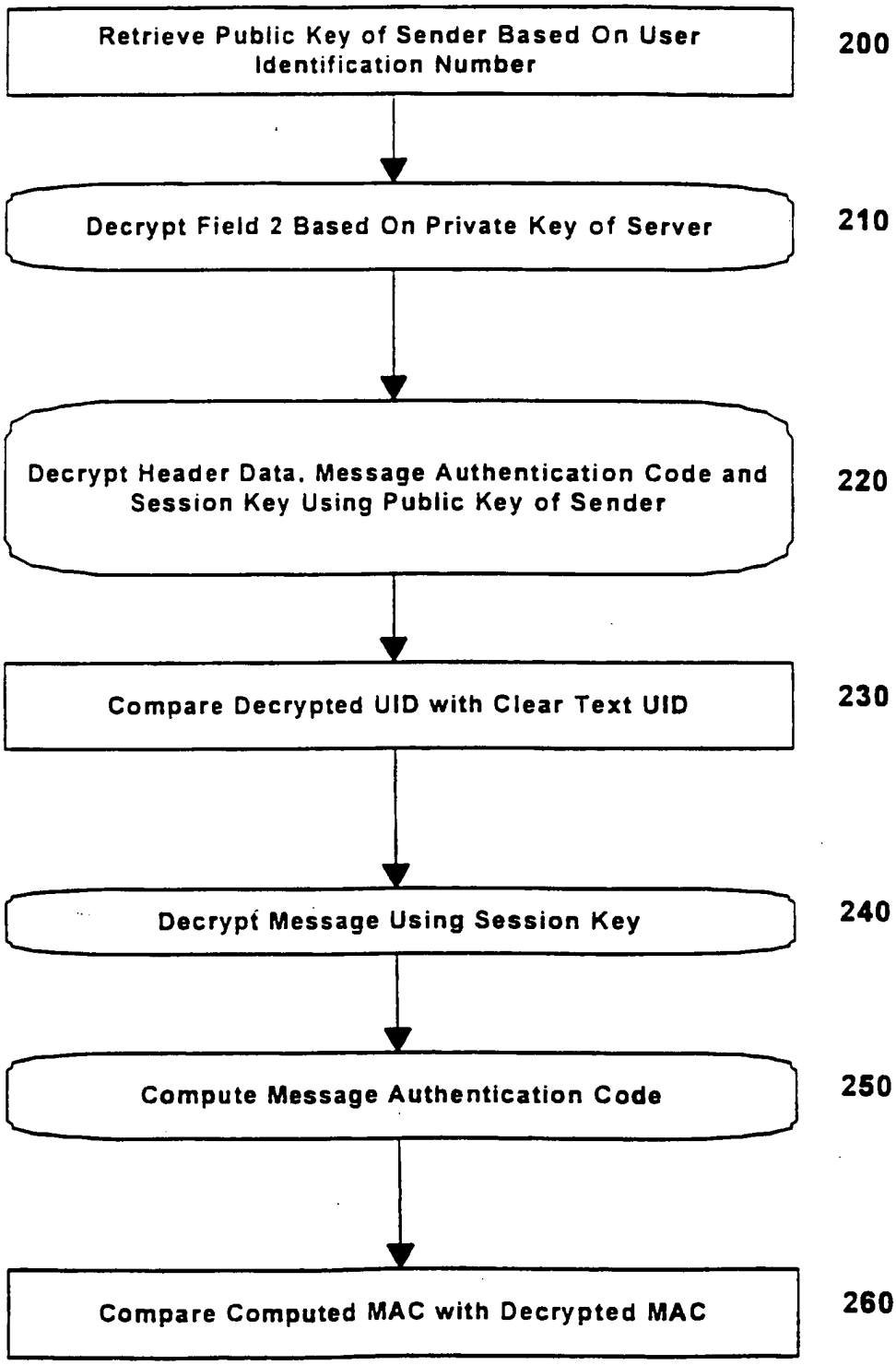


FIG. 7

FIG. 8



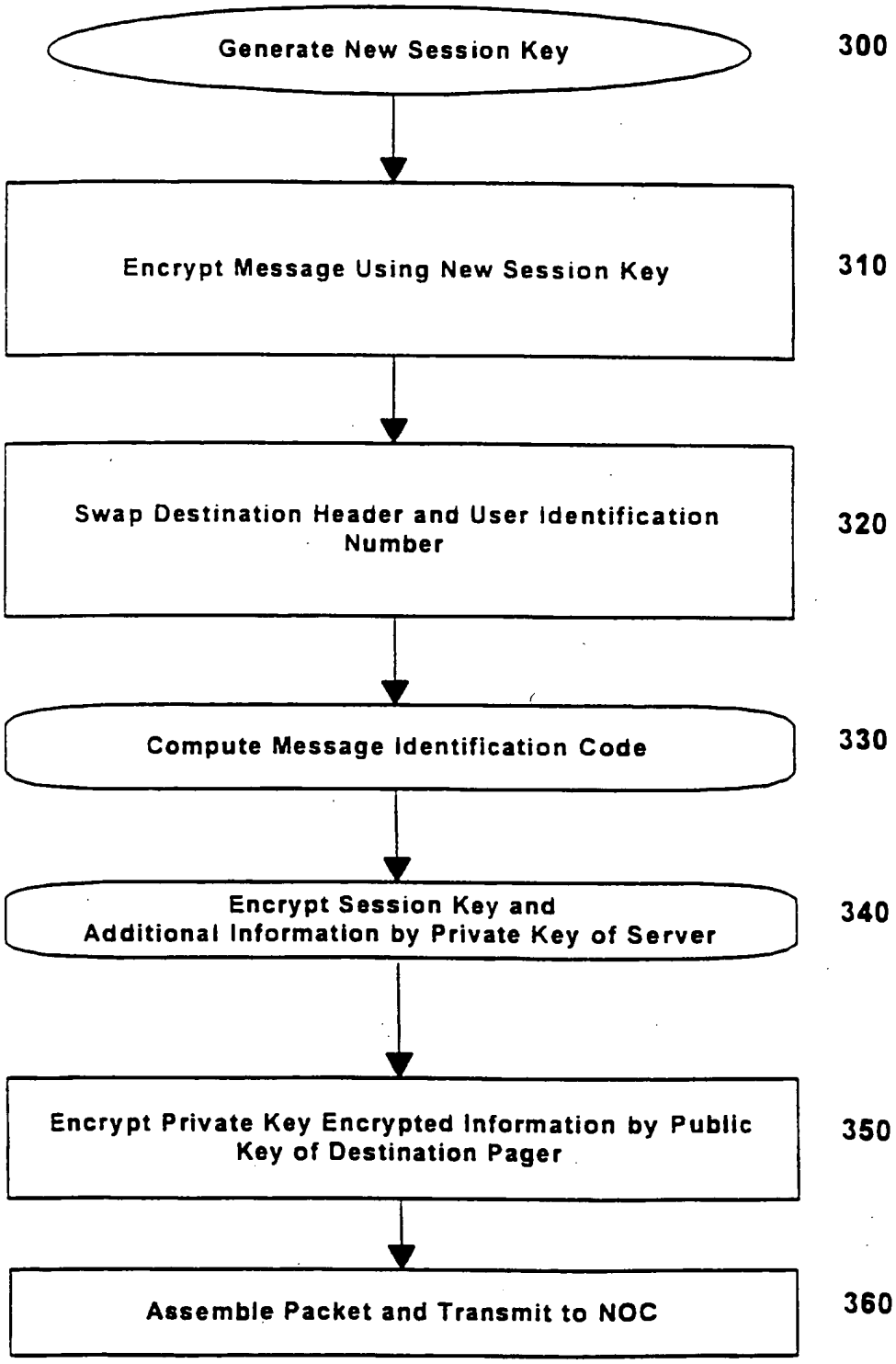
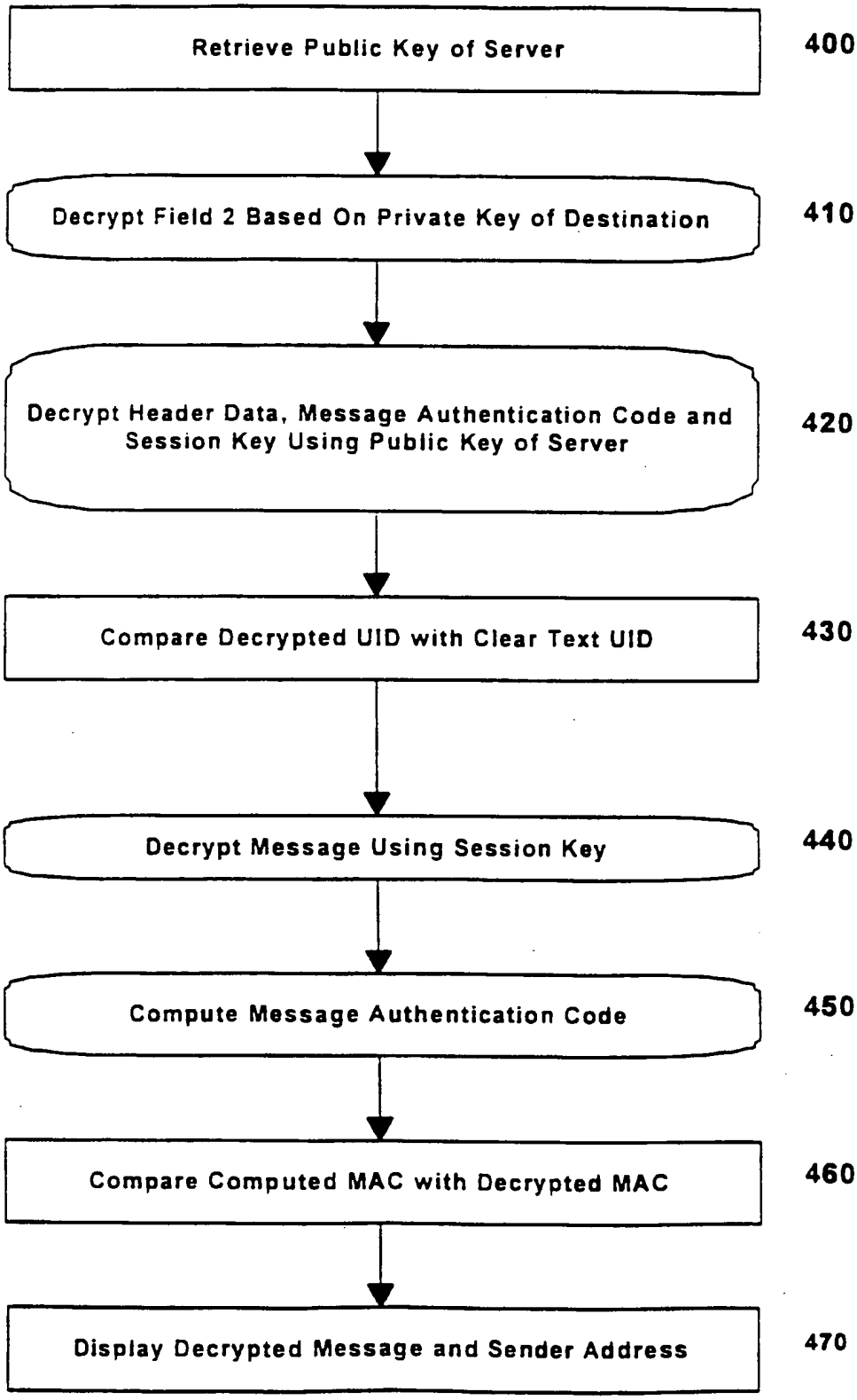


FIG. 9

FIG. 10



INTERNATIONAL SEARCH REPORT

International application No.
PCT/US98/27531

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) :H04L 9/08
US CL :380/21

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 380/21,44,45,49

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

Please See Extra Sheet.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 5,285,496 A (FRANK et al) 08 February 1994 (08.02.94), column 2, lines 28-44, column 4, lines 12-68, column 6, lines 11-49.	1-31,36-56, 58,60-68,74-78
Y	US 5,604,801 A (DOLAN et al) 18 February 1997 (18.02.97), abstract, column 3, lines 2-38, 50-60, column 4, lines 19-24, 40-55.	1-31,36-56 58,60-68, 74-78
A	US 5,602,918 A (CHEN et al) 11 February 1997 (11.02.97), abstract, column 2, lines 36-41,57-60, column 4, lines 43-63.	3-4,6,17-18,27-28,31-32,40,42,53,65,68-70,77
A	US 5,452,356 A (ALBERT et al) 19 September 1995 (19.09.95), column 1, lines 60-68, column 2, lines 1-42, column 11, lines 15-55.	1-78

[X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search 17 FEBRUARY 1999	Date of mailing of the international search report 06 MAY 1999
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer GAIL HAYES Telephone No. (703) 305-9711

Form PCT/ISA/210 (second sheet)(July 1992)*

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US98/27531

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5,495,533 A (LINEHAN et al) 27 February 1996 (27.02.96), column 9, lines 42-58, column 10, lines 22-32.	7,9,35,59,69-70,72-73

Form PCT/ISA/210 (continuation of second sheet)(July 1992)*

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US98/27531

B. FIELDS SEARCHED
Electronic data bases consulted (Name of data base and where practicable terms used):

APS
search terms: cypher, cipher, encode,encrypt, decrypt, key, keys, pager, wireless, proxy server, authenticate, authentication, transmission, transmitting, key management, public key, two-way communication, re-encrypt, messages, data, information

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : H04L 9/30 | A1

(11) International Publication Number: WO 99/35782

(43) International Publication Date: 15 July 1999 (15.07.99)

(21) International Application Number: PCT/US98/27896

(22) International Filing Date: 31 December 1998 (31.12.98)

(30) Priority Data:

60/070,344	2 January 1998 (02.01.98)	US
60/089,529	15 June 1998 (15.06.98)	US

(71) Applicant: CRYPTOGRAPHY RESEARCH, INC. [US/US]; Suite 1088, 870 Market Street, San Francisco, CA 94102 (US).

(72) Inventors: KOCHER, Paul, C.; 143 Fillmore Street, San Francisco, CA 94117 (US). JAFFE, Joshua, M.; 21B Bird Street, San Francisco, CA 94110 (US).

(74) Agents: LAURIE, Ronald, S. et al.; Skadden, Arps, Slate, Meagher & Flom LLP, 525 University Avenue, Palo Alto, CA 94301 (US).

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, HR, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published

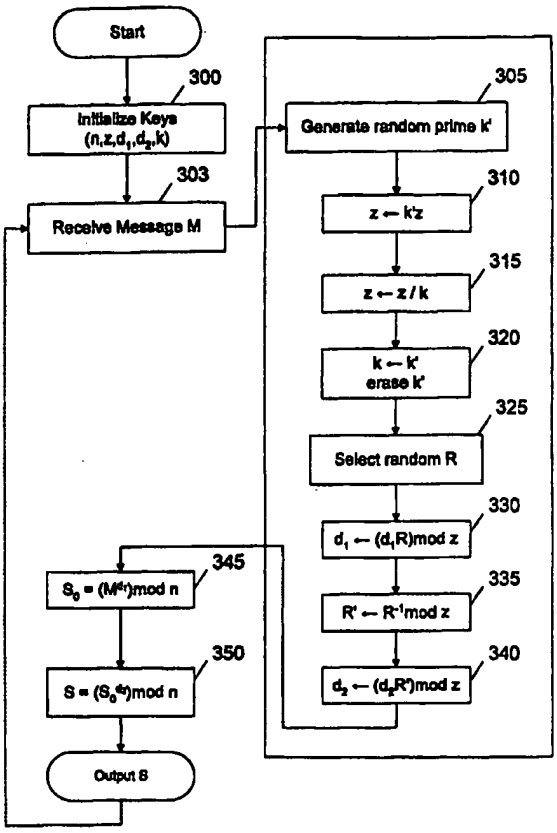
With international search report.

Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: LEAK-RESISTANT CRYPTOGRAPHIC METHOD AND APPARATUS

(57) Abstract

The present invention provides a method and apparatus for securing cryptographic devices against attacks involving external monitoring and analysis. A "self-healing" property is introduced, enabling security to be continually re-established following partial compromises. In addition to producing useful cryptographic results, a typical leak-resistant cryptographic operation modifies or updates (330) secret key material in a manner designed to render useless any information about the secrets that may have previously leaked from the system. Exemplary leak-proof and leak-resistant implementations of the invention are shown for symmetric authentication (350), certified Diffie-Hellman (when either one or both users have certificates), RSA, ElGamal public key decryption (303).



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

LEAK-RESISTANT CRYPTOGRAPHIC METHOD AND APPARATUS

This application claims the benefit of US Provisional Application No. 60/070,344 filed January 2, 1998, and US Provisional Application No. 60/089,529 filed June 15, 1998.

TECHNICAL FIELD

The method and apparatus of the present invention relate generally to cryptographic systems and, more specifically, to securing cryptographic tokens that must maintain the security of secret information in hostile environments.

BACKGROUND OF THE INVENTION

Most cryptosystems require secure key management. In public-key based security systems, private keys must be protected so that attackers cannot use the keys to forge digital signatures, modify data, or decrypt sensitive information. Systems employing symmetric cryptography similarly require that keys be kept secret. Well-designed cryptographic algorithms and protocols should prevent attackers who eavesdrop on communications from breaking systems. However, cryptographic algorithms and protocols traditionally require that tamper-resistant hardware or other implementation-specific measures prevent attackers from accessing or finding the keys.

If the cryptosystem designer can safely assume that the key management system is completely tamper-proof and will not reveal any information relating to the keys except via the messages and operations defined in the protocol, then previously known cryptographic techniques are often sufficient for good security. It is currently extremely difficult, however, to make hardware key management systems that provide good security, particularly in low-cost unshielded cryptographic devices for use in applications where attackers will have physical control over the device. For example, cryptographic tokens (such as smartcards used in electronic cash and copy protection schemes) must protect their keys even in potentially hostile environments. (A token is a device that contains or manipulates cryptographic keys that need to be protected from attackers. Forms in which tokens may be manufactured include, without limitation, smartcards, specialized encryption and key management devices, secure telephones, secure picture phones, secure web servers, consumer electronics devices using cryptography, secure microprocessors, and other tamper-resistant cryptographic systems.)

A variety of physical techniques for protecting cryptographic devices are known, including enclosing key management systems in physically durable enclosures, coating integrated circuits with special coatings that destroy the chip when removed, and wrapping devices with fine wires that detect tampering. However, these approaches are expensive, difficult to use in single-chip solutions (such as smartcards), and difficult to evaluate since there is no mathematical basis for their security. Physical tamper resistance techniques are also ineffective against some attacks. For example, recent work by Cryptography Research has shown that attackers can non-invasively extract secret keys using careful measurement and analysis of many devices' power consumption. Analysis of timing measurements or electromagnetic radiation can also be used to find secret keys.

Some techniques for hindering external monitoring of cryptographic secrets are known, such as using power supplies with large capacitors to mask fluctuations in power consumption, enclosing devices in well-shielded cases to prevent electromagnetic radiation, message blinding to prevent timing attacks, and buffering of inputs/outputs to prevent signals from leaking out on I/O lines. Shielding, introduction of noise, and other such countermeasures are often, however, of limited value, since skilled attackers can still find keys by amplifying signals and filtering out noise by averaging data collected from many operations. Further, in smartcards and other tamper-resistant chips, these countermeasures are often inapplicable or insufficient due to reliance on external power sources, impracticality of shielding, and other physical constraints. The use of blinding and constant-time mathematical algorithms to prevent timing attacks is also known, but does not prevent more complex attacks such as power consumption analysis (particularly if the system designer cannot perfectly predict what information will be available to an attacker, as is often the case before a device has been physically manufactured and characterized).

The present invention makes use of previously-known cryptographic primitives and operations. For example: U.S. patent 5,136,646 to Haber et al. and the pseudorandom number generator used in the RSAREF cryptographic library use repeated application of hash functions; anonymous digital cash schemes use blinding techniques; zero knowledge protocols use hash functions to mask information; and key splitting and threshold schemes store secrets in multiple parts.

SUMMARY OF THE INVENTION

The present invention introduces leak-proof and leak-resistant cryptography, mathematical approaches to tamper resistance that support many existing cryptographic primitives, are inexpensive, can be implemented on existing hardware (whether by itself or via software capable of running on such hardware), and can solve problems involving secrets leaking out of cryptographic devices. Rather than assuming that physical devices will provide perfect security, leak-proof and leak-resistant cryptographic systems may be designed to remain secure even if attackers are able to gather some information about the system and its secrets. This invention describes leak-proof and leak-resistant systems that implement symmetric authentication, Diffie-Hellman exponential key agreement, ElGamal public key encryption, ElGamal signatures, the Digital Signature Standard, RSA, and other algorithms.

One of the characteristic attributes of a typical leak-proof or leak-resistant cryptosystem is that it is "self-healing" such that the value of information leaked to an attacker decreases or vanishes with time. Leak-proof cryptosystems are able to withstand leaks of up to L_{MAX} bits of information per transaction, where L_{MAX} is a security factor chosen by the system designer to exceed to the maximum anticipated leak rate. The more general class of leak-resistant cryptosystems includes leak-proof cryptosystems, and others that can withstand leaks but are not necessarily defined to withstand any defined maximum information leakage rate. Therefore, any leak-proof system shall also be understood to be leak-resistant. The leak-resistant systems of the present invention can survive a variety of monitoring and eavesdropping attacks that would break traditional (non-leak-resistant) cryptosystems.

A typical leak-resistant cryptosystem of the present invention consists of three general parts. The initialization or key generation step produces secure keying material appropriate for the scheme. The update process cryptographically modifies the secret key material in a manner designed to render useless any information about the secrets that may have previously leaked from the system, thus providing security advantages over systems of the background art. The final process performs cryptographic operations, such as producing digital signatures or decrypting messages.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 shows an exemplary leak-resistant symmetric authentication method.

Figure 2 shows an exemplary leak-resistant Diffie-Hellman exponential key exchange operation.

Figure 3 shows an exemplary leak-resistant RSA private key operation.

Figure 4 shows an exemplary leak-resistant ElGamal signing operation.

DETAILED DESCRIPTION OF THE INVENTION

The sections following will describe an introduction to leak-proof/leak-resistant cryptography, followed by various embodiments of the general techniques of the invention as applied to improve the security of common cryptographic protocols.

I. Introduction and Terminology

The leakage rate L is defined as the number of bits of useful information about a cryptosystem's secrets that are revealed per operation, where an operation is a cryptographic transaction. Although an attacker may be able to collect more than L bits worth of measurement data, by definition this data yields no more than L bits of useful information about the system's secrets.

The implementer of a leak-proof system chooses a design parameter L_{MAX}, the maximum amount of leakage per operation the system may allow if it is to remain uncompromised. L_{MAX} should be chosen conservatively, and normally should significantly exceed the amount of useful information known to be leaked to attackers about the system's secrets during each transaction. Designers do not necessarily need to know accurately or completely the quantity and type of information that may leak from their systems; the choice of L_{MAX} may be made using estimates and models for the system's behavior. General factors affecting the choice of L_{MAX} include the types of monitoring potentially available to attackers, the amount of error in attackers' measurements, and engineering constraints that limit L_{MAX}. (Larger values of L_{MAX} increase memory and performance requirements of the device, and in some cases may increase L.) To estimate the amount of useful information an attacker could collect by monitoring a device's power consumption, for example, a designer might consider the amount of noise in the device's power usage, the power line capacitance, the useful time resolution for power consumption measurements, as well as the strength of the signals being monitored. Similarly, the designer knows that timing measurements can rarely yield more than a few bits of information per operation, since timing information is normally quantized to an integral number of clock cycles. In choosing L_{MAX}, the designer should assume that

attackers will be able to combine information gleaned from multiple types of attacks. If the leakage rate is too large (as in the extreme case where L equals the key size because the entire key can be extracted during a single transaction), additional design features should be added to reduce L and reduce the value needed for L_{MAX}. Such additional measures can include known methods, such as filtering the device's power inputs, adding shielding, introducing noise into the timing or power consumption, implementing constant-time and constant execution path algorithms, and changing the device layout. Again, note that the designer of a leak-resistant system does not actually need to know what information is being revealed or how it is leaked; all he or she need do is choose an upper bound for the rate at which attackers might learn information about the keys. In contrast, the designer of a traditional system faces the much harder task of ensuring that no information about the secrets will leak out.

There are many ways information about secrets can leak from cryptosystems. For example, an attacker can use a high-speed analog-to-digital converter to record a smartcard's power consumption during a cryptographic operation. The amount of useful information that can be gained from such a measurement varies, but it would be fairly typical to gain enough information to guess each of 128 key bits correctly with a probability of 0.7. This information can reduce the amount of effort required for a brute force attack. For example, a brute force attack with one message against a key containing k bits where each bit's value is known with probability p can be completed in

$$E(k,p) = \sum_{i=0}^{k} \left[\binom{k}{i} (1-p)^i p^{k-i} \left[\left(\sum_{j=0}^{i} \binom{k}{j} \right) - \frac{1}{2} \binom{k}{i} \right] + \frac{1}{2} \right]$$

operations. The reduction in the effort for a brute force attack is equivalent to shortening the key by $L = \log_2(E(k,½) / E(k,p)) = \log_2(k - E(k,p) - 1)$ bits. (For example, in the case of $k =$ 128 and $p = 0.7$, L is estimated to be about 11 bits for the first measurement. With a multiple message attack, the attacker's effort can fall to as low as $E(k,p) = \frac{1}{p^k}$.) Attackers can gain additional information about the keys by measuring additional operations; unless leak-resistance is used, finding the key becomes easy after just a few dozen operations.

When choosing L_{MAX}, a system designer should consider the signal-to-noise ratio of an attacker's measurements. For example, if the signal and noise are of roughly equivalent magnitude, the designer knows that an attacker's measurements should be incorrect about 25 percent of the time (e.g., $p = 0.75$ if only one observation per key bit is possible). Many

measurement techniques, such as those involving timing, may have signal-to-noise ratios of 1:100 or worse. With such systems, L is generally quite small, but attackers who can make a large number of measurements can use averaging or other statistical techniques to recover the entire key. In extreme cases, attackers may be able to obtain all key bits with virtually perfect accuracy from a single transaction (i.e., $L = k$), necessitating the addition of shielding, noise in the power consumption (or elsewhere), and other measures to reduce p and L. Of course, L_{MAX} should be chosen conservatively; in the example above where less than 4 useful bits are obtained per operation for the given attack, the designer might select $L_{MAX} = 64$ for a leak-proof design.

Leak-proof (and, more generally, leak-resistant) cryptosystems provide system designers with important advantages. When designing a traditional (i.e., non-leak-resistant and non-leak-proof) cryptosystem, a careful cryptosystem designer should study all possible information available to attackers if he or she is to ensure that no analytical techniques could be used to compromise the keys. In practice, many insecure systems are developed and deployed because such analysis is incomplete, too difficult even to attempt, or because the cryptographers working on the system do not understand or cannot completely control the physical characteristics of the device they are designing. Unexpected manufacturing defects or process changes, alterations made to the product by attackers, or modifications made to the product in the field can also introduce problems. Even a system designed and analyzed with great care can be broken if new or improved data collection and analysis techniques are found later. In contrast, with leak-proof cryptography, the system designer only needs to define an upper bound on the maximum rate at which attackers can extract information about the keys. A detailed understanding of the information available to attackers is not required, since leak-proof (and leak-resistant) cryptosystem designs allow for secret information in the device to leak out in (virtually) any way, yet remain secure despite this because leaked information is only of momentary value.

In a typical leak-proof design, with each new cryptographic operation i, the attacker is assumed to be able to choose any function F_i and determine the L_{MAX}-bit result of computing F_i on the device's secrets, inputs, intermediates, and outputs over the course of the operation. The attacker is even allowed to choose a new function F_i with each new operation. The system may be considered leak-proof with a security factor n and leak rate L_{MAX} if, after observing a large number of operations, an attacker cannot forge signatures, decrypt data, or

perform other sensitive operations without performing an exhaustive search to find an n-bit key or performing a comparable $O(2^n)$ operation. In addition to choosing L_{MAX}, designers also choose n, and should select a value large enough to make exhaustive search infeasible. In the sections that follow, various embodiments of the invention, as applied to improve the security of common cryptographic operations and protocols, will be described in more detail.

II. Symmetric Cryptographic Protocols

A. Symmetric Authentication

An exemplary cryptographic protocol that can be secured using the techniques of the present invention is symmetric authentication.

1. Conventional Symmetric Authentication

Assume a user wishes to authenticate herself to a server using an n-bit secret key, K, known to both the server and the user's cryptographic token, but not known to attackers. The cryptographic token should be able to resist tampering to prevent, for example, attackers from being able to extract secrets from a stolen token. If the user's token has perfect tamper resistance (i.e., L=0), authentication protocols of the background art can be used. Typically the server sends a unique, unpredictable challenge value R to the user's token, which computes the value $A = H(R \| K)$, where "||" denotes concatenation and H is a one-way cryptographic hash function such as SHA. The user sends A to the server, which independently computes A (using its copy of K) and compares its result with the received value. The user authentication succeeds only if the comparison operation indicates a match.

If the function H is secure and if K is sufficiently large to prevent brute force attacks, attackers should not be able to obtain any useful information from the (R, A) values of old authentication sessions. To ensure that attackers cannot impersonate users by replaying old values of A, the server generates values of R that are effectively (with sufficiently high probability) unique. In most cases, the server should also make R unpredictable to ensure that an attacker with temporary possession of a token cannot compute future values of A. For example, R might be a 128-bit number produced using a secure random number generator (or pseudorandom number generator) in the server. The properties of cryptographic hash functions such as H have been the subject of considerable discussion in the literature, and need not be described in detail here. Hash functions typically provide functionality modeled after a random oracle, deterministically producing a particular output from any input. Ideally, such functions should be collision-resistant, non-invertable, should not leak partial

information about the input from the output, and should not leak information about the output unless the entire input is known. Hash functions can have any output size. For example, MD5 produces 128-bit outputs and SHA produces 160-bit outputs. Hash functions may be constructed from other cryptographic primitives or other hash functions.

While the cryptographic security of the protocol using technology of the background art may be good, it is not leak-proof; even a one-bit leak function (with L=1) can reveal the key. For example, if the leak function F equals bit (R mod n) of K, an attacker can break the system quickly since a new key bit is revealed with every transaction where (R mod n) has a new value. Therefore, there is a need for a leak-proof/leak-resistant symmetric authentication protocol.

2. Leak-Resistant Symmetric Authentication

The following is one embodiment of a leak-resistant (and, in fact, also leak-proof) symmetric authentication protocol, described in the context of a maximum leakage rate of L_{MAX} bits per transaction from the token and a security factor n, meaning that attacks of complexity $O(2^n)$, such as brute-force attacks against an n-bit key, are acceptable, but there should not be significantly easier attacks. The user's token maintains a counter t, which is initialized to zero, and an $(n+2L_{MAX})$-bit shared secret K_t, which is initialized with a secret K_0. Note that against adversaries performing precomputation attacks based on Hellman's time/memory trade-off, larger values of n may be in order. Note also that some useful protocol security features, such as user and/or server identifiers in the hash operation inputs, have been omitted for simplicity in the protocol description. It is also assumed that no leaking will occur from the server. For simplicity in the protocol description, some possible security features (such as user and/or server identifiers in the hash operation inputs) have been omitted, and it is assumed that the server is in a physically secure environment. However, those skilled in the art will appreciate that the invention is not limited to such assumptions, which have been made as a matter of convenience rather than necessity.

As in the traditional protocol, the server begins the authentication process by generating a unique and unpredictable value R at step 105. For example, R might be a 128-bit output from a secure random number generator. At step 110, the server sends R to the user's token. At step 112, the token receives R. At step 115, the token increments its counter t by computing $t \leftarrow t + 1$. At step 120, the token updates K_t by computing $K_t \leftarrow H_K(t \| K_t)$, where H_K is a cryptographic hash function that produces an $(n+2L_{MAX})$ bit output from the old value

of K_t and the (newly incremented) value of t. Note that in the replacement operations (denoted "←"), the token deletes the old values of t and K_t, replacing them with the new values. By deleting the old K_t, the token ensures that future leak functions cannot reveal information about the old (deleted) value. At step 122, the token uses the new values of t and K_t to compute an authenticator $A = H_A(K_t \| t \| R)$. At step 125, the token sends both t and the authenticator A to the server, which receives them at step 130. At step 135, the server verifies that t is acceptable (e.g., not too large but larger than the value received in the last successful authentication). If t is invalid, the server proceeds to step 175. Otherwise, at step 140, the server initializes its loop counter i to zero and its key register K_t' to K_0. At step 145, the server compares i with the received value of t, proceeding to step 160 if they are equal. Otherwise, at step 150, the server increments i by computing $i \leftarrow i + 1$. At step 155, the server computes $K_t' \leftarrow H_K(i \| K_t')$, then proceeds back to step 145. At step 160, the server computes $A' = H_A(K_t' \| t \| R)$. Finally, at step 165, the server compares A and A', where the authentication succeeds at step 170 if they match, or fails at 175 if they do not match.

This design assumes that at the beginning of any transaction the attacker may have L_{MAX} bits of useful information about the state of the token (e.g., K_t) that were obtained using the leak function F in a previous operation. During the transaction, the attacker can gain an additional L_{MAX} bits of useful information from the token. If, at any time, any $2L_{MAX}$ (or fewer) bits of useful information about the secret are known to the attacker, there are still $(n+2L_{MAX})-2L_{MAX} = n$ or more unknown bits. These n bits of unknown information ensure that attacks will require $O(2^n)$ effort, corresponding to the desired security factor. However, the attacker should have no more than L_{MAX} bits of useful information about K_t at the end of the transaction. The property that attackers lose useful information during normal operation of the system is a characteristic of the leak-proof or leak-resistant cryptosystem. In general, this information loss is achieved when the cryptosystem performs operations that convert attackers' useful partial information about the secret into useless information. (Information is considered useless if it gives an attacker nothing better than the ability to test candidate values in an $O(2^n)$ exhaustive search or other "hard" operation. For example, if exhaustive search of X is hard and H is a good hash function, $H(X)$ is useless information to an attacker trying to find X.)

Thus, the attacker is assumed to begin with L_{MAX} bits of useful information about K_t before the token's $K_t \leftarrow H_K(t \| K_t)$ computation. (Initial information about anything other

than K_t is of no value to an attacker because K_t is the only secret value in the token. The function H_K and the value of t are not assumed to be secret.) The attacker's information can be any function of K_t produced from the previous operation's leaks.

3. Security Characteristics of Leak-Proof Systems

The following section provides a technical discussion of the security characteristics of the exemplary leak-proof system described above. The following analysis is provided as an example of how the design can be analyzed, and how a system may be designed using general assumptions about attackers' capabilities. The discussion and assumptions do not necessarily apply to other embodiments of the invention and should not be construed as limiting the scope or applicability of the invention in any way.

During the course of a transaction, the leak function F might reveal up to L_{MAX} information about the system and its secrets. The design assumes that any information contained in the system may be leaked by F, provided that F does not reveal useful new information about values of K_t that were deleted before the operation started, and F does not reveal useful information about values of K_t that will be computed in future operations. These constraints are completely reasonable, since real-world leaks would not reveal information about deleted or not-yet-existent data. (The only way information about future K_t values could be leaked would be the bizarre case where the leak function itself included, or was somehow derived from, the function H_K.) In practice, these constraints on F are academic and of little concern, but they are relevant when constructing proofs to demonstrate the security of a leak-proof system.

If the leak occurs at the beginning of the H_K computation, it could give the attacker up to $2L_{MAX}$ bits of useful information about the input value of K_t. Because K_t contains $(2L_{MAX}+n)$ bits of secret information and the attacker may have up to $2L_{MAX}$ bits of useful information about the initial value of K_t, there remain at least $(2L_{MAX}+n)-2L_{MAX} = n$ bits of information in K_t that are secret. The hash function H_K effectively mixes up these n bits to produce a secure new K_t during each transaction such that the attacker's information about the old K_t is no longer useful.

If the leak occurs at the end of the H_K computation, it could give an attacker up to L_{MAX} bits of information about the final value of H_K, yielding L_{MAX} bits of information about

the input to the subsequent transaction. This is not a problem, since the design assumes that attackers have up to L_{MAX} bits of information about K_t at the beginning of each transaction.

A third possibility is that the attacker's L_{MAX} bits of information might describe intermediates computed during the operation H_K. However, even if the attacker could obtain L_{MAX} new bits of information about the input to H_K and also L_{MAX} bits of information about the output from H_K, the system would be secure, since the attacker would never have more than $2L_{MAX}$ bits of information about the input K_t or more than L_{MAX} bits of information about the output K_t. Provided that L_{MAX} bits of information from within H_K cannot reveal more than L_{MAX} bits of information about the input, or more than L_{MAX} bits of information about the output, the system will be secure. This will be true unless H_K somehow compresses the input to form a short intermediate which is expanded to form the output. While hash functions whose internal states are smaller than their outputs should not be used, most cryptographic hash functions are fine.

A fourth possibility is that part or all of the leak could occur during the $A = H_A(K_t \| t \| R)$ calculation. The attacker's total "budget" for observations is L_{MAX} bits. If L_1 bits of leak occur during the H_K computation, an additional L_2 bits of information can leak during the $A = H_A(K_t \| t \| R)$ operation, where $L_2 \leq L_{MAX} - L_1$. If the second leak provides information about K_t, this is no different from leaking information about the result of the H_K computation; the attacker will still conclude the transaction with no more than L_{MAX} bits of information about K_t because $L_1 + L_2 \leq L_{MAX}$. However, the second leak could reveal information about A. To keep A secure against leaks (to prevent, for example, an attacker from using a leak to capture A and using A before the legitimate user can), the size of A should include an extra L_{MAX} bits (to provide security even if $L_2 = L_{MAX}$). Like H_K, H_A should not leak information about deleted or future values of K_t that are not used in or produced by the given operation. As with the similar assumptions on leaks from H_K, this limitation is primarily academic and of little practical concern, since real-world leak functions do not reveal information about deleted or not-yet-computed data. However, designers might be cautious when using unusual designs for H_A that are based on or derived from H_K, particularly if the operation $H_A(K_t \| t \| R)$ could reveal useful information about the result of computing $H_K(t \| K_t)$.

B. Other Leak-Resistant Symmetric Schemes

The same basic technique of updating a key (K) with each transaction, such that leakage about a key during one transaction does not reveal useful information about a key in a

subsequent (or past) transaction, can be easily extended to other applications besides authentication.

1. Symmetric Data Verification

For example and without limitation, leak-resistant symmetric data verification is often useful where a device needs to support symmetrically-signed code, data, content, or parameter updates (all of which will, as a matter of convenience, be denoted as "data" herein). In existing systems, a hash or MAC of the data is typically computed using a secret key and the data is rejected if computed hash or MAC does not match a value received with the data. For example, a MAC may be computed as HMAC(K, data), where HMAC is defined in "RFC 2104, HMAC: Keyed-Hashing for Message Authentication" by H. Krawczyk, M. Bellare, and R. Canetti, 1997. Traditional (non-leak-resistant) designs are often vulnerable to attacks including power consumption analysis of MAC functions and timing analysis of comparison operations.

In an exemplary leak-resistant verification protocol, a verifying device (the "verifier") maintains a counter t and a key K_t, which are initialized (for example at the factory) with $t \leftarrow 0$ and $K_t \leftarrow K_0$. Before the transaction, the verifier provides t to the device providing the signed data (the "signer"), which also knows K_0. The signer uses t to compute K_{t+1}' (the prime indicating a quantity derived by the signer, rather than at the verifier) from K_0 (or K_t' or any other available value of K_i'). using the relation $K_i' = H_K(i \| K_{i-1}')$, computes signature $S' = \text{HMAC}(K_{t+1}', \text{data})$, and sends S' plus any other needed information (such as data or t) to the verifier. The verifier confirms that the received value of t (if any) matches its value of t, and rejects the signature if it does not. If t matches, the verifier increments t and updates K_t in its nonvolatile memory by computing $t \leftarrow t + 1$ and $K_t \leftarrow H_K(t \| K_t)$. In an alternative embodiment, if the received value of t is larger than the internal value but the difference is not unreasonably large, it may be more appropriate to accept the signature and perform multiple updates to K_t (to catch up with the signer) instead of rejecting the signature outright. Finally, the verifier computes $S = \text{HMAC}(K_t, \text{data})$ and verifies that $S = S'$, rejecting the signature if S does not equal the value of S' received with the data.

2. Symmetric Encryption

Besides authentication and verification, leak-resistant symmetric cryptography can also be tailored to a wide variety of applications and environments. For example, if data

encryption is desired instead of authentication, the same techniques as were disclosed above may be used to generate a key K_t used for encryption rather than verification.

3. Variations in Computational Implementation

In the foregoing, various applications were disclosed for the basic technique of updating a key K_t in accordance with a counter and deleting old key values to ensure that future leakage cannot reveal information about the now-deleted key. Those skilled in the art will realize, however, that the exemplary techniques described above may be modified in various ways without departing from the spirit and scope of the invention. For example, if communications between the device and the server are unreliable (for example if the server uses voice recognition or manual input to receive t and A), then small errors in the signature may be ignored. (One skilled in the art will appreciate that many functions may be used to determine whether a signature corresponds – sufficiently closely -- to its expected value.) In another variation of the basic technique, the order of operations and of data values may be adjusted, or additional steps and parameters may be added, without significantly changing the invention. In another variation, to save on communication bandwidth or memory, the high order bits or digits of t may not need to be communicated or remembered. In another variation, as a performance optimization, devices need not recompute K_t from K_0 with each new transaction. For example, when a transaction succeeds, the server can discard K_0 and maintain the validated version of K_t. In another variation, if bi-directional authentication is required, the protocol can include a step whereby the server can authenticates itself to the user (or user's token) after the user's authentication is complete. In another variation, if the server needs to be secured against leaks as well (as in the case where the role of "server" is played by an ordinary user), it can maintain its own counter t. In each transaction, the parties agree to use the larger of their two t values, where the device with the smaller t value performs extra updates to K_t to synchronize t. In an alternate embodiment for devices that contain a clock and a reliable power source (e.g., battery), the update operation may be performed periodically, for example by computing $K_t \leftarrow H_K(t \parallel K_t)$ once per second. The token uses the current K_t to compute $A = H_A(K_t \parallel t \parallel R)$ or, if the token does not have any means for receiving R, it can output $A = H_A(K_t)$. The server can use its clock and local copy of the secret to maintain its own version of K_t, which it can use to determine whether received values of A are recent and correct. All of the foregoing show that the method and apparatus of the present invention can be implemented using numerous variations and modifications to

the exemplary embodiments described herein, as would be understood by one skilled in the art.

III. Asymmetric Cryptographic Protocols

The foregoing illustrates various embodiments of the invention that may be used with symmetric cryptographic protocols. As will be seen below, still other techniques of the present invention may be used in connection with asymmetric cryptographic operations and protocols. While symmetric cryptosystems are sufficient for some applications, asymmetric cryptography is required for many applications. There are several ways leak resistance can be incorporated into public key cryptosystems, but it is often preferable to have as little impact as possible on the overall system architecture. Most of the exemplary designs have thus been chosen to incorporate leak resistance into widely used cryptosystems in a way that only alters the key management device, and does not affect the certification process, certificate format, public key format, or processes for using the public key.

A. Certified Diffie-Hellman

Diffie-Hellman exponential key exchange is a widely used asymmetric protocol whereby two parties who do not share a secret key can negotiate a shared secret key. Implementations of Diffie-Hellman can leak information about the secret exponents, enabling attackers to determine the secret keys produced by those implementations. Consequently, a leak-resistant implementation of Diffie-Hellman would be useful. To understand such a leak-resistant implementation, it will be useful to first review a conventional Diffie-Hellman implementation.

1. Conventional Certified Diffie-Hellman

Typical protocols in the background art for performing certified Diffie-Hellman exponential key agreement involve two communicating users (or devices) and a certifying authority (CA). The CA uses an asymmetric signature algorithm (such as DSA) to sign certificates that specify a user's public Diffie-Hellman parameters (the prime p and generator g), public key (p^x mod g, where x is the user's secret exponent), and auxiliary information (such as the user's identity, a description of privileges granted to the certificate holder, a serial number, expiration date, etc.). Certificates may be verified by anyone with the CA's public signature verification key. To obtain a certificate, user U typically generates a secret exponent (x_U), computes his or her own public key $y_u = g^{x_u} \bmod p$, presents y_U along with any required auxiliary identifying or authenticating information (e.g., a passport) to the CA,

who issues the user a certificate C_U. Depending on the system, p and g may be unique for each user, or they may be system-wide constants (as will be assumed in the following description of Diffie-Hellman using the background art).

Using techniques of the background art, Alice and Bob can use their certificates to establish a secure communication channel. They first exchange certificates (C_{Alice} and C_{Bob}). Each verifies that the other's certificate is acceptable (e.g., properly formatted, properly signed by a trusted CA, not expired, not revoked, etc.). Because this protocol will assume that p and g are constants, they also check that the certificate's p and g match the expected values. Alice extracts Bob's public key (y_{Bob}) from C_{Bob} and uses her secret exponent (x_{Alice}) to compute $z_{\text{Alice}} = (y_{\text{Bob}})^{x_{\text{Alice}}} \bmod p$. Bob uses his secret exponent and Alice's public key to compute $z_{\text{Bob}} = (y_{\text{Alice}})^{x_{\text{Bob}}} \bmod p$. If everything works correctly, $z_{\text{Alice}} = z_{\text{Bob}}$, since:

$$\begin{aligned} z_{\text{Alice}} &= (y_{\text{Bob}})^{x_{\text{Alice}}} \bmod p \\ &= \left(g^{x_{\text{Bob}}}\right)^{x_{\text{Alice}}} \bmod p \\ &= \left(g^{x_{\text{Alice}}}\right)^{x_{\text{Bob}}} \bmod p \\ &= (y_{\text{Alice}})^{x_{\text{Bob}}} \bmod p \\ &= z_{\text{Bob}}. \end{aligned}$$

Thus, Alice and Bob have a shared key $z = z_{\text{Alice}} = z_{\text{Bob}}$. An attacker who pretends to be Alice but does not know her secret exponent (x_{Alice}) will not be able to compute $z_{\text{Alice}} = (y_{\text{Bob}})^{x_{\text{Alice}}} \bmod p$ correctly. Alice and Bob can positively identify themselves by showing that they correctly found z. For example, each can compute and send the other the hash of z concatenated with their own certificate. Once Alice and Bob have verified each other, they can use a symmetric key derived from z to secure their communications. (For an example of a protocol in the background art that uses authenticated Diffie-Hellman, see "The SSL Protocol Version 3.0" by A. Freier, P. Karlton, and P. Kocher, March 1996.)

2. Leak-Resistant Certified Diffie-Hellman

A satisfactory leak-resistant public key cryptographic scheme should overcome the problem that, while certification requires the public key be constant, information about the corresponding private key should not leak out of the token that contains it. In the symmetric protocol described above, the design assumes that the leak function reveals no useful information about old deleted values of K_t or about future values of K_t that have not yet been

computed. Existing public key schemes, however, require that implementations repeatedly perform a consistent, usually deterministic, operation using the private key. For example, in the case of Diffie-Hellman, a leak-resistant token that is compatible with existing protocols and implementations should be able to perform the secret key operation $y^x \bmod p$, while ensuring that the exponent x remains secret. The radical reshuffling of the secret provided by the hash function H_K in the symmetric approach cannot be used because the device should be able to perform the same operation consistently.

The operations used by the token to perform the private key operation are modified to add leak resistance using the following variables:

Register	Comment
x_1	First part of the secret key (in nonvolatile updateable memory)
x_2	Second part of the secret key (in nonvolatile updateable memory)
g	The generator (not secret).
p	The public prime, preferably a strong prime (not secret).

The prime p and generator g may be global parameters, or may be specific to individual users or groups of users (or tokens). In either case, the certificate recipient should be able to obtain p and g securely, usually as built-in constants or by extracting them from the certificate.

To generate a new secret key, the key generation device (often but not always the cryptographic token that will contain the key) first obtains or generates p and g, where p is the prime and g is a generator mod p. If p and g are not system-wide parameters, algorithms known in the background art for selecting large prime numbers and generators may be used. It is recommended that p be chosen with $\frac{p-1}{2}$ also prime, or at least that $\phi(p)$ not be smooth. (When $\frac{p-1}{2}$ is not prime, information about x_1 and x_2 modulo small factors of $\phi(p)$ may be leaked, which is why it is preferable that $\phi(p)$ not be smooth. Note that ϕ denotes Euler's totient function.) Once p and g have been chosen, the device generates two random exponents x_1 and x_2. The lowest-order bit of x_1 and of x_2 is not considered secret, and may be set to 1. Using p, g, x_1, and x_2, the device can then compute its public key as $g^{x_1 x_2} \bmod p$ and submit it, along with any required identifying information or parameters needed (e.g., p and g), to the CA for certification.

Figure 2 illustrates the process followed by the token to perform private key operations. At step 205, the token obtains the input message y, its own (non-secret) prime p, and its own secret key halves (x_1 and x_2). If x_1, x_2, and p are stored in encrypted and/or

authenticated form, they would be decrypted or verified at this point. At this step, the token should verify that $1 < y < p-1$. At step 210, the token uses a random number generator (or pseudorandom number generator) to select a random integer b_0, where $0 < b_0 < p$. At step 215, the token computes $b_1 = b_0^{-1} \bmod p$. The inverse computation mod p may be performed using the extended Euclidean algorithm or the formula $b_1 = b_0^{\phi(p)-1} \bmod p$. At step 220, the token computes $b_2 = b_1^{x_1} \bmod p$. At this point, b_1 is no longer needed; its storage space may be used to store b_2. Efficient algorithms for computing modular exponentiation, widely known in the art, may be used to complete step 220. Alternatively, when a fast modular exponentiator is available, the computation b_2 may be performed using the relationship $b_2 = b_0^{\phi(p)-x_1} \bmod p$. At step 225, the token computes $b_3 = b_2^{x_2} \bmod p$. At this point, b_2 is no longer needed; its storage space may be used to store b_3. At step 230, the token computes $z_0 = b_0 y \bmod p$. At this point, y and b_0 are no longer needed; their space may be used to store r_1 (computed at step 235) and z_0. At step 235, the token uses a random number generator to select a random integer r_1, where $0 < r_1 < \phi(p)$ and $\gcd(r_1, \phi(p)) = 1$. (If $\frac{p-1}{2}$ is known to be prime, it is sufficient to verify that r_1 is odd.) At step 240, the token updates x_1 by computing $x_1 \leftarrow x_1 r_1 \bmod \phi(p)$. The old value of x_1 is deleted and replaced with the updated value. At step 245, the token computes $r_2 = (r_1^{-1}) \bmod \phi(p)$. If $\frac{p-1}{2}$ is prime, then r_2 can be found using a modular exponentiator and the Chinese Remainder Theorem. Note that r_1 is not needed after this step, so its space may be used to store r_2. At step 250, the token updates x_2 by computing $x_2 \leftarrow x_2 r_2 \bmod \phi(p)$. The old value of x_2 should be deleted and replaced with the updated value. At step 255, the token computes $z_1 = (z_0)^{r_1} \bmod p$. Note that z_0 is not needed after this step, so its space may be used to store z_1. At step 260, the token computes $z_2 = (z_1)^{r_2} \bmod p$. Note that z_1 is not needed after this step, so its space may be used to store z_2. At step 265, the token finds the exponential key exchange result by computing $z = z_2 b_3 \bmod p$. Finally, at step 270, the token erases and frees any remaining temporary variables.

The process shown in Figure 2 correctly computes $z = y^x \bmod p$, where $x = x_1 x_2 \bmod \phi(p)$, since:

$$
\begin{aligned}
z &= z_2 b_3 \bmod p \\
&= \left(z_1^{x_2} \bmod p\right)\left(b_2^{x_2} \bmod p\right) \bmod p \\
&= \left(\left(z_0^{x_1} \bmod p\right)^{x_2}\right)\left(\left(b_1^{x_1} \bmod p\right)^{x_2}\right) \bmod p \\
&= \left(b_0 y \bmod p\right)^{x_1 x_2}\left(b_0^{-1} \bmod p\right)^{x_1 x_2} \bmod p \\
&= y^{x_1 x_2} \bmod p \\
&= y^{x} \bmod p.
\end{aligned}
$$

The invention is useful for private key owners communicating with other users (or devices) who have certificates, and also when communicating with users who do not.

If Alice has a certificate and wishes to communicate with Bob who does not have a certificate, the protocol proceeds as follows. Alice sends her certificate (C_{Alice}) to Bob, who receives it and verifies that it is acceptable. Bob extracts y_{Alice} (along with p_{Alice} and g_{Alice}, unless they are system-wide parameters) from C_{Alice}. Next, Bob generates a random exponent x_{BA}, where $0 < x_{\text{AB}} < \phi(p_{\text{Alice}})$. Bob then uses his exponent x_{AB} and Alice's parameters to calculate $y_{\text{BA}} = \left(g_{\text{Alice}}^{x_{\text{BA}}}\right) \bmod p_{\text{Alice}}$ and the session key $z = \left(y_{\text{Alice}}^{x_{\text{BA}}}\right) \bmod p_{\text{Alice}}$. Bob sends y_{BA} to Alice, who performs the operation illustrated in Figure 2 to update her internal parameters and derive z from y_{BA}. Alice then proves that she computed z correctly, for example by sending Bob $H(z \parallel C_{\text{Alice}})$. (Alice cannot authenticate Bob because he does not have a certificate. Consequently, she does not necessarily need to verify that he computed z successfully.) Finally, Alice and Bob can use z (or, more commonly, a key derived from z) to secure their communications.

If both Alice and Bob have certificates, the protocol works as follows. First, Alice and Bob exchange certificates (C_{Alice} and C_{Bob}), and each verifies that other's certificate is valid. Alice then extracts the parameters p_{Bob}, g_{Bob}, and y_{Bob} from C_{Bob}, and Bob extracts p_{Alice}, g_{Alice}, and y_{Alice} from C_{Alice}. Alice then generates a random exponent x_{AB} where $0 < x_{\text{AB}} < \phi(p_{\text{Bob}})$, computes $y_{\text{AB}} = \left(g_{\text{Bob}}\right)^{x_{\text{AB}}} \bmod p_{\text{Bob}}$, and computes $z_{\text{AB}} - \left(y_{\text{Bob}}\right)^{x_{\text{AB}}} \bmod p_{\text{Bob}}$. Bob generates a random x_{BA} where $0 < x_{\text{BA}} < \phi(p_{\text{Alice}})$, computes $y_{\text{BA}} = \left(g_{\text{Alice}}\right)^{x_{\text{BA}}} \bmod p_{\text{Alice}}$, and computes $z_{\text{BA}} - \left(y_{\text{Alice}}\right)^{x_{\text{BA}}} \bmod p_{\text{Alice}}$. Bob sends y_{BA} to Alice, and Alice sends y_{AB} to Bob. Alice and Bob each perform the operation shown in Figure 2, where each uses the prime p from their own certificate and their own secret exponent halves (x_1 and x_2). For the message y in Figure 2, Alice uses y_{BA} (received from Bob), and Bob uses y_{AB} (received from Alice). Using the process shown in Figure 2, Alice computes z. Using z and z_{AB} (computed

previously), she can find a session key K. This may be done, for example, by using a hash function H to compute $K = H(z \| z_{AB})$. The value of z Bob obtains using the process shown in Figure 2 should equal Alice's z_{AB}, and Bob's z_{BA} (computed previously) should equal Alice's z. If there were no errors or attacks, Bob should thus be able to find K, e.g., by computing $K = H(z_{BA} \| z)$. Alice and Bob now share K. Alice can prove her identity by showing that she computed K correctly, for example by sending Bob $H(K \| C_{Alice})$. Bob can prove his identity by sending Alice $H(K \| C_{Bob})$. Alice and Bob can then secure their communications by encrypting and authenticating using K or a key derived from K.

Note that this protocol, like the others, is provided as an example only; many variations and enhancements of the present invention are possible and will be evident to one skilled in the art. For example, certificates may come from a directory, more than two parties can participate in the key agreement, key escrow functionality may be added, the prime modulus p may be replaced with a composite number, etc. Note also that Alice and Bob as they are called in the protocol are not necessarily people; they would normally be computers, cryptographic devices, etc.

For leak resistance to be effective, attackers should not be able to gain new useful information about the secret variables with each additional operation unless a comparable amount of old useful information is made useless. While the symmetric design is based on the assumption that leaked information will not survive the hash operation H_K, this design uses multiplication operations mod $\phi(p)$ to update x_1 and x_2. The most common variety of leaked information, statistical information about exponent bits, is not of use to attackers in this design, as the exponent update process ($x_1 \leftarrow x_1 r_1 \bmod \phi(p)$ and $x_2 \leftarrow x_2 r_2 \bmod \phi(p)$) destroys the utility of this information. The only relevant characteristic that survives the update process is that $x_1 x_2 \bmod \phi(p)$ remains constant, so the system designer should be careful to ensure that the leak function does not reveal information allowing the attacker to find new useful information about $x_1 x_2 \bmod \phi(p)$.

There is a modest performance penalty, approximately a factor of four, for the leak-resistant design as described. One way to improve performance is to remove the blinding and unblinding operations, which are often unnecessary. (The blinding operations prevent attackers from correlating input values of y with the numbers processed by the modular exponentiation operation.) Alternatively or additionally, it is possible to update and reuse

values of b_0, b_3, r_1, and r_2 by computing $b_0 \leftarrow (b_0)^v \bmod p$, $b_3 \leftarrow (b_3)^v \bmod p$, $r_1 \leftarrow (r_1)^w \bmod \phi(p)$, and $r_2 \leftarrow (r_2)^w \bmod \phi(p)$, where v and w are fairly short random exponents. Note that the relationship $b_3 \leftarrow b_0^{-x_1 x_2} \bmod p$ remains true when b_0 and b_3 are both raised to the power v (mod p). The relationship $r_2 = (r_1^{-1}) \bmod \phi(p)$ also remains true when r_1 and r_2 are exponentiated (mod $\phi(p)$). Other parameter update operations may also be used, such as exponentiation with fixed exponents (e.g., $v = w = 3$), or multiplication with random values and their inverses, mod p and $\phi(p)$. The time per transaction with this update process is about half that of the unoptimized leak-resistant implementation, but additional storage is required and care should be taken to ensure that b_0, b_3, r_1, and r_2 will not be leaked or otherwise compromised.

It should also be noted that with this particular type of certified Diffie-Hellman, the negotiated key is the same every time any given pair of users communicate. Consequently, though the blinding operation performed using b_0 and b_3 does serve to protect the exponents, the result K can be leaked in the final step or by the system after the process is complete. If storage is available, parties could keep track of the values of y they have received (or their hashes) and reject duplicates. Alternatively, to ensure that a different result is obtained from each negotiation, Alice and Bob can generate and exchange additional exponents, w_{Alice} and w_{Bob}, for example with $0 < w < 2^{128}$ (where $2^{128} << p$). Alice sets $y = (y_{\text{BA}})^{w_{\text{Alice}} w_{\text{Bob}}} \bmod p$ instead of just $y = y_{\text{BA}}$, and Bob sets $y = (y_{\text{AB}})^{w_{\text{Bob}} w_{\text{Alice}}} \bmod p$ instead of $y = y_{\text{AB}}$ before performing the operation shown in Figure 2.

B. Leak-Resistant RSA

Another asymmetric cryptographic protocol is RSA, which is widely used for digital signatures and public key encryption. RSA private key operations rely on secret exponents. If information about these secret exponents leaks from an implementation, its security can be compromised. Consequently, a leak-resistant implementation of RSA would be useful.

To give RSA private key operations resistance to leaks, it is possible to divide the secret exponent into two halves such that information about either half is destroyed with each operation. These are two kinds of RSA private key operations. The first, private key signing, involves signing a message with one's own private key to produce a digital signature verifiable by anyone with one's corresponding public key. RSA signing operations involve computing $S = M^d \bmod n$, where M is the message, S is the signature (verifiable using $M = S^e$

mod n), d is the secret exponent and equals e^{-1} mod $\phi(n)$, and n is the modulus and equals pq, where n and e are public and p and q are secret primes, and ϕ is Euler's phi function. An RSA public key consists of e and n, while an RSA private key consists of d and n (or other representations of them). For RSA to be secure, d, $\phi(n)$, p, and q should all be secret.

The other RSA operation is decryption, which is used to recover messages encrypted using one's public key. RSA decryption is virtually identical to signing, since the decrypted message M is recovered from the ciphertext C by computing $M = C^d$ mod n, where the ciphertext C was produced by computing $C=M^e$ mod n. Although the following discussion uses variable names from the RSA signing operation, the same techniques may be applied similarly to decryption.

An exemplary leak-resistant scheme for RSA implementations may be constructed as illustrated in Figure 3. At step 300, prior to the commencement of any signing or decryption operations, the device is initialized with (or creates) the public and private keys. The device contains the public modulus n and the secret key components d_1, d_2, and z, and k, where k is a prime number of medium-size (e.g., $0 < k < 2^{128}$) chosen at random, $z = k\phi(n)$, d_1 is a random number such that $0 < d_1 < z$ and $\gcd(d_1, z) = 1$, and $d_2 = (e^{-1} \bmod \phi(n))(d_1^{-1} \bmod z) \bmod z$. In this invention, d_1 and d_2 replace the usual RSA secret exponent d. Techniques for generating the initial RSA primes (e.g., p and q) and modulus (n) are well known in the background art. At step 305, the device computes a random prime k' of medium size (e.g., $0 < k' < 2^{128}$). (Algorithms for efficiently generating prime numbers are known in the art.)

At step 303, the device (token) receives a message M to sign (or to decrypt). At step 310, the device updates z by computing $z \leftarrow k'z$. At step 315, the device updates z again by computing $z \leftarrow z / k$. (There should be no remainder from this operation, since k divides z.) At step 320, k is replaced with k' by performing $k \leftarrow k'$. Because k' will not be used in subsequent operations, its storage space may be used to hold R (produced at step 325). At step 325, the device selects a random R where $0 < R < z$ and $\gcd(R, z) = 1$. At step 330, the device updates d_1 by computing $d_1 \leftarrow d_1 R \bmod z$. At step 335, the device finds the inverse of R by computing $R' \leftarrow R^{-1} \bmod z$ using, for example, the extended Euclidean algorithm. Note that R is no longer needed after this step, so its storage space may be erased and used to hold R'. At step 340, the device updates d_2 by computing $d_2 \leftarrow d_2 R' \bmod z$. At step 345, the device computes $S_0 = M^{d_1} \bmod n$, where M is the input message to be signed (or the message

to be decrypted). Note that M is no longer needed after this step, so its storage space may be used for S_0. At step 350, the device computes $S = S_0^{d_2} \bmod n$, yielding the final signature (or plaintext if decrypting a message). Leak-resistant RSA has similar security characteristics as normal RSA; standard message padding, post-processing, and key sizes may be used. Public key operations are also performed normally (e.g., $M = S^e \bmod n$).

A simpler RSA leak resistance scheme may be implemented by splitting the exponent d into two halves d_1 and d_2 such that $d_1 + d_2 = d$. This can be achieved during key generation by choosing d_1 to be a random integer where $0 \le d_1 \le d$, and choosing $d_2 \leftarrow d - d_1$. To perform private key operations, the device needs d_1 and d_2, but it does not need to contain d. Prior to each private key operation, the cryptographic device identifies which of d_1 and d_2 is larger. If $d_1 > d_2$, then the device computes a random integer r where $0 \le r \le d_1$, adds r to d_2 (i.e., $d_2 \leftarrow d_2 + r$), and subtracts r from d_1 (i.e., $d_1 \leftarrow d_1 - r$). Otherwise, if $d_1 \le d_2$, then the device chooses a random integer r where $0 \le r \le d_2$, adds r to d_1 (i.e., $d_1 \leftarrow d_1 + r$), and subtracts r from d_2 (i.e., $d_2 \leftarrow d_2 - r$). Then, to perform the private key operation on a message M, the device computes $s_1 = M^{d_1} \bmod n$, $s_2 = M^{d_2} \bmod n$, and computes the signature $S = s_1 s_2 \bmod n$. While this approach of splitting the exponent into two halves whose sum equals the exponent can also be used with Diffie-Hellman and other cryptosystems, dividing the exponent into the product of two numbers mod ϕ(p) is usually preferable since the assumption that information about d_1+d_2 will not leak is less conservative than the assumption that information about $x_1 x_2 \bmod \phi(p)$ will not leak. In the case of RSA, updates mod $\phi(n)$ cannot be done safely, since $\phi(n)$ must be kept secret.

When the Chinese Remainder Theorem is required for performance, it is possible to use similar techniques to add leak resistance by maintaining multiples of the secret primes (p and q) that are updated every time (e.g., multiplying by the new multiple then dividing by the old multiple). These techniques also protect the exponents (d_p and d_q) as multiples of their normal values. At the end of the operation, the result S is corrected to compensate for the adjustments to d_p, d_q, p, and q.

An exemplary embodiment maintains state information consisting of the values n, B_i, B_p, k, p_k, q_k, d_{pk}, d_{qk}, p_{Inv}, and f. To convert a traditional RSA CRT private key (consisting of p, q, d_p, and d_q with $p < q$) into the new representation, a random value for k is chosen, where $0 < k < 2^{64}$. The value B_i is chosen at random where $0 < B_i < n$, and R_1 and R_2 are chosen at

random where $0 < R_1 < 2^{64}$ and $0 < R_2 < 2^{64}$. (Of course, constants such as 2^{64} are chosen as example values. It is possible, but not necessary, to place constraints on random numbers, such as requiring that they be prime.) The leak-resistant private key state is then initialized by setting $n \leftarrow pq$, $B_f \leftarrow B_i^{-d} \bmod n$, $p_k \leftarrow (k)(p)$, $q_k \leftarrow (k)(q)$, $d_{pk} \leftarrow d_p + (R_1)(p) - R_1$, $d_{qk} \leftarrow d_q + (R_2)(q) - R_2$, $p_{Inv} \leftarrow k(p^{-1} \bmod q)$, and $f \leftarrow 0$.

To update the system state, first a random value α may be produced where $0 < \alpha < 2^{64}$. Then compute $p_k \leftarrow ((\alpha)(p_k)) / k$, $q_k \leftarrow ((\alpha)(q_k)) / k$, $p_{Inv} \leftarrow ((\alpha)(p_{Inv})) / k$, $k \leftarrow \alpha$. The exponents d_{pk} and d_{qk} may be updated by computing $d_{pk} \leftarrow d_{pk} \pm (R_3 p_k - R_3 k)$ and $d_{qk} \leftarrow d_{qk} \pm (R_4 q_k - R_4 k)$, where R_3 and R_4 can be random or constant values (even 1). The blinding factors B_i and B_f may be updated by computing $B_i = B_i^2 \bmod n$ and $B_f = B_f^2 \bmod n$, by computing new blinding factors, by exponentiating with a value other than 2, etc. Update processes should be performed as often as practical, for example before or after each modular exponentiation process. Before the update begins, a failure counter f is incremented, and when the update completes f is set to zero. If f ever exceeds a threshold value indicating too many consecutive failures, the device should temporarily or permanently disable itself. Note that if the update process is interrupted, memory values should not be left in intermediate states. This can be done by using complete reliable memory updates. If the total set of variable changes is too large for a single complete update, it is possible to store α first then do each variable update reliably which keeping track of how many have been completed.

To perform a private key operation (such as decryption or signing), the input message C is received by the modular exponentiator. Next, the value is blinded by computing $C' \leftarrow (C)(B_i) \bmod n$. The blinded input message is then used to compute modified CRT intermediates by computing $m_{pk} \leftarrow (C')^{d_{pk}} \bmod p_k$ and $m_{qk} \leftarrow (C')^{d_{qk}} \bmod q_k$. Next in the exemplary embodiment, the CRT intermediates are multiplied by k, e.g. $m_{pk} \leftarrow (k)(m_{pk}) \bmod p_k$ and $m_{qk} \leftarrow (k)(m_{qk}) \bmod q_k$. The CRT difference is then computed as $m_{pqk} = (m_{pk} [+ q_k] - m_{qk}) [\bmod q_k]$, where the addition of q_k and/or reduction mod q_k are optional. (The addition of q_k ensures that the result is non-negative.) The blinded result can be computed as

$$M' = \frac{(m_{pk})k + p_k\left[\left(\frac{(p_{Inv})(m_{pqk})}{k}\right) \bmod q_k\right]}{k^2}$$

, then the final result M is computed as $M = (M')B_f \bmod n$.

As one of ordinary skill in the art will appreciate, variant forms of the invention are possible. For example, the computational processes can be re-ordered or modified without significantly changing the invention. Some portions (such as the initial and blinding steps) can be skipped. In another example, it is also possible to use multiple blinding factors (for example, instead of or in addition to the value k).

In some cases, other techniques may also be appropriate. For example, exponent vector codings may be rechosen frequently using, for example, a random number generator. Also, Montgomery arithmetic may be performed mod j where j is a value that is changed with each operation (as opposed to traditional Montgomery implementations where j is constant with $j = 2^k$). The foregoing shows that the method and apparatus of the present invention can be implemented using numerous variations and modifications to the exemplary embodiments described herein, as would be known by one skilled in the art.

C. Leak-Resistant ElGamal Public Key Encryption and Digital Signatures

Still other asymmetric cryptographic protocols that may be improved using the techniques of the invention. For example, ElGamal and related cryptosystems are widely used for digital signatures and public key encryption. If information about the secret exponents and parameters leaks from an ElGamal implementation, security can be compromised. Consequently, leak-resistant implementations of ElGamal would be useful.

The private key in the ElGamal public key encryption scheme is a randomly selected secret a where $1 \le a \le p-2$. The non-secret parameters are a prime p, a generator α, and $\alpha^a \bmod p$. To encrypt a message m, one selects a random k (where $1 \le k \le p-2$) and computes the ciphertext (γ, δ) where $\gamma = \alpha^k \bmod p$ and $\delta = m(\alpha^a \bmod p)^k \bmod p$. Decryption is performed by computing $m = \delta(\gamma^{p-1-a}) \bmod p$. (See the Handbook of Applied Cryptography by A. Menezes, P. van Oorschot, and S. Vanstone, 1997, pages 294-298, for a description of ElGamal public-key encryption).

To make the ElGamal public-key decryption process leak-resistant, the secret exponent $(p - 1 - a)$ is stored in two halves a_1 and a_2, such that $a_1 a_2 = (\phi(p)-a) \bmod \phi(p)$. When generating ElGamal parameters for this leak-resistant implementation, it is recommended, but not required, that p be chosen with $\frac{p-1}{2}$ prime so that $\phi(p)/2$ is prime. The variables a_1 and a_2 are normally chosen initially as random integers between 0 and $\phi(p)$.

Alternatively, it is possible to generate a first, then choose a_1 and a_2, as by selecting a_1 relatively prime to $\phi(p)$ and computing $a_2 = (a^{-1} \bmod \phi(p))(a_1^{-1} \bmod \phi(p)) \bmod \phi(p)$.

Figure 4 illustrates an exemplary leak-resistant ElGamal decryption process. At step 405, the decryption device receives an encrypted message pair (γ, δ). At step 410, the device selects a random r_1 where $1 \le r_1 < \phi(p)$ and $\gcd(r_1, \phi(p)) = 1$. At step 415, the device updates a_1 by computing $a_1 \leftarrow a_1 r_1 \bmod \phi(p)$, over-writing the old value of a_1 with the new value. At step 420, the device computes the inverse of r_1 by computing $r_2 = (r_1)^{-1} \bmod \phi(p)$. Because r_1 is not used after this step, its storage space may be used to hold r_2. Note that if $\frac{p-1}{2}$ is prime, then r_2 may also be found by finding $r_2' = r_1^{(p-1)/2-2} \bmod \frac{p-1}{2}$, and using the CRT to find $r_2 \pmod{p-1}$. At step 425, the device updates a_2 by computing $a_2 \leftarrow a_2 r_2 \bmod \phi(p)$. At step 430, the device begins the private key (decryption) process by computing $m' = \gamma^{a_1} \bmod p$. At step 435, the device computes $m = \delta (m')^{a_2} \bmod p$ and returns the message m. If verification is successful, the result equals the original message because:

$$\begin{aligned}(\delta)(m')^{a_2} \bmod p &= \left(m(\alpha^a)^k\right)\left(\gamma^{a_1} \bmod p\right)^{a_2} \bmod p \\ &= \left(m\alpha^{ak}\right)\left(\gamma^{a_1 a_2 \bmod \phi(p)}\right) \bmod p \\ &= \left(m\alpha^{ak}\right)\left((\alpha^k \bmod p)^{-a \bmod \phi(p)}\right) \bmod p \\ &= \left(m\alpha^{ak}\right)\left(\alpha^{-ak}\right) \bmod p \\ &= m\end{aligned}$$

As with the ElGamal public key encryption scheme, the private key for the ElGamal digital signature scheme is a randomly-selected secret a, where $1 \le a \le p-2$. The public key is also similar, consisting of a prime p, a generator α, and public parameter y where $y = \alpha^a \bmod p$. To sign a message m, the private key holder chooses or precomputes a random secret integer k (where $1 \le k \le p-2$ and k is relatively prime to $p-1$) and its inverse, $k^{-1} \bmod \phi(p)$. Next, the signer computes the signature (r, s), where $r = \alpha^k \bmod p$, $s = \left(\left(k^{-1} \bmod \phi(p)\right)\left[\mathrm{H}(m) - ar\right]\right) \bmod \phi(p)$, and $\mathrm{H}(m)$ is the hash of the message. Signature verification is performed using the public key (p, α, y) by verifying that $1 \le r < p$ and by verifying that $y^r r^s \bmod p = \alpha^{\mathrm{H}(m)} \bmod p$.

To make the ElGamal digital signing process leak-resistant, the token containing the private key maintains three persistent variables, a_k, w, and r. Initially, $a_k = a$ (the private exponent), $w = 1$, and $r = \alpha$. When a message m is to be signed (or during the

precomputation before signing), the token generates a random number b and its inverse b^{-1} mod $\phi(p)$, where b is relatively prime to $\phi(p)$ and $0 < b < \phi(p)$. The token then updates a_k, w, and r by computing $a_k \leftarrow (a_k)(b^{-1}) \bmod \phi(p)$, $w \leftarrow (w)(b^{-1}) \bmod \phi(p)$, and $r \leftarrow (r^b) \bmod p$. The signature (r, s) is formed from the updated value of r and s, where $s = (w(\mathrm{H}(m) - a_k r)) \bmod \phi(p)$. Note that a_k, w, and r are not randomized prior to the first operation, but should be randomized before exposure to possible attack, since otherwise the first operation may leak more information than subsequent ones. It is thus recommended that a dummy signature or parameter update with $a_k \leftarrow (a_k)(b^{-1}) \bmod \phi(p)$, $w \leftarrow (w)(b^{-1}) \bmod \phi(p)$, and $r \leftarrow (r^b) \bmod p$ be performed immediately after key generation. Valid signatures produced using the exemplary tamper-resistant ElGamal process may be checked using the normal ElGamal signature verification procedure.

It is also possible to split all or some the ElGamal variables into two halves as part of the leak resistance scheme. In such a variant, a is replaced with a_1 and a_2, w with w_1 and w_2, and r with r_1 and r_2. It is also possible to reorder the operations by performing, for example, the parameter updates as a precomputation step prior to receipt of the enciphered message. Other variations and modifications to the exemplary embodiments described herein will be evident to one skilled in the art.

D. Leak-Resistant DSA

Another commonly used asymmetric cryptographic protocol is the Digital Signature Algorithm (DSA, also known as the Digital Signature Standard, or DSS), which is defined in "Digital Signature Standard (DSS)," Federal Information Processing Standards Publication 186, National Institute of Standards and Technology, May 19, 1994 and described in detail in the Handbook of Applied Cryptography, pages 452 to 454. DSA is widely used for digital signatures. If information about the secret key leaks from a DSA implementation, security can be compromised. Consequently, leak-resistant implementations of DSA would be useful.

In non-leak-proof systems, the private key consists of a secret parameter a, and the public key consists of (p, q, α, y), where p is a large (usually 512 to 1024 bit) prime, q is a 160-bit prime, α is a generator of the cyclic group of order q mod p, and $y = \alpha^a \bmod p$. To sign a message whose hash is $\mathrm{H}(m)$, the signer first generates (or precomputes) a random integer k and its inverse k^{-1} mod q, where $0 < k < q$. The signer then computes the signature (r, s), where $r = (\alpha^k \bmod p) \bmod q$, and $s = (k^{-1} \bmod q)(\mathrm{H}(m) + ar) \bmod q$.

In an exemplary embodiment of a leak-resistant DSA signing process, the token containing the private key maintains two variables in nonvolatile memory, a_k and k, which are initialized with $a_k = a$ and $k = 1$. When a message m is to be signed (or during the precomputation before signing), the token generates a random integer b and its inverse b^{-1} mod q, where $0 < b < q$. The token then updates a_k and k by computing $a_k \leftarrow (a_k b^{-1} \bmod q)(k) \bmod q$, followed by $k \leftarrow b$. The signature (r, s) is formed from the updated values of a_k and k by computing $r = \alpha^k \bmod p$ (which may be reduced mod q), and $s = [(b^{-1}\mathrm{H}(m) \bmod q) + (a_k r) \bmod q] \bmod q$. As indicated, when computing s, $b^{-1}\mathrm{H}(m) \bmod q$ and $(a_k r) \bmod q$ are computed first, then combined mod q. Note that a_k and k should be randomized prior to the first operation, since the first update may leak more information than subsequent updates. It is thus recommended that a dummy signature (or parameter update) be performed immediately after key generation. Valid signatures produced using the leak-resistant DSA process may be checked using the normal DSA signature verification procedure.

IV. Other Algorithms and Applications

Still other cryptographic processes can be made leak-proof or leak-resistant, or may be incorporated into leak-resistant cryptosystems. For example, cryptosystems such as those based on elliptic curves (including elliptic curve analogs of other cryptosystems), secret sharing schemes, anonymous electronic cash protocols, threshold signatures schemes, etc. be made leak resistant using the techniques of the present invention.

Implementation details of the schemes described may be adjusted without materially changing the invention, for example by re-ordering operations, inserting steps, substituting equivalent or similar operations, etc. Also, while new keys are normally generated when a new system is produced, it is often possible to add leak resistance retroactively while maintaining or converting existing private keys.

Leak-resistant designs avoid performing repeated mathematical operations using non-changing (static) secret values, since they are likely to leak out. However, in environments where it is possible to implement a simple function (such as an exclusive OR) that does not leak information, it is possible use this function to implement more complex cryptographic operations.

While the exemplary implementations assume that the leak functions can reveal any information present in the system, designers may often safely use the (weaker) assumption

that information not used in a given operation will not be leaked by that operation. Schemes using this weaker assumption may contain a large table of precomputed subkey values, from which a unique or random subset are selected and/or updated for each operation. For example, DES implementations may use indexed permutation lookup tables in which a few table elements are exchanged with each operation.

While leak resistance provides many advantages, the use of leak resistance by itself cannot guarantee good security. For example, leak-resistant cryptosystems are not inherently secure against error attacks, so operations should be verified. (Changes can even be made to the cryptosystem and/or leak resistance operations to detect errors.) Similarly, leak resistance by itself does not prevent attacks that extract the entire state out of a device (e.g., $L=L_{MAX}$). For example, traditional tamper resistance techniques may be required to prevent attackers from staining ROM or EEPROM memory cells and reading the contents under a microscope. Implementers should also be aware of interruption attacks, such as those that involve disconnecting the power or resetting a device during an operation, to ensure that secrets will not be compromised or that a single leaky operation will not be performed repeatedly. (As a countermeasure, devices can increment a counter in nonvolatile memory prior to each operation, and reset or reduce the counter value when the operation completes successfully. If the number of interrupted operations since the last successful update exceeds a threshold value, the device can disable itself.) Other tamper resistance mechanisms and techniques, such as the use of fixed-time and fixed-execution path code or implementations for critical operations, may need to be used in conjunction with leak resistance, particularly for systems with a relatively low self-healing rate (e.g., L_{MAX} is small).

Leak-resistant algorithms, protocols, and devices may be used in virtually any application requiring cryptographic security and secure key management, including without limitation: smartcards, electronic cash, electronic payments, funds transfer, remote access, timestamping, certification, certificate validation, secure e-mail, secure facsimile, telecommunications security (voice and data), computer networks, radio and satellite communications, infrared communications, access control, door locks, wireless keys, biometric devices, automobile ignition locks, copy protection devices, payment systems, systems for controlling the use and payment of copyrighted information, and point of sale terminals.

The foregoing shows that the method and apparatus of the present invention can be implemented using numerous variations and modifications to the exemplary embodiments described herein, as would be known by one skilled in the art. Thus, it is intended that the scope of the present invention be limited only with regard to the claims below.

WHAT IS CLAIMED IS:

1. A method for implementing RSA with the Chinese Remainder Theorem for use in a cryptographic system, with resistance to leakage attacks against said cryptographic system, comprising the steps of:

(a) obtaining a representation of an RSA private key corresponding to an RSA public key, said private key characterized by secret factors p and q;

(b) storing said representation of said private key in a memory;

(c) obtaining a message for use in an RSA cryptographic operation;

(d) computing a first modulus, corresponding to a multiple of p, where the value of said multiple of p and the value of said multiple of p divided by p are both unknown to an attacker of said cryptographic system;

(e) reducing said message modulo said first modulus;

(f) performing modular exponentiation on the result of step (e);

(g) computing a second modulus, corresponding to a multiple of q, where the value of said multiple of q and the value of said multiple of q divided by q are both unknown to an attacker of said cryptographic system;

(h) reducing said message modulo said second modulus;

(i) performing modular exponentiation on the result of step (h);

(j) combining the results of said steps (e) and (h) to produce a result which, if operated on with an RSA public key operation using said RSA public key, yields said message; and

(k) repeating steps (c) through (j) a plurality of times using different values for said multiple of p and for said multiple of q.

2. The method of claim 1 where:

(i) said step (b) includes storing an exponent d_p of said RSA private key in said memory as a plurality of parameters;

(ii) an arithmetic function of at least one of said plurality of parameters is congruent to d_p, modulo (p-1);

(iii) none of said parameters comprising said stored d_p is equal to d_p;

(iv) an exponent used in said step (f) is at least one of said parameters;

(v) at least one of said parameters in said memory changes with said repetitions of said steps (c) through (j).

3. The method of claim 2 where said plurality of parameters includes a first parameter equal to said d_p plus a multiple of phi(p), and also includes a second parameter equal to a multiple of phi(p), where phi denotes Euler's totient function.

4. The method of claim 1 where the value of said multiple of p divided by p is equal to the value of said multiple of q divided by q.

5. The method of claim 1 where said multiple of p and said multiple of q used in said steps (c) through (j) are updated and modified in said memory after said step (b).

6. The method of claim 1 performed in a smart card.

7. The method of claim 1 where at least two of said steps are performed in an order other than (a) through (k)

8. A method for implementing RSA for use in a cryptographic system, with resistance to leakage attacks against said cryptographic system, comprising the steps of:

(a) obtaining an RSA private key corresponding to an RSA public key, said RSA public key having an RSA modulus n;

(b) storing said private key in a memory in a form whereby a secret parameter of said key is stored as an arithmetic combination of phi(x) and a first at least one key masking parameter, where

(i) an operand x in said phi(x) is an exact multiple of at least one factor of said modulus n of said RSA public key; and

(ii) said first key masking parameter is unknown to an attacker of said cryptosystem;

(iii) a representation of said first key masking parameter is stored in said memory;

(iv) phi denotes Euler's totient function;

(c) receiving a message;

(d) deriving an RSA input from said message;

(e) performing modular exponentiation to raise said RSA input to a power dependent on said secret parameter, modulo an RSA modulus stored in said memory, to produce an RSA result such that said RSA result raised to the

power of the public exponent of said RSA public key, modulo the modulus of said RSA public key, equals said RSA input;

(f) updating said secret parameter in said memory by:

(i) modifying said first key masking parameter to produce a new key masking parameter, where said modification is performed in a manner such that an attacker with partial useful information about said first key masking parameter has less useful information about said new key masking parameter; and

(ii) using said new key masking parameter to update said secret parameter in said memory;

(g) repeating steps (d) through (f) a plurality of times, where the power used for each of said modular exponentiation steps (e) is different.

9. The method of claim 8 where said operand x in said phi(x) corresponds to said RSA modulus n of said RSA public key.

10. The method of claim 8 where said operand x in said phi(x) corresponds to a prime factor of said RSA modulus n of said RSA public key, and where said modular exponentiation of said step (e) is performed using the Chinese Remainder Theorem.

11. A method for implementing exponential key exchange for use in a cryptographic system, with resistance to leakage attacks against said cryptographic system, comprising the steps of:

(a) obtaining, and storing in a memory, exponential key exchange parameters g and p, and a plurality of secret exponent parameters on which an arithmetic relationship may be computed to produce an exponent x;

(b) using a key update transformation to produce a plurality of updated secret exponent parameters while maintaining said arithmetic relationship thereamong;

(c) receiving a public value y from a party with whom said key exchange is desired;

(d) using said updated secret exponent parameters to perform a cryptographic computation yielding an exponential key exchange result z = y^x mod p;

(e) using said result z to secure an electronic communication with said party; and

(f) performing said steps (b), (c), (d), and (e) in a plurality of transactions.

12. The method of claim 11 where each of said transactions involves a different said party.

13. The method of claim 11 where said arithmetic relationship is such that said exponential key exchange result is a product of certain of said secret exponent parameters, both before and after said step (b).

14. The method of claim 11 where said key update transformation includes choosing a random key update value r; and where said step (b) includes multiplying one of said secret exponent parameters by r and another of said secret exponent parameters by an inverse of r, said multiplication being performed modulo phi(p), where phi is Euler's totient function.

15. The method of claim 11 where said key update transformation includes choosing a random key update value r; and where said step (b) includes adding r to one of said secret exponent parameters and subtracting r from another of said secret exponent parameters.

16. The method of claim 15 where said secret exponent parameters include two values x_1 and x_2 such that x_1+x_2 is congruent to x, modulo phi(p), where phi is Euler's totient function, and where said step of performing said cryptographic computation yielding said exponential key exchange result includes computing $z_1 = y\hat{}x_1 \bmod p$, $z_2 = y\hat{}x_2 \bmod p$, and $z=z_1z_2 \bmod p$.

17. A cryptographic token configured to perform cryptographic operations using a secret key in a secure manner, comprising:

(a) an interface configured to receive power from a source external to said token;

(b) a memory containing said secret key;

(c) a processor:

(i) configured to receive said power delivered via said interface;

(ii) configured to perform said processing using said secret key from said memory;

(d) said token having a power consumption characteristic:

(i) that is externally measurable; and

(ii) that varies over time in a manner measurably correlated with said cryptographic operations; and

(e) a source of unpredictable information usable in said cryptographic operations to make determination of said secret key infeasible from external measurements of said power consumption characteristic.

18. The cryptographic token of claim 17, in the form of a secure microprocessor.

19. The cryptographic token of claim 17, in the form of a smart card.

20. The cryptographic token of claim 19, wherein said cryptographic operations performed by said smart card enable a holder thereof to decrypt an encrypted communication received via a computer network.

21. The cryptographic token of claim 19, wherein said smart card is configured to store value in an electronic cash scheme.

22. The cryptographic token of claim 21, wherein said cryptographic operations include authenticating that a balance of said stored value has been decreased.

23. The cryptographic token of claim 17, wherein said cryptographic operations include asymmetric private key operations.

24. The cryptographic token of claim 23 wherein said cryptographic operations include exponential key agreement operations.

25. The cryptographic token of claim 23, wherein said cryptographic operations include DSA signing operations.

26. The cryptographic token of claim 23, wherein said cryptographic operations include ElGamal private key operations.

27. The cryptographic token of claim 23, wherein said asymmetric private key operations include RSA private key operations.

28. The cryptographic token of claim 27 wherein said private key operations include Chinese Remainder Theorem operations.

29. The cryptographic token of claim 17, wherein said cryptographic operations include symmetric encryption operations.

30. The cryptographic token of claim 17, wherein said cryptographic operations include symmetric decryption operations.

31. The cryptographic token of claim 17, wherein said cryptographic operations include symmetric authentication operations using said secret key.

32. The cryptographic token of claim 17, wherein said cryptographic operations include authenticating a payment.

33. The cryptographic token of claim 17, wherein said cryptographic operations include securing a broadcast communications signal.

34. The cryptographic token of claim 33, wherein said cryptographic operations include decrypting a satellite broadcast.

35. A method for securely managing and using a private key in a computing environment where information about said private key may leak to attackers, comprising the steps of:

(a) using a first private key, complementary to a public key, to perform first asymmetric cryptographic operation;

(b) reading at least a portion of said first private key from a memory;

(c) transforming said read portion of said first private key to produce a second private key:

(i) said second private key usable to perform a subsequent asymmetric cryptographic operation in a manner that remains complementary to said public key, and

(ii) said transformation enabling said asymmetric cryptographic operations to be performed in a manner such that information leaked during said

first asymmetric cryptographic operation does not provide incrementally useful information about said second private key;

(d) obtaining a datum;

(e) using said second private key to perform said subsequent asymmetric cryptographic operation on said datum.

36. The method of claim 35 where said asymmetric cryptographic operation includes a digital signing operation.

37. The method of claim 36 where said signing operation is an RSA operation.

38. The method of claim 36 where said signing operation is an DSA operation.

39. The method of claim 36 where said signing operation is an ElGamal operation.

40. The method of claim 35 where said asymmetric cryptographic operation includes a decryption operation.

41. The method of claim 40 where said decryption operation is an RSA operation.

42. The method of claim 40 where said decryption operation is an ElGamal operation.

43. The method of claim 35 where at least two of said steps are performed in an order different than (a), (b), (c), (d), (e).

44. The method of claim 35 further comprising the step, after at least said step (c), of replacing said private key in said memory with said second private key.

45. The method of claim 35, performed in a smart card.

46. The method of claim 35, further comprising the steps of: prior to at least said step (c), incrementing a counter stored in a nonvolatile memory and verifying that said counter has not exceeded a threshold value; and after at least said step (c) has completed successfully, decreasing a value of said counter.

47. A method for performing cryptographic transactions while protecting a stored cryptographic key against compromise due to leakage attacks, comprising the steps of:

(a) retrieving a stored private cryptographic key stored in a memory, said stored key having been used in a previous cryptographic transaction;

(b) using a first cryptographic function to derive from said stored key an updated key, about which useful information about said stored key obtained through monitoring of leaked information is effectively uncorrelated to said updated key;

(c) replacing said stored key in said memory with said updated key;

(d) using an asymmetric cryptographic function, cryptographically processing a datum with said updated key; and

(e) sending said cryptographically processed datum to an external device having a public key corresponding to said stored key.

48. The method of claim 47 where said stored key includes a first plurality of parameters, and where said updated key includes a second plurality of parameters.

49. The method of claim 48 where no secret value within said first plurality of parameters is included within said second plurality of parameters.

50. The method of claim 49 where said first plurality of parameters is different than said second plurality of parameters, yet a predetermined relationship among said first plurality of parameters is also maintained among said second plurality of parameters.

51. The method of claim 50 where said relationship among said plurality of parameters is an arithmetic function involving at least two of said plurality of parameters.

52. The method of claim 51 where said arithmetic function is the sum of said parameters.

53. The method of claim 51 where said relationship includes a bitwise combination of said parameters.

54. The method of claim 53 where said bitwise combination is an exclusive OR.

55. The method of claim 47 where said step (b) includes using pseudorandomness to derive said updated key.

56. A method for implementing a private key operation for an asymmetric cryptographic system with resistance to leakage attacks against said cryptographic system, comprising the steps of:

(a) encoding a portion of a private key as at least two component parts, such that an arithmetic function of said parts yields said portion;

(b) modifying said component parts to produce updated component parts, but where said arithmetic function of said updated parts still yields said private key portion;

(c) obtaining a message for use in an asymmetric private key cryptographic operation;

(d) separately applying said component parts to said message to produce an intermediate result;

(e) deriving a final result from said intermediate result such that said final result is a valid result of applying said private key to said message; and

(f) repeating steps (b) through (e) a plurality of times.

57. The method of claim 56 where said private key portion includes an exponent, and where said intermediate result represents the result of raising said message to the power of said exponent, modulo a second key portion.

58. The method of claim 57 where said private key operation is configured for use with an RSA cryptosystem.

59. The method of claim 57 where said private key operation is configured for use with an ElGamal cryptosystem.

60. The method of claim 56 where said private key operation is configured for use with a DSA cryptosystem.

61. The method of claim 60 where said private key is represented by secret parameters a_k and k whose product, modulo a predetermined DSA prime q for said private key, yields said private key portion.

62. The method of claim 56 implemented in a smart card.

63. The method of claim 56 where said private key is configured for use with an elliptic curve cryptosystem.

64. A method for performing cryptographic transactions in a cryptographic token while protecting a stored cryptographic key against compromise due to leakage attacks, including the steps of:

(a) retrieving said stored key from a memory;

(b) cryptographically processing said key, to derive an updated key, by executing a cryptographic update function that:

(i) prevents partial information about said stored key from revealing useful information about said updated key, and

(ii) also prevents partial information about said updated key from revealing useful information about said stored key;

(c) replacing said stored key in said memory with said updated key;

(d) performing a cryptographic operation using said updated key; and

(e) repeating steps (a) through (d) a plurality of times.

65. The method of claim 64 where said cryptographic update function of said step (b) includes a one-way hash operation.

66. The method of claim 64 where said cryptographic operation of said step (d) is a symmetric cryptographic operation; and comprising the further step of sending a result of said cryptographic operation to a party capable of rederiving said updated key.

67. The method of claim 64 further comprising the step, prior to said step (a), of receiving from a second party a symmetric authentication code and a parameter; and said where said step (b) includes iterating a cryptographic transformation a number of times determined from said parameter; and where said step (d) includes performing a symmetric message authentication code verification operation.

68. he method of claim 66 where said step (d) of performing said cryptographic operation includes using said updated key to encrypt a datum.

69. The method of claim 66 where said updated key contains unpredictable information such that said updated key is not stored in its entirety anywhere outside of said cryptographic token; and where the result of said step (d) is independent of said unpredictable information.

70. The method of claim 64 where said step (c) of replacing said stored key includes:
(i) explicitly erasing a region of said memory containing said stored key; and
(ii) storing said updated key in said region of memory.

71. The method of claim 64 performed within a smart card.

FIG. 1

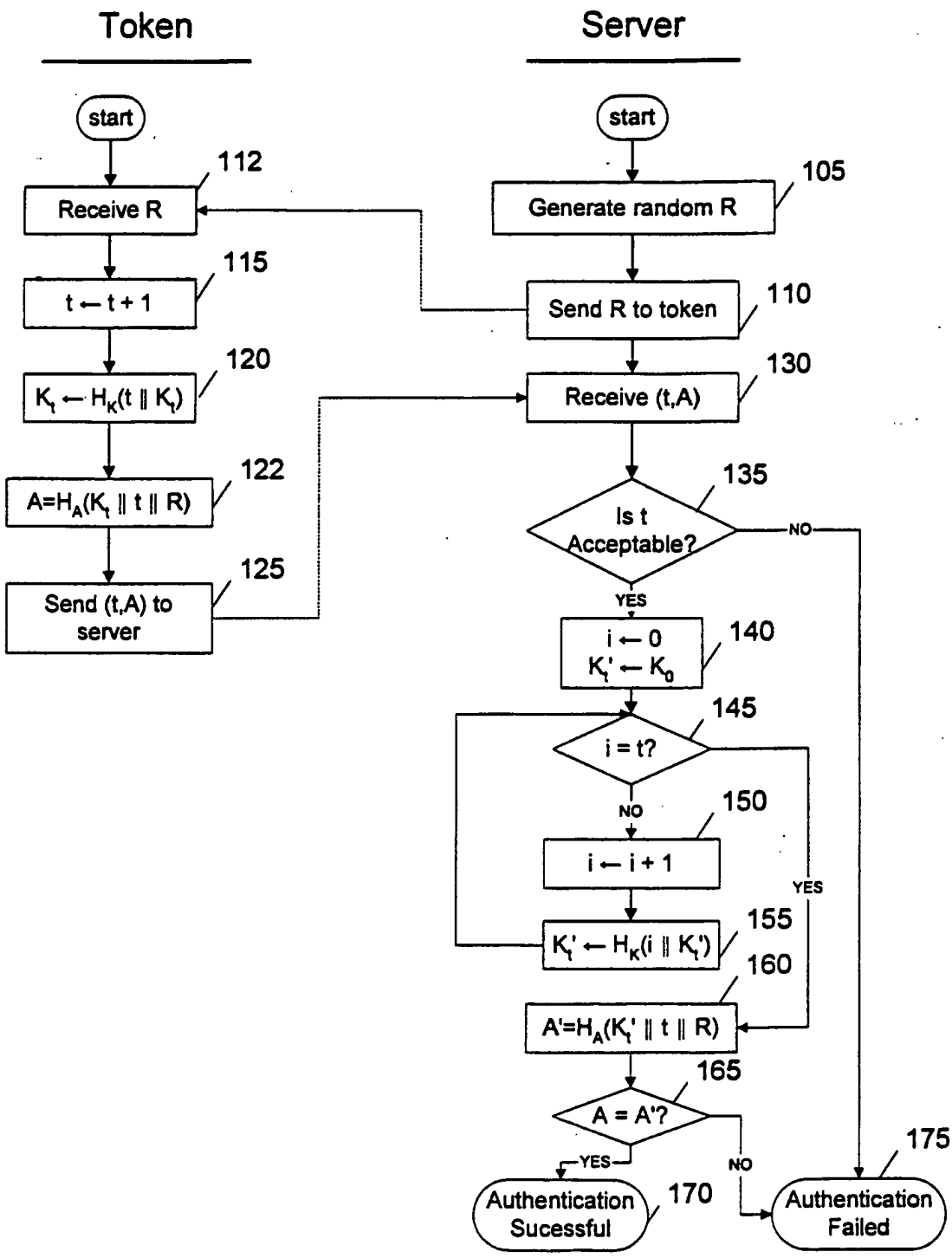


FIG. 2

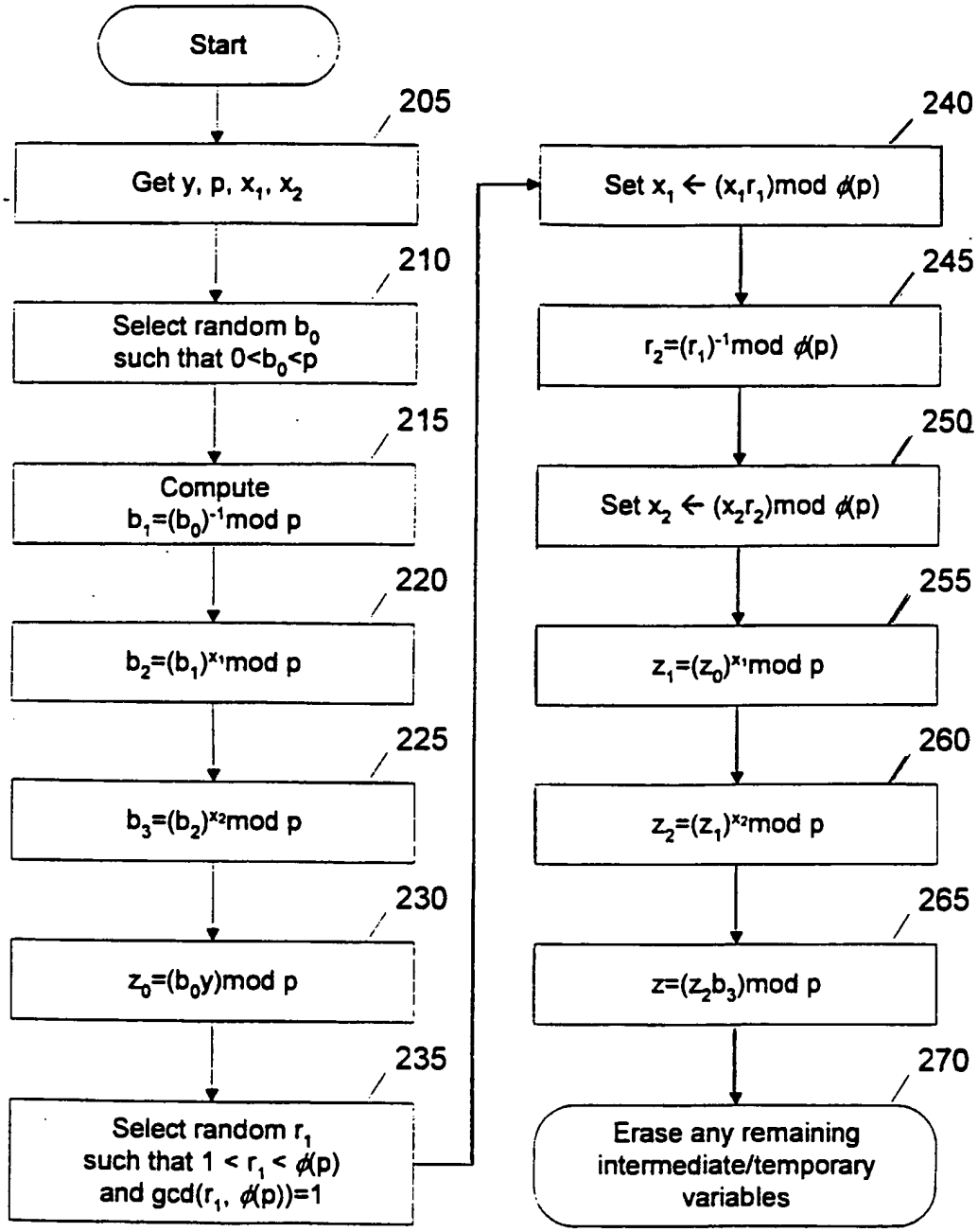


FIG. 3

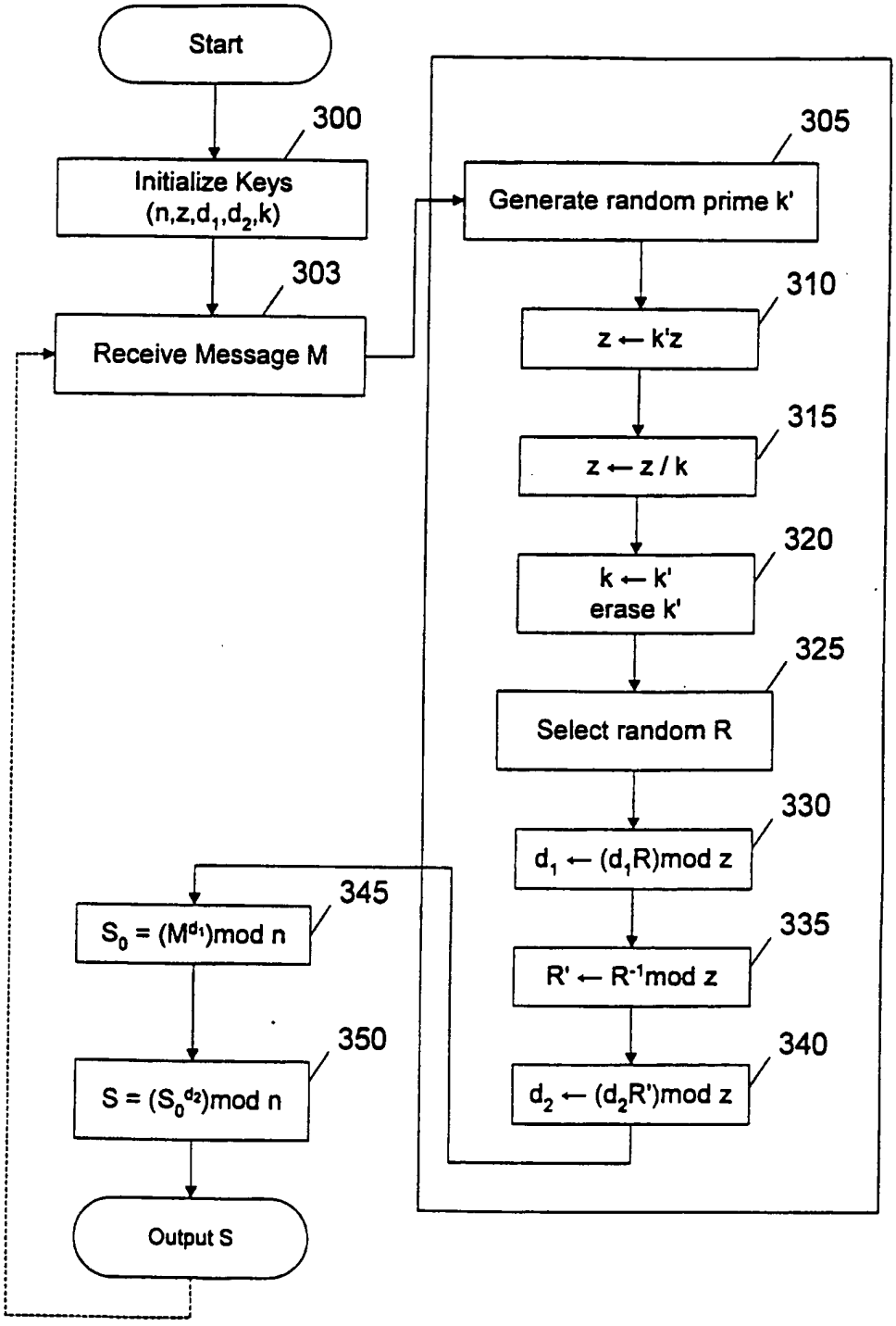
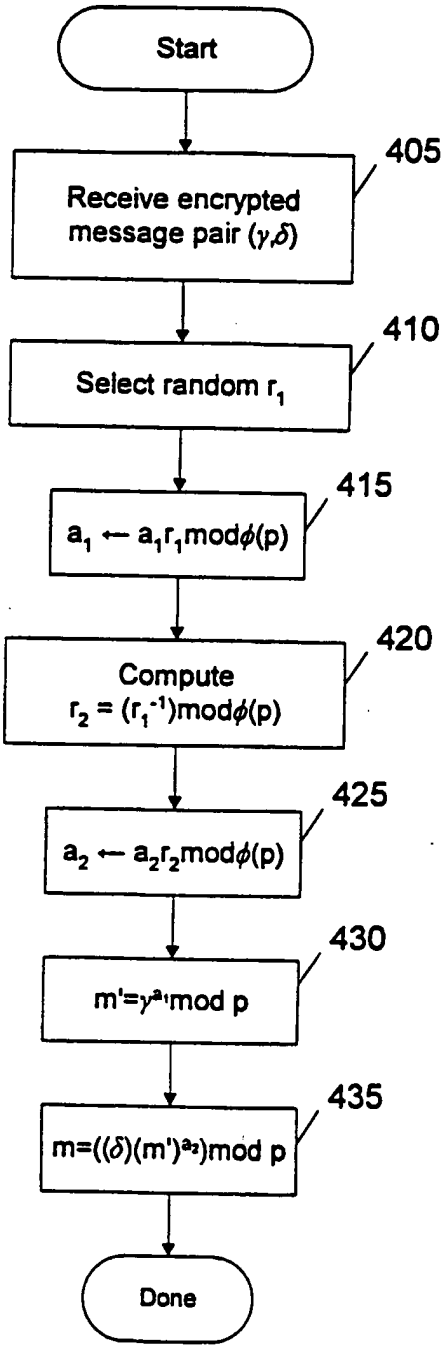


FIG. 4



INTERNATIONAL SEARCH REPORT

International application No.
PCT/US98/27896

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) :H04 L 9/30
US CL :380/30,49

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 380/30,49

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

Please See Extra Sheet.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 4,799,258 A (DAVIES et al) 17 January 1989, abstract, col.4, lines 43-50, col.7,lines 15-33, col.8,lines 12-19	17-23,25- 45
Y	US 5,546,463 A (CAPUTO et al) 13 August 1996, abstract, col.2, lines, 60-65, col.5, lines 39-50,53-58, col.6, lines 7-12	17-23,25-45
A,P	US 5,848,159 A (COLLINS et al.) 08 December 1998, abstract, col.1, lines 56-67, col.4, lines 33-44, col.5, lines 52-67, col.6, lines 24-30	1-16,46-71

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search 30 MARCH 1999	Date of mailing of the international search report 06 MAY 1999
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-0040	Authorized officer GAIL HAYES Telephone No. (703) 305-9711

Form PCT/ISA/210 (second sheet)(July 1992) ★

B. FIELDS SEARCHED
Electronic data bases consulted (Name of data base and where practicable terms used):

APS
search terms:token,smart card,tamper proof, tamper resistant,leak-resistant,RSA, public key, private key, chinese remainder theorem, diffie hellman,dsa,des

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 :
H04N 7/167, G06F 1/00

A1

(11) International Publication Number: **WO 99/48296**

(43) International Publication Date: 23 September 1999 (23.09.99)

(21) International Application Number: PCT/US99/05734

(22) International Filing Date: 16 March 1999 (16.03.99)

(30) Priority Data:
60/078,053 16 March 1998 (16.03.98) US

(71) Applicant: INTERTRUST TECHNOLOGIES CORPORATION [US/US]; 460 Oakmead Parkway, Sunnyvale, CA 94086 (US).

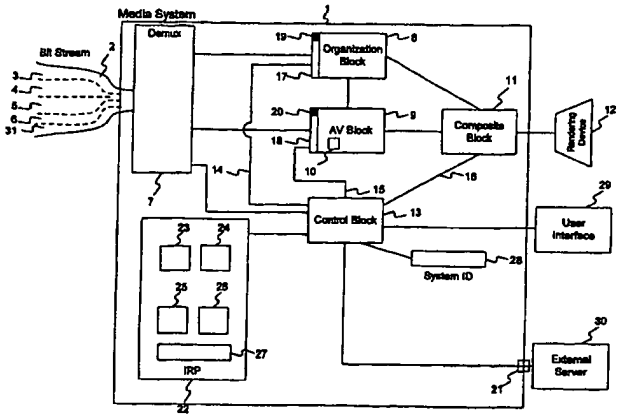
(72) Inventors: SHAMOON, Talal, G.; 533 Bryant Street #5, Palo Alto, CA 94301 (US). HILL, Ralph, D.; 224 Dover Street, Los Gatos, CA 94032 (US). RADCLIFFE, Chris, D.; 3654 Farm Hill Boulevard, Redwood City, CA 94061 (US). HWA, John, P.; 503 Lower Vinters Circle, Fremont, CA 94539 (US).

(74) Agents: GARRETT, Arthur, S. et al.; Finnegan, Henderson, Farabow, Garrett & Dunner, L.L.P., 1300 I Street, Washington, DC 20005-3315 (US).

(81) Designated States: CA, CN, JP, KR, European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: METHODS AND APPARATUS FOR CONTINUOUS CONTROL AND PROTECTION OF MEDIA CONTENT



(57) Abstract

A novel method and apparatus for protection of streamed media content is disclosed. The apparatus includes control means for governance of content streams or objects, decryption means for decrypting content streams or objects under control of the control means, and feedback means for tracking actual use of content streams or objects. The control means may operate in accordance with rules received as part of the streamed content, or through a side-band channel. The rules may specify allowed uses of the content, including whether or not the content can be copied or transferred, and whether and under what circumstances received content may be "checked out" of one device and used in a second device. The rules may also include or specify budgets, and a requirement that audit information be collected and/or transmitted to an external server. The apparatus may include a media player designed to call plugins to assist in rendering content. A "trust plugin" and its use are disclosed so that a media player designed for use with unprotected content may render protected content without the necessity of requiring any changes to the media player. The streamed content may be in a number of different formats, including MPEG-4, MP3, and the RMFF format.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

METHODS AND APPARATUS FOR CONTINUOUS CONTROL AND PROTECTION OF MEDIA CONTENT

FIELD OF THE INVENTION

This invention relates generally to computer and/or electronic security. More particularly, this invention relates to systems and methods for protection of information in streamed format.

BACKGROUND

Streaming digital media consists generally of sequences of digital information received in a "stream" of packets, and designed to be displayed or rendered. Examples include streamed audio content, streamed video, etc.

Digital media streams are becoming an increasingly significant means of content delivery, and form the basis for several adopted, proposed or de facto standards. The acceptance of this format, however, has been retarded by the ease with which digital media streams can be copied and improperly disseminated, and the consequent reluctance of content owners to allow significant properties to be distributed through streaming digital means. For this reason, there is a need for a methodology by which digital media streams can be protected.

SUMMARY OF THE INVENTION

Consistent with the invention, this specification describes a new architecture for protection of information provided in streamed format. This architecture is described in the context of a generic system which resembles a system to render content encoded pursuant to the MPEG-4 specification (ISO/IEC 14496.1), though with certain modifications, and with the proviso that the described system may differ from the MPEG-4 standard in certain respects. A variety of different embodiments is described, including an MPEG-4 embodiment and a system designed to render content encoded pursuant to the MP3 specification (ISO/IEC TR 11172).

According to aspects of the invention, this architecture involves system design aspects and information format aspects. System design aspects include the incorporation of content protection functionality, control functionality, and feedback enabling control functionality to monitor the activities of the system. Information format aspects include the incorporation of rule/control information into information streams, and the protection of content through mechanisms such as encryption and watermarking.

Systems and methods consistent with the present invention perform content protection and digital rights management. A streaming media player consistent with the present invention includes a port designed to accept a digital bit stream. The digital bit stream includes content, which is encrypted at least in part, and a secure container including control information designed to control use of the content, including at least one key suitable for decryption of at least a portion of the content. The media player also includes a control arrangement including a means for opening secure containers and extracting cryptographic keys, and means for decrypting the encrypted portion of the content.

BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings, which are incorporated in and constitute a part of this specification, illustrate an embodiment of the invention and, together with the description, serve to explain the advantages and principles of the invention. In the drawings,

FIG. 1 shows a generic system consistent with the present invention;

FIG. 2 shows an exemplary Header 201 consistent with the present invention;

FIG. 3 shows a general encoding format consistent with the present invention;

FIG. 4 illustrates one manner for storing a representation of a work consistent with the present invention;

FIG. 5 shows an example of a control message format;

FIG. 6 is a flow diagram illustrating one embodiment of the steps which take place using the functional blocks of FIG. 1;

FIG. 7 illustrates a form wherein the control messages may be stored in Control Block 13;

FIG. 8 shows MPEG-4 System 801 consistent with the present invention;

FIG. 9 shows an example of a message format;

FIG. 10 illustrates an IPMP table consistent with the present invention;

FIG. 11 illustrates a system consistent with the present invention;

FIG. 12 illustrates one embodiment of the DigiBox format;

FIG. 13 shows an example of a Real Networks file format (RMFF);

FIG. 14 shows an RNPFF format consistent with the present invention;

FIG. 15 illustrates the flow of changes to data in the Real Networks file format in an architecture consistent with the present invention;

FIG. 16 illustrates a standard Real Networks architecture;

FIG. 17 shows an exemplary architecture in which a trust plugin operates within the overall Real Networks architecture;

FIG. 18 shows a bit stream format consistent with the principles of the present invention;

FIG. 19 shows one embodiment of protection applied to the MP3 format;

FIG. 20 illustrates one embodiment of an MP3 player designed to process and render protected content;

FIG. 21 illustrates the flow of data in one embodiment in which a protected MPEG-4 file may be created consistent with the present invnetion;

FIG. 22 illustrates the flow of data in one embodiment in which control may be incorporated into an existing MPEG-4 stream consistent with the present invention;

FIG. 23 shows a system consistent with the principles of the present invention;

FIG. 24 shows a system consistent with the principles of the present invention;

FIG. 25 shows an example of an aggregate stream consistent with the present invention;

FIG. 26 illustrates a Header CMPO 2601 consistent with the present invention;

FIG. 27 shows exemplary Content Management Protection Objects consistent with the principles of the present invention; and

FIG. 28 shows an example of a CMPO Data Structure 2801 consistent with the present invention.

DETAILED DESCRIPTION

Reference will now be made in detail to implementations consistent with the principles of the present invention as illustrated in the accompanying drawings.

The following U.S. patents and applications, each of which is assigned to the assignee of the current application, are hereby incorporated in their entirety by reference: Ginter, et al., "Systems and Methods for Secure Transaction Management and Electronic Rights Protection," U.S. Patent Application Serial No. 08/964,333, filed on November 4, 1997 ("Ginter '333"); Ginter, et al., "Trusted Infrastructure Support Systems, Methods and Techniques for Secure electronic commerce, Electronic Transactions, Commerce Process Control Automation, Distributed Computing, and Rights Management, " U.S. Patent Application Serial No. 08/699,712, filed on August 12, 1996 ("Ginter '712") ; Van Wie, et al, "Steganographic Techniques for Securely Delivering Electronic Digital Rights Management Information Over Insecure Communications Channels, U.S. Patent Application Serial No. 08/689,606, filed on August 12, 1996 ("Van Wie") ; Ginter, et. al "Software Tamper Resistance and Secure Communication," U.S. Patent Application Serial No. 08/706,206, filed on August 30, 1996 ("Ginter, '206"); Shear, et al, "Cryptographic Methods, Apparatus & Systems for Storage Media Electronic Rights Management in

Closed & Connected Appliances, " U.S. Patent Application Serial No. 08/848,077, filed on May 15, 1997 ("Shear"); Collberg et al, "Obfuscation Techniques for Enhancing Software Security, " U.S. Patent Application Serial No. 09/095,346, filed on June 9, 1998 ("Collberg"); Shear, "Database Usage Metering and Protection System and Method," U.S. Patent No. 4,827,508, issued on May 2, 1989 ("Shear Patent").

FIG. 1 illustrates Media System 1, which is capable of accepting, decoding, and rendering streamed multimedia content. This is a generic system, though it includes elements based on the MPEG-4 specification. Media System 1 may include software modules, hardware (including integrated circuits) or a combination. In one embodiment, Media System 1 may include a Protected Processing Environment (PPE) as described in the Ginter '333 application.

In FIG. 1, Bit Stream 2 represents input information received by System 1. Bit Stream 2 may be received through a connection to an external network (e.g., an Internet connection, a cable hookup, radio transmission from a satellite broadcaster, etc.), or may be received from a portable memory device, such as a DVD player.

Bit Stream 2 is made up of a group of related streams of information, including Organization Stream 3, Audio Stream 4, Video Stream 5, Control Stream 6, and Info Stream 31. Each of these streams is encoded into the overall Bit Stream 2. Each of these represents a category of streams, so that, for example, Video Stream 5 may be made up of a number of separate Video Streams.

These streams correspond generally to streams described in the MPEG-4 format as follows:

Organization Stream 3 corresponds generally to the BIFS stream and the OD ("Object Descriptor") stream.

Audio Stream 4 and Video Stream 5 correspond generally to the Audio and Video streams.

Control Stream 6 corresponds generally to the IPMP stream.

Audio Stream 4 includes compressed (and possibly encrypted) digital audio information. This information is used to create the sound rendered and output by Media System 1. Audio Stream 1 may represent multiple audio streams. These multiple streams may act together to make up the audio output, or may represent alternative audio outputs.

Video Stream 5 includes compressed (and possibly encrypted) digital video information. This information is used to create the images and video rendered and output by Media System 1. Video Stream 5 may represent multiple video streams. These multiple streams may act together to make up the video output, or may represent alternative

video outputs.

Organization Stream 3 includes organizational information and metadata related to the work to be rendered. This information may include a tree or other organizational device which groups audio and video streams into objects. This information may also include metadata associated with the entire work, the objects, or the individual streams.

Control Stream 6 includes control information, divided generally into header information and messages. The header information includes an identifier for each discrete message. The content of the messages, which will be described further below, may include cryptographic keys and rules governing the use of content.

Info Stream 31 carries additional information associated with the content in other components of Bit Stream 2, including but not limited to graphics representing cover art, text for lyrics, coded sheet music or other notation, independent advertising content, concert information, fan club information, and so forth. Info Stream 31 can also carry system management and control information and/or components, such as updates to software or firmware in Media System 1, algorithm implementations for content-specific functions such as watermarking, etc.

Each of these streams is made up of packets of information. In one exemplary embodiment, each packet is 32 bytes in length. Since a single communications channel (e.g., a cable, a bus, an infrared or radio connection) contains packets from each of the streams, packets need to be identified as belonging to a particular stream. In a preferred embodiment, this is done by including a header which identifies a particular stream and specifies the number of following packets which are part of that stream. In another embodiment, each packet may include individual stream information.

Exemplary Header 201 is shown in FIG. 2. This header may generally be used for the Organization, Audio and Video Streams. A header for the Control Stream is described below. Header 201 includes Field 202, which includes a bit pattern identifying Header 201 as a header. Field 203 identifies the particular type of stream (e.g., Audio Stream, Organization Stream, Control Stream, etc.) Field 204 contains an Elementary Stream Identifier (ES_ID), which is used to identify the particular stream, and may be used in cases where multiple streams of a particular stream type may be encountered at the same time. Field 207 contains a time stamp, which is used by the system to synchronize the various streams, including rendering of the streams. Composite Block 11 may, for example, keep track of the elapsed time from the commencement of rendering. Time Stamp 207 may be used by Composite Block 11 to determine when each object is supposed to be rendered. Time Stamp 207 may therefore specify an elapsed time from commencement of rendering,

and Composite Block 11 may use that elapsed time to determine when to render the associated object.

Field 205 contains a Governance Indicator. Field 206 identifies the number of following packets which are part of the identified stream. In each case, the relevant information is encoded in a binary format. For example, Field 202 might include an arbitrary sequence of bits which is recognized as indicating a header, and Field 203 might include two bits, thereby allowing encoding of four different stream types.

Returning to FIG. 1, System 1 includes Demux 7, which accepts as input Bit Stream 2 and routes individual streams (sometimes referred to as Elementary Streams or "ESs") to appropriate functional blocks of the system.

Bit Stream 2 may be encoded in the format illustrated in FIG. 3. In this figure, Header 301 is encountered in the bit stream, with Packet 302 following, and so on through Packet 308.

When Demux 7 encounters Header 301, Demux 7 identifies Header 301 as a header and uses the header information to identify Packets 302-305 as organization stream packets. Demux 7 uses this information to route these packets to Organization Block 8. Demux 7 handles Header 306 in a similar manner, using the contained information to route Packets 307 and 308 to AV ("Audio Video") Block 9.

AV Block 9 includes Decompressor 10, which accepts Elementary Streams from Audio Stream 4 and Video Stream 5 and decompresses those streams. As decompressed, the stream information is placed in a format which allows it to be manipulated and output (through a video display, speakers, etc.). If multiple streams exist (e.g., two video streams each describing an aspect of a video sequence), AV Block 9 uses the ES_ID to assign each packet to the appropriate stream.

Organization Block 8 stores pointer information identifying particular audio streams and video streams contained in a particular object, as well as metadata information describing, for example, where the object is located, when it is to be displayed (e.g., the time stamp associated with the object), and its relationship to other objects (e.g., is one video object in front of or behind another video object). This organization may be maintained hierarchically, with individual streams represented at the lowest level, groupings of streams into objects at a higher level, complete scenes at a still higher level, and the entire work at the highest level.

FIG. 4 illustrates one manner in which Organization Block 8 may store a representation of a work. In this Figure, Tree 401 represents an entire audiovisual work. Branch 402 represents a high-level organization of the work. This may include, for

example, all of the video or possibly the audio and video associated with a particular scene.

Sub-Branch 403 represents a group of related video objects. Each such object may include an entire screen, or an individual entity within the screen. For example, Sub-Branch 403 may represent a background which does not change significantly from one shot to the next. If the video is moving between two points of reference (e.g., a conversation, with the camera point of view changing from one face to the other), Sub-Branch 404 could represent a second background, used in the second point of view.

Nodes 405 and 406 may represent particular video objects contained within the related group. Node 405 could, for example, represent a distant mountain range, while Node 406 represents a tree immediately behind one of the characters.

Each of the nodes specifies or contains a particular ES_ID, representing the stream containing the information used by that node. Node 405, for example, contains ES_ID 407, which identifies a particular video stream which contains compressed (and possibly encrypted) digital information representing the mountain range.

Composite Block 11 accepts input from Organization Block 8 and from AV Block 9. Composite Block 11 uses the input from Organization Block 8 to determine which specific audiovisual elements will be needed at any given time, and to determine the organization and relationship of those elements. Composite Block 11 accepts decompressed audiovisual objects from AV Block 9, and organizes those objects as specified by information from Organization Block 8. Composite Block 11 then passes the organized information to Rendering Device 12, which might be a television screen, stereo speakers, etc.

Control Block 13 stores control messages which may be received through Control Stream 6 and/or may be watermarked into or steganographically encoded in other streams, including Audio Stream 4 and Video Stream 5. One control message format is illustrated by FIG. 5, which shows Control Message 501. Control Message 501 is made up of Header 502 and Message 503. Header 502 consists of Field 508, which includes a bit pattern identifying the following information as a header; Stream Type Field 509, which identifies this as a header for the organization stream; ID Field 504, which identifies this particular control message; Pointer Field 505, which identifies those ESs which are controlled by this message; Time Stamp Field 507, which identifies the particular portion of the stream which is controlled by this control message (this may indicate that the entirety of the stream is controlled); and Length Field 506, which specifies the length (in bytes) of Message 503. Message 503 may include packets following Header 502, using the general format shown in FIG. 3. In the example shown, Control Message 501 carries the unique ID 111000,

encoded in ID Field 504. This control message controls ESs 14 and 95, as indicated by Pointer Field 505. The associated Message contains 1,024 bytes, as indicated by Length Field 506.

In an alternate embodiment, the association of control to content may be made in Organization Block 8, which may store a pointer to particular control messages along with the metadata associated with streams, objects, etc. This may be disadvantageous, however, in that it may be desirable to protect this association from discovery or tampering by users. Since Control Block 13 will generally have to be protected in any event, storing the association in this block may make protection of Organization Block 8 less necessary.

Control Block 13 implements control over System 1 through Control Lines 14, 15 and 16, which control aspects of Organization Block 8, AV Block 9 and Composite Block 11, respectively. Each of these Control Lines may allow two-way communication.

Control Lines 14 and 15 are shown as communicating with AV Block Stream Flow Controller 18 and with Organization Block Stream Flow Controller 17. These Stream Flow Controllers contain functionality controlled by Control Block 13. In the embodiment illustrated, the Stream Flow Controllers are shown as the first stage in a two-stage pipeline, with information being processed by the Stream Flow Controller and then passed on to the associated functional block. This allows isolation of the control functionality from the content manipulation and display functionality of the system, and allows control to be added in without altering the underlying functionality of the blocks. In an alternate embodiment, the Stream Flow Controllers might be integrated directly into the associated functional blocks.

Stream Flow Controllers 17 and 18 contain Cryptographic Engines 19 and 20, respectively. These Cryptographic Engines operate under control of Control Block 13 to decrypt and/or cryptographically validate (e.g., perform secure hashing, message authentication code, and/or digital signature functions) the encrypted packet streams received from Demux 7. Decryption and validation may be selective or optional according to the protection requirements for the stream.

Cryptographic Engines 19 and 20 may be relatively complex, and may, for example, include a validation calculator that performs cryptographic hashing, message authentication code calculation, and/or other cryptographic validation processes. In addition, as is described further below, additional types of governance-related processing may also be used. In one alternative embodiment, a single Stream Flow Controller may be used for both Organization Stream 3 and Audio/Video Streams 4-5. This may reduce the cost of and space used by System 1. These reductions may be significant, since System 1

may contain multiple AV Blocks, each handling a separate Audio or Video Stream in parallel. This alternative may, however, impose a latency overhead which may be unacceptable in a real-time system.

If the Stream Flow Controllers are concentrated in a single block, they may be incorporated directly into Demux 7, which may handle governance processing prior to routing streams to the functional blocks. Such an embodiment would allow for governed decryption or validation of the entirety of Bit Stream 2, which could occur prior to the routing of streams to individual functional blocks. Encryption of the entirety of Bit Stream 2 (as opposed to individual encryption of individual ESs) might be difficult or impossible without incorporating stream controller functionality into Demux 7, since Demux 7 might otherwise have no ability to detect or read the header information necessary to route streams to functional blocks (that header information presumably being encrypted).

As is noted above, each of the individual streams contained in Bit Stream 2 may be individually encrypted. An encrypted stream may be identified by a particular indicator in the header of the stream, shown in FIG. 2 as Governance Indicator 205.

When a header is passed by Demux 7 to the appropriate functional block, the stream flow controller associated with that block reads the header and determines whether the following packets are encrypted or otherwise subject to governance. If the header indicates that no governance is used, the stream flow controller passes the header and the packets through to the functional blocks with no alteration. Governance Indicator 205 may be designed so that conventionally encoded content (e.g., unprotected MPEG-4 content) is recognized as having no Governance Indicator and therefore passed through for normal processing.

If a stream flow controller detects a set governance indicator, it passes the ES_ID associated with that stream and the time stamp associated with the current packets to Control Block 13 along Control Line 14 or 15. Control Block 13 then uses the ES_ID and time stamp information to identify which control message(s) are associated with that ES. Associated messages are then invoked and possibly processed, as may be used for governance purposes.

A simple governance case is illustrated by FIG. 6, which shows steps which take place using the functional blocks of FIG. 1. In Step 601, Demux 7 encounters a header, and determines that the header is part of the AV stream. In Step 602, Demux 7 passes the header to AV Stream Controller 18. In Step 603, AV Stream Controller 18 reads the header and determines that the governance indicator is set, thereby triggering further processing along Path 604. In Step 605, AV Stream Controller 18 obtains the ES_ID and

time stamp from the header and transmits these to Control Block 13, along Control Line 15. In Step 606, Control Block 13 looks up the ES_ID and determines that the ES_ID is associated with a particular control message. In Step 611, Control Block 13 uses the time stamp information to choose among control messages, if there is more than one control message associated with a particular ES. In Step 607, Control Block 13 accesses the appropriate control message, and obtains a cryptographic key or keys for decryption and/or validation. In Step 608, Control Block 13 passes the cryptographic key(s) along Control Line 15 to AV Stream Controller 18. In Step 609, AV Stream Controller 18 uses the cryptographic key as an input to Cryptographic Engine 20, which decrypts and/or validates the packets following the header as those packets are received from Demux 7. In Step 610, the decrypted packets are then passed to AV Block 9, which decompresses and processes them in a conventional manner.

Time stamp information may be useful when it is desirable to change the control message applicable to a particular ES. For example, it may be useful to encode different portions of a stream with different keys, so that an attacker breaking one key (or even a number of keys) will not be able to use the content. This can be done by associating a number of control messages with the same stream, with each control message being valid for a particular period. The time stamp information would then be used to choose which control message (and key) to use at a particular time. Alternatively, one control message may be used, but with updated information being passed in through the Control Stream, the updates consisting of a new time stamp and a new key.

In an alternative embodiment, Control Block 13 may proactively send the appropriate keys to the appropriate stream flow controller by using time stamp information to determine when a key will be will be needed. This may reduce overall latency.

Control Line 16 from FIG. 1 comes into play once information has been passed from Organization Block 8 and AV Block 9 to Composite Block 11, and the finished work is prepared for rendering through Rendering Device 12. When Composite Block 11 sends an object to Rendering Device 11, Composite Block 11 sends a start message to Control Block 13. This message identifies the object (including any associated ES_IDs), and specifies the start time of the display (or other rendering) of that object. When an object is no longer being rendered, Composite Block 11 sends an end message to Control Block 13, specifying that rendering of the object has ended, and the time at which the ending occurred. Multiple copies of a particular object may be rendered at the same time. For this reason, start and stop messages sent by Composite Block 11 may include an assigned instance ID, which specifies which instance of an object is being rendered.

Control Block 13 may store information relating to start and stop times of particular objects, and/or may pass this information to external devices (e.g., External Server 30) through Port 21. This information allows Control Block 13 to keep track not only of which objects have been decrypted, but of which objects have actually been used. This may be used, since System 1 may decrypt, validate, and/or decompress many more objects than are actually used. Control Block 13 can also determine the length of use of objects, and can determine which objects have been used together. Information of this type may be used for sophisticated billing and auditing systems, which are described further below.

Control Line 16 may also be used to control the operation of Composite Block 11. In particular, Control Block 13 may store information specifying when rendering of a particular object is valid, and may keep track of the number of times an object has been rendered. If Control Block 13 determines that an object is being rendered illegally (i.e., in violation of rules controlling rendering), Control Block 13 may terminate operation of Composite Block 11, or may force erasure of the illegal object.

In an alternate embodiment, the level of control provided by Control Line 16 may at least in part be provided without requiring the presence of that line. Instead, Control Block 13 may store a hash of the organization information currently valid for Organization Block 8. This hash may be received through Control Stream 6, or, alternatively, may be generated by Control Block 13 based on the information contained in Organization Block 8.

Control Block 13 may periodically create a hash of the information currently resident in Organization Block 8, and compare that to the stored hash. A difference may indicate that an unauthorized alteration has been made to the information in Organization Block 8, thereby potentially allowing a user to render information in a manner violative of the rules associated with that information. In such an event, Control Block 13 may take appropriate action, including deleting the information currently resident in Organization Block 8.

If System 1 is designed so that Organization Block 8 controls the use of content by Composite Block 11, so that content cannot be rendered except as is specified by the organization information, Control Block 13 may be able to control rendering of information through verifying that the current Organization Block contents match the hash which has been received by Control Block 13, thereby eliminating at least one reason for the presence of Control Line 16.

Control Block 13 may also be responsible for securely validating the origin, integrity, authenticity, or other properties of received content, through cryptographic

validation means such as secure hashing, message authentication codes, and/or digital signatures.

System 1 may also include an Inter-Rights Point, indicated as IRP 22. IRP 22 is a protected processing environment (e.g., a PPE) in which rules/controls may be processed, and which may store sensitive information, such as cryptographic keys. IRP 22 may be incorporated within Control Block 13, or may be a separate module. As is illustrated, IRP 22 may include CPU 23 (which can be any type of processing unit), Cryptographic Engine 24, Random Number Generator 25, Real Time Clock 26, and Secure Memory 27. In particular embodiments, some of these elements may be omitted, and additional functionality may be included.

Governance Rules

Control messages stored by Control Block 13 may be very complex. FIG. 7 illustrates the form in which the control messages may be stored in Control Block 13, consisting of Array 717. Column 701 consists of the address at which the control messages are stored. Column 702 consists of the identifier for each control message. This function may be combined with that of Column 701, by using the location information of Column 701 as the identifier, or by storing the message in a location which corresponds to the identifier. Column 703 consists of the ES_IDs for each stream controlled by the control message. Column 704 consists of the message itself. Thus, the control message stored at location 1 has the ID 15, and controls stream 903.

In a simple case, the message may include a cryptographic key, used to decrypt the content associated with the stream(s) controlled by the message. This is illustrated by Cryptographic Key 705 from FIG. 7. Cryptographic keys and/or validation values may also be included to permit cryptographic validation of the integrity or origin of the stream.

In a more complex case, the message may include one or more rules designed to govern access to or use of governed content. Rules may fall into a number of categories.

Rules may require that a particular aspect of System 1, or a user of System 1, be verified prior to decryption or use of the governed content. For example, System 1 may include System ID 28, which stores a unique identifier for the system. A particular rule contained in a control message may specify that a particular stream can only be decrypted on a system in which System ID 28 contains a particular value. This is illustrated at row 2 in FIG. 7, in which the message is shown as consisting of a rule and commands. The rule may be implicit, and therefore may not be stored explicitly in the table (e.g. the table may store only the rule, the rule - specific functions (commands) invoked by the rule, or only the functions).

In this case, when Stream Controller 18 encounters a Header for stream 2031 containing a set governance indicator, Stream Controller 18 passes the associated ES_ID (2031) to Control Block 13. Control Block 13 then uses the ES_ID to identify Control Message 20 which governs stream 2031. Control Message 20 includes Rule 706, which includes (or invokes) Commands 707, and an Authorized System ID 708. Authorized System ID 708 may have been received by System 1, either as part of Control Message 20, or as part of another control message (e.g., Control Message 9), which Control Message 20 could then reference in order to obtain access to the Authorized System ID. Such a case might exist, for example, if a cable subscriber had pre-registered for a premium show. The cable system might recognize that registration, and authorize the user to view the show, by sending to the user an ID corresponding to the System ID.

When Rule 706 is invoked, corresponding Commands 707 access System ID 28 and obtain the system ID number. The commands then compare that number to Authorized System ID 708, specified by Rule 706. If the two numbers match, Commands 707 release Cryptographic Key 709 to Stream Controller 18, which uses Cryptographic Key 709 to decrypt the stream corresponding to ES_ID 2031. If the two numbers do not match, Commands 707 fail to release Cryptographic Key 709, so that Stream Controller 18 is unable to decrypt the stream.

In order to carry out these functions, in one embodiment, Control Block 13 includes, or has access to, a processing unit and memory. The processing unit is preferably capable of executing any of the commands which may be included or invoked by any of the rules. The memory will store the rules and association information (ID of the control message and IDs of any governed ESs).

Since the functions being carried out by Control Block 13 are sensitive, and involve governance of content which may be valuable, Control Block 13 may be partially or completely protected by a barrier which resists tampering and observation. As is described above, the processing unit, secure memory, and various other governance-related elements may be contained in IRP 22, which may be included in or separate from Control Block 13.

Control Block 13 may also carry out somewhat more complex operations. In one example, a control message may require that information from System 1 not only be accessed and compared to expected information, but stored for future use. For example, a control message might allow decryption of a Stream, but only after System ID 28 has been downloaded to and stored in Control Block 13. This would allow a control message to check the stored System ID against System ID 28 on a regular basis, or perhaps on every

attempted re-viewing of a particular Stream, thereby allowing the control message to insure that the Stream is only played on a single System.

Control Block 13 may also obtain information dynamically. For example, System 1 may include User Interface 29, which can include any type of user input functionality (e.g., hardware buttons, information displayed on a video screen, etc.) A particular rule from a control message may require that the user enter information prior to allowing decryption or use of a stream. That information may, for example, be a password, which the Rule can then check against a stored password to insure that the particular user is authorized to render the stream.

Information obtained from the user might be more complicated. For example, a rule might require that the user input payment or personal information prior to allowing release of a cryptographic key. Payment information could, for example, constitute a credit card or debit card number. Personal information could include the user's name, age, address, email address, phone number, etc. Entered information could then be sent through Port 21 to External Server 30 for verification. Following receipt of a verification message from External Server 30, the Rule could then authorize release of a cryptographic key. Alternatively, Control Block 13 may be designed to operate in an "off-line" mode, storing the information pending later hookup to an external device (or network). In such a case, Control Block 13 might require that a connection be made at periodic intervals, or might limit the number of authorizations which may be obtained pending the establishment of an external connection.

In a somewhat more complex scenario, a control message may include conditional rules. One particular example is illustrated by row 4 of the table shown in FIG. 7, in which Control Message 700 is shown as controlling streams 49-53. Control Message 700 further consists of Rule 710, Commands 711 and Cryptographic Keys 712-716. There could, of course, be a number of additional cryptographic keys stored with the message.

In this case, Rule 710 specifies that a user who agrees to pay a certain amount (or provide a certain amount of information) may view Stream 49, but all other users are required to view Stream 50, or a combination of Streams 49 and 50. In this case, Stream 49 may represent a movie or television program, while Stream 50 represents advertisements. In one embodiment, different portions of Stream 49 may be decrypted with different keys so that, for example, a first portion is decrypted with Key 712, a second portion is decrypted with Key 713, a third portion is decrypted with Key 714, and so on. Rule 710 may include all keys used to decrypt the entirety of Stream 49. When the user initially attempts to access the video encoded in Stream 49, Rule 710 could put up a

message asking if the user would prefer to use pay for view mode or advertising mode. If the user selects pay for view mode, Rule 710 could store (or transmit) the payment information, and pass Cryptographic Key 712 to Stream Controller 18. Stream Controller 18 could use Cryptographic Key 712 to decrypt the first stream until receipt of a header indicating that a different key is needed to decrypt the following set of packets. Upon request by Stream Controller 18, Control Block 13 would then check to determine that payment had been made, and then release Cryptographic Key 713, which would be used to decrypt the following packets, and so on. Rule 710 could additionally release Cryptographic Key 716, corresponding to Organization Stream 52, which corresponds to video without advertisements.

If, on the other hand, the user had chosen the advertising mode, Rule 710 could release Cryptographic Key 712 to Stream Controller 18 to allow decryption of Stream 49. Rule 710 could also authorize decryption of Stream 50 which contains the advertisements. Rule 710 could further release Cryptographic Key 715 to Organization Block 8. Cryptographic Key 715 matches Organization Stream 51. Organization Stream 51 references the video from Stream 49, but also references advertisements from Stream 50. Rule 710 would refuse to release Cryptographic Key 716, which corresponds to Organization Stream 52, which corresponds to the video without advertisements.

In operation, Control Block 13 could monitor information from Composite Block 11 over Control Line 16. That information could include the identity of each object actually rendered, as well as a start and stop time for the rendering. Control Block 13 could use this information to determine that an advertisement had actually been rendered, prior to releasing Cryptographic Key 713 for decryption of the second portion of video from Stream 49. This feedback loop allows Control Block 13 to be certain that the advertisements are not only being decrypted, but are also being displayed. This may be necessary because Composite Block 11 may be relatively unprotected, thereby allowing an unscrupulous user to remove advertisements before viewing.

A variety of additional relatively complex scenarios are possible. For example, rules from Control Block 13 could customize the programming for a particular geographic location or a particular type of viewer, by using information on the location or the viewer to control conditional decryption or use. This information could be stored in System 1 or entered by the user.

In another example, shown at row 5 of Array 717, Rule 719 may specify Budget 718, which may include information relating to the number of uses available to the user, the amount of money the user has to spend, etc. In operation, Rule 719 may require that

Budget 718 be securely stored and decremented each time a budgeted activity occurs (e.g., each time the associated work is played). Once the budget reaches zero, Rule 719 may specify that the work may no longer be played, or may display a message to the user indicating that the user may obtain additional budget by, for example, entering a credit card number or password, or contacting an external server.

In another example, a rule may control the ability of a user to copy a work to another device. The rule may, for example, specify that the user is authorized to use the governed work on more than one device, but with only one use being valid at any time. The rule may specify that an indication be securely stored regarding whether the user has "checked out" the work. If the user copies the work to another device (e.g., through Port 21), the rule may require that the work only be transmitted in encrypted form, and that the relevant control messages be transmitted along with it. The rule can further require that an indicator be securely set, and that the indicator be checked each time the user attempts to use or copy the work. If the indicator is set, the rule might require that the work not be decrypted or used, since the user only has the right to use the work on one device at a time, and the indicator establishes that the work is currently "checked out" to another device and has not been checked back in.

The receiving device may include the same type of indicator, and may allow the user to use the work only as long as the indicator is not set. If the user desires to use the work on the original device, the two devices may communicate, with the indicator being set in the second and reset in the first. This allows the work to be stored in two locations, but only used in one.

In another embodiment, the same result may be reached by copying the relevant control message from one device to the other, then erasing it from the original device. Because the control message includes keys used for decryption, this would insure that the work could only be used in one device at a time.

In one embodiment, this technique may be used to communicate digital media files (e.g., music, video, etc.) from a personal computer to a consumer electronics device without allowing the user to make multiple choices for simultaneous use. Thus, a larger, more sophisticated device (e.g., a personal computer), could download a file, then "check out" the file to a portable device lacking certain functions present in the personal computer (e.g., a hand-held music player).

Rules may also be used to specify that an initial user may transfer the file to another user, but only by giving up control over the file. Such rules could operate similarly to the

technique described above for transferring a file from one device to another, or could require that the original file be entirely erased from the original device after the transfer.

Rules in Control Block 13 may be added or updated through at least two channels. New rules may be obtained through Control Stream 6. If a control message contains an identifier corresponding to a control message already present in Control Block 13, that control message (including contained rules) may overwrite the original control message. A new rule may, for example, be identical to an existing rule, but with a new time stamp and new keys, thereby allowing decryption of a stream which had been encrypted with multiple keys. System 1 may be designed so that certain rules may not be overwritable. This may be enforced by designating certain positions in Array 717 as non-overwritable, or by providing a flag or other indicator to show that a particular rule cannot be overwritten or altered. This would allow for certain types of superdistribution models, including allowing a downstream distributor to add rules without allowing the downstream distributor to remove or alter the rules added by upstream distributors.

In addition, new rules may be encoded into Organization Stream 3, Audio Stream 4, or Video Stream 5, in the form of a watermark or steganographic encoding.

New rules may also be obtained through Port 21. Port 21 may connect to an external device (e.g., a smart card, portable memory, etc.) or may connect to an external network (e.g., External Server 30). Rules may be obtained through Port 21 either in an ad hoc manner, or as a result of requests sent by Control Block 13. For example, Control Message 14 (FIG. 7, row 6) may include a rule specifying that a new rule be downloaded from a particular URL, and used to govern Stream 1201.

Control messages, including rules, may be encoded using secure transmission formats such as DigiBoxes. A DigiBox is a secure container means for delivering a set of business rules, content description information, content decryption information and/or content validation information. One or more DigiBoxes can be placed into the headers of the media content or into data streams within the media.

FIG. 12 illustrates one embodiment of the DigiBox format and the manner in which that format is incorporated into a control message. Control Message 1201 is made up of Control Message Header 1202 and Control Message Contents 1203. As is described elsewhere, Control Message Header 1202 may include information used by Demux 7 (FIG. 1) to appropriately route the message to Control Block 13.

Control Message Contents 1203 of Control Message 1201 consists of DigiBox 1204, and may also include additional information. DigiBox 1204 consists of DigiBox Header 1205, Rules 1206 and Data 1207. Rules 1206 may include one or more rules. Data

1207 may include various types of data, including ES_ID 1208, Cryptographic Key 1209, and Validation Data 1210. Data 1207 may also include cryptographic information such as a specification of the encryption algorithm, chaining modes used with the algorithm, keys and initialization vectors used by the decryption and chaining.

Initialization vectors contained within Data 1207 are similar to cryptographic keys, in that they constitute input to the original encryption process and therefore are necessary for decryption. In one well-known prior art embodiment, the initialization vectors may be generated by starting with a base initialization vector (a 64 bit random number) and xor'ing in the frame number or start time for the content item.

Validation Data 1210 contained within Data 1207 may include cryptographic has or authentication values, cryptographic keys for calculating keyed authentication values (e.g., message authentication codes), digital signatures, and/or public key certificates used in validating digital certificates.

Thus, the DigiBox may incorporate the information described above as part of the control message, including the rules, the stream ID and the cryptographic keys and values.

In an alternative embodiment, DigiBox Header 1205 may be designed so that it can be read by Demux 7 and routed to Control Block 13. In such an embodiment, DigiBox 1204 would itself constitute the entirety of the control message, thus obviating the need to nest DigiBox 1204 within Control Message 1201.

Some or all of the contents of DigiBox 1204 will generally be encrypted. This may include Rules 1206, Data 1207, and possibly some or all of Header 1205. System 1 may be designed so that a DigiBox may only be decrypted (opened) in a protected environment such as IRP 22. In an alternate embodiment, Control Block 13 may directly incorporate the functionality of IRP 22, so that the DigiBox may be opened in Control Block 13 without the necessity of routing the DigiBox to IRP 22 for processing. In one embodiment, the cryptographic key used to decrypt DigiBox 1204 may be stored in IRP 22 (or Control Block 13), so that the DigiBox can only be opened in that protected environment.

Rules 1206 are rules governing access to or use of DigiBox Data 1207. In one embodiment, these rules do not directly control the governed streams. Since Cryptographic Key 1209 can only be accessed and used through compliance with Rules 1206, however, Rules 1206 in fact indirectly control the governed streams, since those streams can only be decrypted through use of the key, which can only be obtained in compliance with the rules. In another embodiment, Data 1207 may include additional rules, which may be extracted from the DigiBox and stored in a table such as Array 717 of FIG. 7.

The rules governing access to or use of a DigiBox may accompany the DigiBox, (as shown in FIG. 12) or may be separately transmitted, in which event Rules 1206 would contain a pointer or reference to the rules used to access Data 1207. Upon receipt of a DigiBox, Control Block 13 may receive rules separately through Control Stream 6, or may request and receive rules through Port 21.

Pipelined Implementation

One potential drawback to the system illustrated in FIG.1 consists of the fact that the system introduces complexity and feedback into a pipelined system designed to render content in real time. The rendering pipeline generally consists of Demux 7, Organization Block 8 and AV Block 9, Composite Block 11 and Rendering Device 12. Because content is received in a streamed fashion, and must be rendered in real time, pipelined processing must occur in a highly efficient manner, under tight time constraints. A failure to process within the time available may mean that output to Rendering Device 12 may be interrupted, or that incoming Bit Stream 2 may overflow available buffers, thereby causing the loss of some portion of the incoming data.

An alternative embodiment of System 1 is designed to address these problems, although at a possible cost in the ability to use standard system components and a possible cost in overall system security. This alternative embodiment is illustrated in FIG. 11, which shows System 1101.

System 1101 is similar to System 1 from FIG. 1 in many respects. It receives Bit Stream 1102, which consists of Organization Stream 1103, Audio Stream 1104, Video Stream 1105 and Control Stream 1106. These streams are received by Demux 1107, which passes Organization Stream 1103 to Organization Block and passes Audio Stream 1104 and Video Stream 1105 to AV Block 1109. Organization Block 1108 and AV Block 1109 operate similarly to their counterparts in FIG. 1, and pass information to Composite Block 1110, which organizes the information into a coherent whole and passes it to Rendering Device 1111. Streams sent to Organization Block 1108 are decrypted and/or validated by Stream Flow Controller 1112, and streams sent to AV Block 1109 are decrypted and/or validated by Stream Flow Controller 1113.

System 1101 differs from System 1, however, in that control and feedback are distributed, and integrated directly into the processing and rendering pipeline. System 1101 thus lacks a separate control block, and also lacks a feedback path back from the Composite Block 1110.

In System 1101, control is exercised directly at Organization Block 1108 and AV Block 1109. As in System 1, cryptographic keys are received through Control Stream 1106

(in an alternative embodiment, the keys could be incorporated directly into header or other information in Organization Stream 1103 or Audio/Video Streams 1104 and 1105). Those keys are included in a data format which includes information regarding the stream type of the encrypted content and, if multiple stream types are possible, an identifier for the particular controlled stream.

When Demux 1107 encounters a key in Control Stream 1106, it reads the information relating to the stream type, and routes the key to the appropriate stream flow controller. If Demux 1107 encounters a key designated for decryption or validation of Organization Stream 1103, for example, it routes that key to Stream Flow Controller 1112.

Stream Flow Controller 1112 stores received keys in Storage Location 1114. Storage Location 1114 stores the keys and also stores an indicator of the controlled stream ID.

Stream Flow Controller 1112 includes Cryptographic Engine 1115, which uses the received keys to decrypt and/or validate encrypted and/or protected portions of Organization Stream 1103. The keys may themselves be received in an encrypted manner, in order to provide some degree of security. In such a case, Stream Flow Controller may use a variety of techniques to decrypt the key, including using stored information as a key, or as a key seed. That stored information could, for example, constitute a "meta-key" provided earlier through Bit Stream 1102 or through a separate port.

Stream Flow Controller 1113, associated with AV Block 1109, contains a corresponding Storage Location 1116 and Cryptographic Engine 1117, and operates in a manner similar to the operation described for Stream Flow Controller 1112.

This implementation avoids the latency penalty which may be inherent in the necessity for communication between stream flow controllers and a separate control block.

This alternate implementation may also eliminate the feedback channel from the composite block (FIG.1, Control Line 16). This feedback channel may be used in order to insure that the content being passed from Composite Block 11 to Rendering Device 12 is content that has been authorized for rendering. In the alternate embodiment shown in FIG.11, this feedback channel does not exist. Instead, this implementation relies on the fact that Composite Block 1110 depends upon information from Organization Block 1108 to determine the exact structure of the information being sent to Rendering Device 1111. Composite Block 1110 cannot composite information in a manner contrary to the organization dictated by Organization Block 1108.

In one embodiment, this control by Organization Block 1108 may be sufficient to obviate the need for any feedback, since Organization Block 1108 may be designed so that

it accepts information only through Stream Controller 1112, and Stream Controller 1112 may be designed so that it only decrypts or validates information under the control of rules stored in Storage Location 1114.

In such an embodiment, security may be further increased by incorporating Secure Memory 1118 into Organization Block 1108. Secure Memory 1118 may store a copy or hash of the organization tree validly decrypted by Stream Controller 1112, and in current use in Main Organization Block Memory 1119. Organization Block 1108 may be used to periodically compare the organization tree stored in Main Organization Block Memory 1119 to the tree stored in Secure Memory 1118. If a discrepancy is spotted, this may indicate that an attacker has altered the organization tree stored in Main Organization Block 1119, thereby possibly allowing for the rendering of content in violation of applicable rules. Under such circumstances, Organization Block 1108 may be used to take protective measures, including replacing the contents of Main Organization Block Memory 1119 with the contents of Secure Memory 1118.

MPEG-4 Implementation

The generic system described above may be embodied in an MPEG-4 system, as illustrated in FIG. 8, which shows MPEG-4 System 801.

MPEG-4 System 801 accepts MPEG-4 Bit Stream 802 as input. MPEG-4 Bit Stream 802 includes BIFS Stream 803, OD Stream 804, Audio Stream 805, Video Stream 806 and IPMP Stream 807. These streams are passed to Demux 808, which examines header information and routes packets as appropriate, to BIFS 809, AVO 810, OD 811 or IPMP System 812.

IPMP System 812 receives IPMP messages through IPMP Stream 807. Those messages may include header information identifying the particular message, as well as an associated IPMP message. The IPMP message may include control information, which may include a cryptographic key, validation information, and/or may include complex governance rules, as are described above.

Stream Controllers 813, 814 and 815 act to decrypt, validate, and/or govern streams passed to BIFS 809, AVO 810 and OD 811, respectively.

OD 811 holds object descriptors, which contain metadata describing particular objects. This metadata includes an identifier of the particular Elementary Stream or streams which include the object, and may also include a pointer to a particular IPMP message which governs the object. Alternatively, the relationship between IPMP messages and particular objects or streams may be stored in a table or other form within IPMP System 812.

IPMP System 812 may exercise control over other functional blocks through Control Lines 816, 817, 818 and 819, each of which may transmit control/governance signals from IPMP System 812 and information or requests from other functional blocks to IPMP System 812. The information requests may include an ES_ID and a time stamp, which IPMP System 812 may use to determine which particular message (e.g., key) should be used and when.

In an alternative embodiment, IPMP System 812 may exercise control over Composite and Render 821 by receiving a hash of the currently valid BIFS tree (possibly through IPMP stream 807), and periodically checking the hash against the BIFS tree stored in BIFS 809. Because BIFS 809 controls the manner in which Composite and Render 821 renders information, if IPMP System 812 confirms that the current BIFS tree is the same as the authorized tree received through BIFS Stream 803, IPMP System 812 can confirm that the proper content is being rendered, even without receiving feedback directly from Composite and Render 821. This may be necessary, since BIFS 809 may communicate with Port 822, which may allow a user to insert information into BIFS 809, thereby creating a possibility that a user could insert an unauthorized BIFS tree and thereby gain unauthorized access to content.

When a stream controller receives encrypted or otherwise governed information, it may send the ES_ID and time stamp directly to IPMP System 812. Alternatively, it may send this information to OD 811, which may reply with the ID of the IPMP message which governs that object or stream. The stream controller can then use that IPMP message ID to request decryption, validation, and/or governance from IPMP System 812. Alternatively, OD 811 can pass the IPMP ID to IPMP System 812, which can initiate contact with the appropriate stream controller.

IPMP System 812 may obtain IPMP information through two channels other than IPMP Stream 807. The first of these channels is Port 820, which may be directly connected to a device or memory (e.g., a smart card, a DVD disk, etc.) or to an external network (e.g., the Internet). An IPMP message may contain a pointer to information obtainable through Port 812, such as a URL, address on a DVD disk, etc. That URL may contain specific controls needed by the IPMP message, or may contain ancillary required information, such as, for example, information relating to the budget of a particular user.

IPMP System 812 may also obtain IPMP information through OD updates contained in OD Stream 804. OD Stream 804 contains metadata identifying particular objects. A particular OD Message may take the format shown in FIG. 9. In this figure, OD Message 901 includes Header 902, which identifies the following packets as part of the OD

stream, and indicates the number of packets. OD Message 901 further consists of Message 903, which includes a series of Pointers 904 and associated Metadata 905. Each Pointer 904 identifies a particular Elementary Stream, and the associated metadata is applicable to that stream. Finally, OD Message 901 may contain an IPMP Pointer 906, which identifies a particular IPMP message.

In aggregate, the information contained in OD Message 901 constitutes an object descriptor, since it identifies and describes each elementary stream which makes up the object, and identifies the IPMP message which governs the object. OD Message 901 may be stored in OD 811, along with other messages, each constituting an object descriptor.

Object descriptors stored in OD 811 may be updated through OD Stream 804, which may pass through a new object descriptor corresponding to the same object. The new object descriptor then overwrites the existing object descriptor. This mechanism may be used to change the IPMP message which controls a particular object, by using a new object descriptor which is identical to the existing object descriptor, with the exception of the IPMP pointer.

OD Stream 804 can also carry IPMP_DescriptorUpdate messages. Each such message may have the same format as IPMP messages carried on the IPMP stream, including an IPMP ID and an IPMP message.

IPMP_DescriptorUpdate messages may be stored in a table or array in OD 811, or may be passed to IPMP System 812, where they may overwrite existing stored IPMP messages, or may add to the stored messages.

Since IPMP information may be separately conveyed through the OD stream or the IPMP stream, MPEG-4 System 801 may be designed so that it only accepts information through one or the other of these channels.

In another embodiment, the existence of the two channels may be used to allow multi-stage distribution, with governance added at later stages, but with no risk that later alterations may override governance added at an earlier stage.

Such a system is illustrated in FIG. 10. In this Figure, IPMP System 812 includes IPMP Table 1002, which has slots for 256 IPMP messages. This table stores the IPMP_ID implicitly, as the location at which the information is stored, shown in Column 1003. The IPMP message associated with IPMP_ID 4, for example, is stored at slot 4 of IPMP Table 1002.

Each location in IPMP Table 1002 includes Valid Indicator 1004 and Source Indicator 1005. Valid Indicator 1004 is set for a particular location when an IPMP message is stored at that location. This allows IPMP System 812 to identify slots which are

unfilled, which otherwise might be difficult, since at start-up the slots may be filled with random information. This also allows IPMP System 812 to identify messages which are no longer valid and which may be replaced. Valid Indicator 1004 may store time stamp information for the period during which the message is valid with IPMP System 812 determining validity by checking the stored time stamp information against the currently valid time.

Source Indicator 1005 is set based on whether the associated IPMP message was received from IPMP Stream 807 or from OD Stream 804.

These indicators allow IPMP System 812 to establish a hierarchy of messages, and to control the manner in which messages are added and updated. IPMP System 812 may be designed to evaluate the indicators for a particular location once a message is received corresponding to that location. If the valid indicator is set to invalid, IPMP System 812 may be designed to automatically write the IPMP message into that slot. If the valid indicator is set to valid, IPMP System 812 may then be designed to check the source indicator. If the source indicator indicates that the associated message was received through OD Stream 804, IPMP System 1812 may be designed to overwrite the existing message with the new message. If, however, the source indicator indicates that the associated message was received through IPMP Stream 807, IPMP System 812 may be designed to check the source of the new message. That check may be accomplished by examining the header associated with the new message, to determine if the new message was part of OD Stream 804 or part of IPMP Stream 807. Alternatively, IPMP System 812 may derive this information by determining whether the message was received directly from Demux 808 or through OD 811.

If the new message came through IPMP Stream 807, IPMP System 812 may be designed to store the new message in Table 1002, overwriting the existing message. If the new message came through OD Stream 804, on the other hand, IPMP System 812 may be designed to reject the new message.

This message hierarchy can be used to allow for a hierarchy of control. A studio, for example, may encode a movie in MPEG-4 format. The studio may store IPMP messages in the IPMP stream. Those messages may include a requirement that IPMP System 812 require that a trailer for another movie from the same studio be displayed prior to the display of the feature movie. IPMP System 812 could be used to monitor the beginning and end of rendering of the trailer (using feedback through Control Line 819) to ensure that the entire trailer plays, and that the user does not fast-forward through it.

The movie studio could encrypt the various elementary streams, including the IPMP stream. The movie studio could then provide the movie to a distributor, such as a cable channel. The movie studio could provide the distributor with a key enabling the distributor to decrypt the OD stream (or could leave the OD stream unencrypted), and the ability to insert new messages in that stream. The cable channel could, for example, include a rule in the OD stream specifying that the IPMP system check to determine if a user has paid for premium viewing, decrypt the movie if premium viewing has been paid for, but insert advertisements (and require that they be rendered) if premium viewing has not been paid for).

The cable channel would therefore have the ability to add its own rules into the MPEG-4 Bit Stream, but with no risk that the cable channel would eliminate or alter the rules used by the movie studio (e.g., by changing the trailer from a movie being promoted by the studio to a rival movie being promoted by the cable channel). The studio's rules could specify the types of new rules which would be allowed through the OD stream, thereby providing the studio a high degree of control.

This same mechanism could be used to allow superdistribution of content, possibly from one user to another. A user could be provided with a programming interface enabling the insertion of messages into the OD stream. A user might, for example, insert a message requiring that a payment of $1.00 be made to the user's account before the movie can be viewed. The user could then provide the movie to another user (or distribute it through a medium whereby copying is uncontrolled, such as the Internet), and still receive payment. Because the user's rules could not overrule the studio's rules, however, the studio could be certain that its rules would be observed. Those might include rules specifying the types of rules a user would be allowed to add (e.g., limiting the price for redistribution).

MPEG-4 System 801 may also be designed to include a particular type of IPMP system, which may be incompatible with IPMP systems that may be designed into other MPEG-4 systems. This may be possible because the MPEG-4 standard does not specify the format of the information contained in the IPMP stream, thereby allowing different content providers to encode information in differing manners.

IPMP System 812 in MPEG-4 System 801 may be designed for an environment in which differing IPMP formats exist. That system may scan the IPMP stream for headers that are compatible with IPMP System 812. All other headers (and associated packets) may be discarded. Such a mechanism would allow content providers to incorporate the same IPMP message in multiple formats, without any concern that encountering an unfamiliar format would cause an IPMP system to fail. In particular, IPMP headers can

incorporate an IPMP System Type Identifier. Those identifiers could be assigned by a central authority, to avoid the possibility that two incompatible systems might choose the same identifier.

IPMP System 801 might be designed to be compatible with multiple formats. In such a case, IPMP System 801 might scan headers to locate the first header containing an IPMP System Identifier compatible with IPMP System 801. IPMP System 801 could then select only headers corresponding to that IPMP System Identifier, discarding all other headers, including headers incorporating alternate IPMP System Identifiers also recognized by the IPMP system.

Such a design would allow a content provider to provide multiple formats, and to order them from most to least preferred, by including the most preferred format first, the second most preferred format second, and so on. Since IPMP System 801 locks onto the first compatible format it finds, this ordering in IPMP Stream 801 would insure that the IPMP system chose the format most desired by the content provider.

Even if different IPMP formats are used, content will probably be encoded (and encrypted) using a single algorithm, since sending multiple versions of content would impose a significant bandwidth burden. Thus, ordinarily it will be necessary for content to be encrypted using a recognized and common encryption scheme. One such scheme could use the DES algorithm in output feedback mode.

This method of screening IPMP headers, and locking onto a particular format may also be used to customize an MPEG-4 bit Stream for the functional capabilities of a particular MPEG-4 system. Systems capable of rendering MPEG-4 content may span a considerable range of functionality, from high-end home theaters to handheld devices. Governance options suitable for one type of system may be irrelevant to other systems.

For example, MPEG-4 System 801 may include a connection to the Internet through Port 820, whereas a second MPEG-4 system (for example a handheld Walkman-like device) may lack such a connection. A content provider might want to provide an option to a viewer, allowing the viewer to see content for free in return for providing information about the viewer. The content provider could insert a rule asking the user whether the user wants to view the content at a cost, or enter identification information. The rule could then send the information through a port to the Internet, to a URL specified in the rule. A site at that URL could then evaluate the user information, and download advertisements targeted to the particular user.

Although this might be a valuable option for a content provider, it obviously makes no sense for a device which is not necessarily connected to the Internet. It would make no

sense to present this option to the user of a non-connected device, since even if that user entered the information, the rule would have no way to provide the information to an external URL or download the advertisements. In such a case, the content provider might prefer to require that the user watch preselected ads contained in the original MPEG-4 bit stream.

Header information in the IPMP stream could be used to customize an MPEG-4 bit stream for particular devices. As with the IPMP System Type information, IPMP Header information could include MPEG-4 System Types. These could include 8 or 16-bit values, with particular features represented by bit maps. Thus, the presence of a bit at position 2, for example, could indicate that a device includes a persistent connection to the Internet.

An IPMP system could then evaluate the headers, and lock on to the first header describing functionality less than or equal to the functionality contained in the MPEG-4 device in which the IPMP system is embedded. If the header constituted a complete match for the functionality of the MPEG-4 device, the IPMP system could then cease looking. If the header constitutes less than a complete match (e.g., a header for a system which has an Internet connection, but lacks a digital output port, when the system includes both), the IPMP system can lock on to that header, but continue to scan for closer matches, locking on to a closer match if and when one is found.

The IPMP messages identified by a particular header would be those suited for the particular functionality of the MPEG-4 device, and would allow for customization of the MPEG-4 bit stream for that functionality. In the context of the example given above, the IPMP system for an MPEG-4 device containing an Internet connection would lock on to a particular header, and would download the IPMP messages characterized by that header. Those messages would prompt the user for information, would provide that information to the URL, and would authorize decryption and rendering of the movie, with the advertisements inserted at the appropriate spot.

In the case of an MPEG-4 device without an Internet connection, on the other hand, the IPMP system would lock onto a set of headers lacking the bit indicating an Internet connection, and would download the rules associated with that header. Those rules might not provide any option to the user. The rules might allow decryption of the content, but would also specify decryption of an additional ES from the MPEG-4 stream. That additional ES would contain the advertisements, and the IPMP system would require decryption and rendering of the advertisements, checking Control Line 819 to make certain that this had occurred. In the case of the system with the Internet connection, however, the rules allowing decryption and requiring rendering of the ES containing the advertisements

would never be loaded, since those rules would be contained within messages identified by the wrong type of header. The advertisement ES would therefore never be decrypted and would be ignored by the MPEG-4 device.

FIG. 21 illustrates one manner in which a protected MPEG-4 file may be created. In this figure, CreateBox 2101 represents a DigiBox creation utility, which accepts keys and rules. In one embodiment, CreateBox 2101 may pass these keys and rules to IRP 2102 and receive DigiBox 2103 from IRP 2102. In another embodiment, IRP 2102 may be incorporated into CreateBox 2101, which accepts keys and rules and outputs DigiBox 2103.

DigiBox 2103 contains governance rules, initialization vectors and keys. DigiBox 2103 is passed from CreateBox 2101 to Bif Encoder 2104. Bif Encoder 2104 may be conventional, with the exception that it is designed to accept and process DigiBoxes such as DigiBox 2103. Bif Encoder 2104 also accepts a .txt file containing a scene graph, and initial object descriptor commands.

Bif Encoder 2104 outputs a .bif file, containing the scene graph stream (in compressed binary form) and a .od file, containing the initial object descriptor commands, the object descriptor stream, and DigiBox 2103.

Bif Encoder 2104 passes the .bif file and the .od file to Mux 2105. Mux 2105 also accepts compressed audio and video files, as well as a .scr file that contains the stream description. Mux 2105 creates IPMP streams, descriptors and messages, encrypts the content streams, interleaves the received streams, and outputs Protected MPEG-4 Content File 2106, consisting of Initial Object Descriptor 2107 and Encrypted Content 2108. Initial Object Descriptor 2107 contains DigiBox 2103, as well as other information. Encrypted Content 2108 may include a scene graph stream (i.e., a BIFS stream), an object descriptor stream, IPMP streams, and encrypted content streams.

If DigiBox 2103 contains all keys and rules necessary to render all of the content, it may be unnecessary for Mux 2105 to create any IPMP streams. If additional keys or rules may be necessary for at least a portion of the content, Mux 2105 may incorporate those rules and keys into one or more additional DigiBoxes, and incorporate those DigiBoxes either in the IPMP stream or in the OD update stream.

FIG. 22 illustrates one manner in which control may be incorporated into an existing MPEG-4 stream. In this figure, Unprotected MPEG-4 Content File 2201 includes Initial Object Descriptor 2202 and Content 2203. The content may include a scene description stream (or BIF stream), an object descriptor stream, a video stream, an audio stream, and possibly additional content streams.

Unprotected MPEG-4 Content File 2201 is passed to Repackager 2204, which also accepts keys and rules. Repackager 2204 passes the keys and rules to IRP 2205, and receives DigiBox 2206 in return, containing keys, rules and initialization vectors. In an alternate embodiment, IRP 2205 may be incorporated directly into Repackager 2204.

Repackager 2204 demuxes Unprotected MPEG-4 Content File 2201. It inserts DigiBox 2206 into the Initial Object Descriptor and encrypts the various content streams. Repackager 2204 also adds the IPMP stream, if this is necessary (including if additional DigiBoxes are necessary).

Repackager 2204 outputs Protected MPEG-4 Content File 2207, consisting of Initial Object Descriptor 2208 (including DigiBox 2206) and Encrypted Content 2209 (consisting of various streams, including the IPMP streams, if necessary).

Real Networks Implementation

In one embodiment, the elements described above may be used in connection with information encoded in compliance with formats established by Real Networks, Inc.

The Real Networks file format (RMFF) is illustrated in FIG. 13. This format includes a block of headers at the beginning (Header 1301), followed by a collection of content packets (Content 1302), followed by an index used for seek and goto operations (Index 1303). Each file can contain several streams of different types. For each stream, there is a "Media Properties Header" (1304) used to describe the format of the media content (e.g., compression format) and provide stream specific information (e.g., parameters for the decompressor).

Real Networks streams can be protected by inserting a DigiBox into Header 1301 and encrypting the data packets contained in Content 1302. The altered format is illustrated in FIG.14, which shows Header 1401, including Media Properties Headers 1402 and 1403, which in turn contain DigiBoxes 1404 and 1405, respectively. The format also includes encrypted Content 1406 and Index 1407.

In one embodiment, the declared type of the data is changed from the standard Real Networks format to a new type (e.g., RNWK_Protected.) The old type is then saved. Changing the type forces the Real Networks player to load a "Trust Plugin," since this Plugin is registered as the only decoder module that can process streams of type "RNWK-Protected." The Trust Plugin opens the DigiBox, gets approval from the user, if it is needed, determines the original content type, loads a decoder plugin for the original content, and then decrypts and/or validates the content, passing it to the content decoder plugin to be decompressed and presented to the user.

In one embodiment, the specific alterations made to the Real Networks file format are the following:

- Increase the preroll time to force larger buffers on playback. In a current embodiment, an increase of 3 seconds is used. Larger buffers are needed because of the extra steps needed to decrypt the content.
- Modify each stream-specific header by changing the mime type to "RNWK-Protected", saving the old mime type in the decoder specific information and adding a content identifier and DigiBox to the decoder specific information. The DigiBox contains the key, initialization vector (IV), version information, and watermarking instructions. The key, IV and content identifier are generated automatically, or can be provided as command-line parameters. The same key, IV and content identifier are used for every stream.
- Content packets are selectively encrypted. In one embodiment, content packets whose start time in milliseconds is in the first half-second of each 5 seconds (i.e., starttime % 5000 < 500) are encrypted. This encrypts approximately one-tenth of the content reducing encryption and decryption costs, and damages the content, sufficiently to prevent resale. The encryption algorithm can be DES using output-feedback mode or any similar algorithm. The initialization vector is computed for each packet by xoring the stream's IV with the packet's start time in milliseconds. Some information unique to the stream should also be xored into the IV. In one embodiment, the same IV is used for multiple packets whenever two or more streams have packets with the same start time. This usually happens for the first packet in each stream since they usually have start time 0. Other than the first packet, it is rare to have two packets have the same start time.

In one embodiment, these changes to the Real Networks file format are accomplished as is shown in FIG. 15. As is illustrated, RMFF file 1501 is formatted in the standard Real Networks RMFF format. This file is passed to Packager 1502. Also passed to Packager 1502 is Rights File 1503. Packager 1503 generates Protected RMFF File 1504, which includes various alterations as described above and as listed in FIG. 15, including the incorporation of one or more DigiBoxes in the header, encryption of the content, modification of the mime type, etc.

In one embodiment, the trust plugin described above is illustrated in FIGs. 16 and 17. FIG. 16 illustrates the standard Real Networks architecture. File 1601 (e.g., a streaming audio file in Real Networks format) is provided to Real Networks G2 Client

Core 1602. File 1601 may be provided to RealNetworks G2 Client Core 1602 from Server 1603, or through Direct Connection 1604.

Upon receipt of File 1601, Real Networks G2 Client Core 1602 accesses a rendering plugin appropriate to File 1601, based on information which is obtained from the header associated with File 1601. Rendering Plugins 1605 and 1606 are shown. If File 1601 is of a type which cannot be rendered by either Rendering Plugin 1605 or Rendering Plugin 1606, Real Networks G2 Client Core 1602 may attempt to access an appropriate plugin, e.g., by asking for the user's assistance or by accessing a site associated with the particular file type.

Rendering Plug-In 1605 or 1606 processes File 1601 in a conventional manner. This processing most likely includes decompression of File 1601, and may include other types of processing useful for rendering the content. Once this processing is complete (keeping in mind that the content is streamed, so that processing may be occurring on one set of packets at the same time that another set of packets is being rendered), File 1601 is passed back to Real Networks G2 Client Core 1602, which then passes the information to Rendering Device 1607. Rendering Device 1607 may, for example, be a set of stereo speakers, a television receiver, etc.

FIG. 17 illustrates the manner in which a trust plugin operates within the overall Real Networks architecture. Much of the architecture illustrated in FIG. 17 is the same as that illustrated in FIG. 16. Thus, File 1701 is provided to Real Networks G2 Client Core 1702 through Server 1703 or through Direct Connection 1704. The file is processed by Real Networks G2 Client Core 1702, using plugins, including Rendering Plugins 1705 and 1706, and is then passed to Rendering Device 1707.

FIG. 17 differs from FIG. 16 in its incorporation of Trust Plugins 1708 and 1709, and IRP 1710. When initially registered with Real Networks G2 Client Core 1702, Trust Plugins 1708 and 1709 inform Real Networks G2 Client Core 1702 that they can process content of type RNWK-Protected. Whenever Real Networks G2 Client Core 1702 encounters a stream of this type, it is then enabled to create an instance of the trust plugin to process the stream, e.g., Trust Plugin 1708. It then passes the stream to the trust plugin.

The stream passed to Trust Plugin 1708 may be in the format shown in FIG. 14. In such a case, Trust Plugin 1708 extracts DigiBox 1404 from Media Properties Header 1402. It also extracts the content id and original mime type from Media Properties Header 1402. The Trust Plugin first checks to see if any other stream with the same content identifier has been opened. If so, then DigiBox 1404 is not processed further. Instead, the key and IV from the box for this other stream are used. This avoids the time cost of opening a second

box. Also, this ensures that a user is only asked to pay once even if there are multiple protected streams. By sharing content ids, keys, and IVs, several files can be played with the user only paying once. This is useful when SMIL is used to play several RMFF files as a single presentation.

In an alternate and possibly more secure embodiment, this check is not performed, and the key and IV from the current DigiBox are used even if another stream with the content identifier has already been opened.

If no other stream has been identified with the same content identifier, Trust Plugin 1708 passes DigiBox 1404 to IRP 1710. IRP 1710 may be a software process running on the same computer as Real Networks G2 Client Core and Trust Plugin 1708. IRP 1710 may run in a protected environment or may incorporate tamper resistance techniques designed to render IRP 1710 resistant to attack.

IRP 1708 may process DigiBox 1404 and extract a cryptographic key and an IV, which may then be passed to Trust Plugin 1708. Trust Plugin 1708 may then use this information to decrypt Encrypted Contents 1406.

Trust Plugin 1708 uses the original mime type information extracted from Media Properties Header 1402 to create an instance of the rendering plugin to be used for the content (e.g., Rendering Plugin 1705). Once this is done, Trust Plugin 1708 behaves like an ordinary rendering plugin to the Real Networks G2 Client Core 1702, in that Real Networks G2 Client Core 1702 passes streamed information to Trust Plugin 1708, which decrypts that information and passes it to Rendering Plugin 1705. From the perspective of Real Networks G2 Client Core 1702, Trust Plugin 1708 constitutes the appropriate rendering pluin, and the core is not aware that the information is being passed by Trust Plugin 1708 to a second plugin (e.g., Rendering Plugin 1705).

Similarly, from the point of view of Rendering Plugin 1705, Trust Plugin 1708 behaves like Real Networks G2 Client Core 1702. Thus although Rendering Plugin 1705 receives decrypted stream information from Trust Plugin 1708, Rendering Plugin 1705 operates exactly as if the information had been received directly from Real Networks G2 Client Core 1702. In this manner, content formatted for Rendering Plugin 1705 may instead be first processed by Trust Plugin 1708, without requiring any alteration to Real Networks G2 Client Core 1702 or Rendering Plugin 1705.

Trust Plugin 1708 may also perform other processing that may be helpful for security purposes. For example, Trust Plugin 1708 may watermark the decrypted file prior to passing it to Rendering Plugin 1705, keeping in mind that the watermark algorithm must be such that it will survive decompression of the file by Rendering Plugin 1705.

MP3 Embodiment

The techniques described above can also be applied to MP3 streaming content.

The MP-3 specification does not define a standard file format, but does define a bit stream, which is illustrated in FIG.18. In FIG. 18, MP-3 Bit Stream 1801 includes Content 1802. Content 1802 is divided into frames, shown as Frame 1803, Frame 1804 and Frame 1805. The dots between Frame 1804 and 1805 symbolize the fact that Content 1802 may include a large number of frames.

Each frame includes its own small header, shown in FIG. 18 as Headers 1806, 1807 and 1808.

Many MP3 players support a small trailer defined by the ID3 V1 specification, shown as Trailer 1809. This is a 128 byte trailer for carrying fields like artist, title and year, shown as Fields 1810, 1811 and 1812. The ID3 V1 trailer is ignored by players not designed to read such trailers, since it does not appear to be valid MP3 data.

FIG. 19 shows one embodiment of protection applied to the MP3 format. This protected format constitutes File 1908 and includes the following items:

- Unencrypted MP3 Content 1912. This is the first information encountered by a player, and will be rendered by any standard MP3 player. It can include a message to the user indicating that the content is protected and providing instructions as to how the content can be accessed (e.g., a URL for a trust plugin, instructions on payment mechanisms, etc.) Unencrypted MP3 Content 1912 may include a "teaser," consisting of an initial portion of the content (e.g., 30 seconds), which is rendered at no cost, thereby allowing a user to sample the content prior to making a decision to purchase it.
- Encrypted MP-3 Content 1901, which may include thousands of MP-3 frames. In one embodiment, the first eight frames out of every 32 frames are encrypted. Thus, one-quareter of the frames are rendered unuseable unless a player is able to decrypt them. In practice, this may render the content un-sellable or unuseable, without imposing excessive encryption or decryption costs. To further reduce encryption and decryption costs, only 32 bytes in each frame are encrypted. In a current embodiment, these are the first 32 bytes after the header and CRC information. In a different embodiment, a different 32 bytes may be encrypted in every frame. In a current embodiment, the content is encrypted with the DES using algorithm output-feedback mode. The initial IV for the file is randomly generated and then xored with the frame number to generate a unique IV for each frame.

Many alternate embodiments may exist, including encrypting more or less information, and using different encryption algorithms.

- ID3 V1 Trailer 1902, including 128 bytes.

- Content ID 1903, including 16 bytes. This is used by the player application to avoid opening DigiBoxes which it has already opened.
- DigiBox 1904, which may comprise approximately 18K bytes. It includes Key 1909, IV 1910 and Watermarking Instructions 1911. Watermarking Instructions 1911 may be used in a process of watermarking the associated content.
- Address 1905, which contains the address in the file of Content ID 1903 and consists of 4 bytes.
- Trust ID 1906, which identifies this trusted MP-3 file and consists of 16 bytes.
- ID3 V1 Trailer 1907, which is a copy of Trailer 1902.

A conventional MP3 player encountering File 1908 would be unable to render Content 1901, since at least a portion of that content is encrypted. Such a player would most likely read through to Trailer 1902 and cease processing at that point. A conventional player looking for the ID3 trailer information will seek to the end and find it.

FIG. 20 illustrates one embodiment of an MP3 player designed to process and render protected content. This figure shows MP3 Player 2001, which includes Buffer 2006 and Decompressor 2007, and renders content to Rendering Device 2008. In one embodiment, this is a modified version of a player distributed by Sonique.

Player 2001 obtains Protected MP3 File 2002 through any standard interface. Protected MP3 File 2002 may have the format illustrated in FIG. 19.

When Player 2001 is asked to play Protected MP3 File 2002, Player 2001 first calls Trust Plug-In 2003, which includes Approval Function 2009 and Decrypt Function 2005. Trust Plugin 2003 calls Approval Function 2009 to determine if Protected MP3 File 2002 is protected and whether authorization exists to play the file. Approval Function 2009 is first given a pointer to Protected MP3 File 2002. It then checks Protected MP3 File 2002 for the presence of Trust ID 1906. If Trust ID 1906 is not found, Approval Function 2009 returns an indicator that the file is not protected. Player 2001 then proceeds to render the file as a normal MP3 file.

If Trust ID 1906 is found, Approval Function 2009 checks Content ID 1903 to see if it matches the Content ID of a file that has already been opened.

If Protected MP3 File 2002 has not been previously opened, DigiBox 1904 is retrieved by Approval Function 2009, and is passed to IRP 2004, which may include software running in a protected environment, or incorporating tamper resistance. IRP 2004 attempts to open DigiBox 1904 in compliance with the rules associated with that DigiBox. One such rule may require, for example, that the user indicate assent to pay for use of the content. If DigiBox 1904 cannot be opened (e.g., the user refuses to pay) a value is

returned to Approval Function 2009 indicating that the file is protected and may not be played.

If DigiBox 1904 is opened in compliance with applicable rules, the key and IV are retrieved and passed to Decrypt Function 2005. The key and IV are stored with the content id for later re-use and Decrypt Function 2005 is initialized. This may improve overall system performance, since it reduces the number of times a DigiBox must be opened. Each such action may introduce significant latency.

On the other hand, storing this information in unprotected memory may reduce overall system security. Security may be enhanced either by not storing this information (thereby requiring that each DigiBox be opened, even if the corresponding file has already been opened through another DigiBox), or by storing this information in a protected form or in a secure location.

The stored key, IV and content id are referenced when Approval Function 2009 first checks Content ID 1903 to determine if it matches the Content ID of an already opened file. If the new Content ID matches a stored Content ID, Decrypt Function 2005 is reinitialized using the stored key and IV corresponding to the matching content id an value indicating that this is a protected file for which play is authorized is returned to Approval Function 2009.

Once Protected MP3 File 2002 has been opened, each time Player 2001 needs a packet, Player 2001 reads it into Buffer 2006, strips off the header and CRC and passes the remaining data and a frame number to Decrypt Function 2005, which decrypts the frame if necessary, and returns it to Player 2001.

In a current embodiment, although audio content is encrypted, headers or trailers are not encrypted. This allows the Player 2001 to process information in headers or trailers without intervention from Approval Function 2009 or Decrypt Function 2005. This allows Player 2001 to place information such as playing time, artist and title into a playlist display, and initialize Decompressor 2007, without any action required from Trust Plugin 2003.

Commerce Appliance Embodiment

This section will describe an embodiment, comprising a Commerce Appliance architecture designed to allow persistent control of digital works in consumer electronics devices. Although this is described as a separate embodiment, it should be understood that the features of this embodiment may be combined with, or supplant, the features of any of the embodiments provided elsewhere in this description.

In one embodiment, this section will describe modifications to the MPEG-4 standard designed to support the association of persistent rules and controls with MPEG-4

content, as well as elements necessary for a Commerce Appliance to use such content. This is intended, however, merely as an example.

In one embodiment, shown in FIG. 23, each Commerce Appliance 2301 includes a CMPS ("Content Management and Protection System") 2302. Each CMPS is responsible for governing the use of controlled content, including decrypting the content and ensuring that the content is only used as permitted by associated rules.

Each governed digital work is associated with one or more CMPOs (Content Management Protection Object), e.g., CMPOs 2303. Each CMPO may specify rules governing the use of the digital work, and may include keys used to decrypt the work.

CMPOs may be organized in an hierarchical fashion. In one embodiment, a content aggregator (e.g., a cable channel, a web site, etc.) may specify a Channel CMPO ("CCMPO") used to associate certain global rules with all content present on that channel. Each independent work may in turn have an associated Master CMPO ("MCMPO") used to associate rules applicable to the work as a whole. Each object (or Elementary Stream, in MPEG-4) may have associated with it a CMPO containing rules governing the particular object.

In one exemplary application, Commerce Appliance 2301 may be an MPEG-4 player containing CMPS 2302. Upon receipt of a user command to play a particular work, CMPS 2302 may download a MCMPO associated with the work and obtain rules, which may include conditions required for decryption and viewing of the work. If the rules are satisfied, CMPS 2302 may use keys from the MCMPO to decrypt any Elementary Streams ("ES"), and may pass the decrypted ESs into the buffers. Composition and rendering of the MPEG-4 work may thereafter proceeds according to the MPEG-4 standard, except that any storage location or bus which may contain the work in the clear must be secure, and CMPS 2302 may have the ability to govern downstream processing, as well as to obtain information regarding which AVOs were actually released for viewing.

In a variation, the process of obtaining and governing the work may include downloading a CCMPO which applies rules governing this and other works. If rules contained in the CCMPO are satisfied, CMPS 2302 may obtain a key used to decrypt the MCMPO associated with the particular work to be viewed.

In another variation, a CMPO may be associated with each ES. In this variation, the MCMPO supplies one or more keys for decryption of each CMPO, and each CMPO may in turn supply a key for decryption of the associated ES.

Commerce Appliance 2301 is a content-rendering device which includes the capability of supporting distributed, peer management of content related rights by securely

applying rules and controls to govern the use of content. Commerce Appliance 2301 may include general-purpose functions devoted to acquisition and managed rendering of content (e.g., a DVD (and/or any other optical disk format) player is able to play a DVD (and/or any other optical disk format) disk and output content to a television.) Commerce Appliance 2301 may make use of any of the means for protecting and using digital content on high capacity optical disk, in one non-limiting example, a DVD disk, as described in the aforementioned Shear patent application.

Commerce Appliance 2301 also includes special-purpose functions relating to other management and protection of content functions. These special-purpose functions may be supported by one or more embedded or otherwise included CMPS 2302 in the form of a single CMPS or a cooperative CMPS arrangement, and may include a user interface (e.g., User Interface 2304) designed to display control-related information to the user and/or to receive control-related information and directions from the user. Commerce Appliance 2301 may also be designed so that it is networkable with other Commerce Appliances (e.g., a set-top box connected to a DVD player and a digital television) and/or with other devices, such as a computer arrangement, which may also include one or more CMPSs.

An important form of Commerce Appliance specifically anticipates secure coupling on a periodic or continual fashion with a computer managed docking environment (e.g., a standalone computer or other computer managed device which itself may be a Commerce Appliance) where the one or more CMPSs of the Commerce Appliance interoperate with the docking environment to form a single user arrangement whose performance of certain functions and/or certain content usage events is enabled by such inter-operation through, at least in part, cooperation between CMPSs and content usage management information of the Commerce Appliance and the trust environment capabilities of the docking environment, (e.g., further one or more CMPSs and content usage management information, such as, for example, information provided by use of CI).

An exemplary Commerce Appliance may be designed to comply with the emerging MPEG-4 standard for the formatting, multiplexing, transmission, compositing, and rendering of video and other types of information.

Commerce Appliance 2301 may be any computing device, one non-limiting example of which is a Personal Computer (PC) that includes MPEG-4 software (and/or hardware) for rendering content. In accordance with the present invention, the PC may also use one or more CMPSs as described herein.

The commerce appliance function is not restricted to streamed channel content but may include various browser-type applications consisting of aggregated composite content

such as still imagery, text, synthetic and natural video and audio and functional content such as applets, animation models and so on. these devices include browsers, set-top boxes, etc.

Content Management and Protection System (CMPS)

Each commerce appliance includes one or more CMPS (e.g., CMPS 2302). The CMPS is responsible for invocation and application of rules and controls, including the use of rules and controls to govern the manner in which controlled content is used.

Particular functions of CMPS 2302 include the following:

(a) Identification and interpretation of rules.

CMPS 2302 must determine which rules are to be applied, and must determine how those rules are to be interpreted in light of existing state information. In one embodiment, this requires that CMPS 2302 obtain and decrypt one or more CMPOs 2303 associated with a work.

(b) Identification of content associated with particular rules.

CMPS 2302 must determine which content is governed by particular one or more rules. This may be accomplished by obtaining information from one or more CMPOs 2303 and/or other CI. In one embodiment, a CCMPO may identify a set of works, a MCMPO may identify a particular work and a CMPO may identify a particular ES or Audio Visual Object ("AVO").

(c) Decryption of content as allowed by the rules.

CMPS 2302 may be designed so that all content is routed through CMPS 2302 for decryption, prior to reinsertion into the data flow required by the relevant standard. In the case of MPEG-4, for example, the output from Demux 2305 may be fed into CMPS 2302. CMPS 2302 may then decrypt the content and, if relevant rules and controls are satisfied, feed the content into the MPEG-4 buffers. From that point, the data flow associated with the content may be as described by MPEG-4.

(d) Control of content based on rules.

CMPS 2302 may be used to control usage of content after the initial decryption, for example, through the use of secure event management as described in the incorporated Ginter '333 patent application. In the case of MPEG-4 systems, this may require that CMPS 2302 exercise control over hardware and/or software which performs the following functions: demuxing (performed by Demux 2305), decompression/buffering/decode into AVOs (performed by Scene Descriptor Graph 2306, AVO Decode 2307 and Object Descriptors 2308), scene rendering (performed in Composite and Render 2309).

CMPS 2302 may also be used to control use and consequences according to: (1) generational copy protection rules such as the CGMS and/or SGMS standards; (2) various Conditional Access control methods, such as those proposed and/or implemented by NDS as described in MPEG-4 document M2959, DAVIC "Copyright Control Framework" document, and in other publications; (3) a Rights Management Language, such as those proposed in the Ginter '333 patent application and/or as described by U.S. Patent No. 5,638, 443 to Stefik, et al.; (4) use policies described in accordance with AT&T's Policy Maker, as described by Blaze, Feigenbaum, and Lacy; (5) the CCI layer bits for IEEE 1394 serial bus transmission as specified by the DTDG subgroup of the DVD Copy Protection Technical Working Group and/or as implemented by the Hitachi, Intel, Matsushita, Sony and Toshiba proposed standard (hereafter "the five company proposal"); (6) controls transmitted using any secure container technology such as, for example, IBM Cryptolope; (7) any other means for specifying use rules and consequences.

(e) Monitoring use of content.

CMPS 2302 may be used to monitor content to: (i) ensure that rules are being complied with; (ii) ensure that no attempts are being made to tamper with the system or protected content; and (iii) record information used by rules, including usage information needed for payment purposes.

(f) Updating user budgets.

CMPS 2302 may be used to update user or other budgets to reflect usage.

(g) Exhaust information.

CMPS 2302 may be used to output payment and usage information ("exhaust information") to external processes, including one or more Commerce Utility Systems.

(h) Hardware identification and configuration.

(i) Obtaining new, additional, and/or augmented rules from an external process, one non-limiting example of which is a Rights and Permission Clearinghouse as described in the incorporated Shear patent application.

(j) Receiving keys, digital credentials, such as certificates, and/or administrative information, from certifying authorities, deployment managers, clearinghouses, and/or other trusted infrastructure services.

(k) Securely sending and/or receiving user and/or appliance profiling and/or attribute information.

(l) Securely identifying a user or a member of a class of users who requests content and/or CMPO and/or CMPS usage.

(m) Securely certifying or otherwise guaranteeing the authenticity of application code, for example certifying within CMPO 2301 and/or CMPS 2302 that application code containing rules and/or other application information, such as information written in Java code for conditional execution within a Commerce Appliance, and/or that executes at least in part outside of CMPO 2301 and/or CMPS 2302, has not been altered and/or has been delivered by a guaranteed (e.g., trusted) party.

(n) Securely processing independently delivered CI, such as described in the incorporated Ginter '333 patent application, to perform content usage control that protects the rights of plural, independent parties in a commerce value chain.

(o) Securely performing watermarking (including, for example fingerprinting) functions, for example as described in the Ginter '333 patent application and as incorporated herein, for example including interpreting watermarking information to control content usage and/or to issue an event message, wherein such event message may be reported back to a remote authority, such as, for example, a MCMPO rights clearinghouse management location.

CMPS 2302 may be used to identify and record the current hardware configuration of the Commerce Appliance and any connected devices (e.g., which loudspeakers are available, identification of attached monitors, including whether particular monitors have digital output ports, etc.) If attached devices (such as loudspeakers) also include CMPSs, the CMPSs may be used to communicate for purposes of coordination (e.g., a CMPS in a set-top box and/or loudspeaker arrangement may communicate with a CMPS in a downstream digital television or other display device to establish which CMPS will be responsible for governance or the nature of cooperative governance through a virtual rights process, said process optionally involving a rights authority server that may find, locate, provide, aggregate, distribute, and/or manage rights processes, such as described in the aforementioned Shear patent application, for employing plural CMPSs, for example, for a single user content processing and usage arrangement).

The present invention includes arrangements comprising plural Commerce Appliances and/or CMPSs in one or more user locations, non-limiting examples of which include a home, apartment, loft, office, and/or vehicle, such as a car, truck, sports utility vehicle, boat, ship, or airplane, that may communicate among themselves at least occasionally and may comprise a virtual network that operates in a logically cooperative manner, through at least in part the use of such CMPSs, to ensure optimal commercial flexibility and efficiency and the enforcement of rights of commerce value chain participants, including financial and copyright rights of providers, infrastructure rights of

appliance providers, societal rights of government and/or societal bodies, and privacy rights of all parties, including consumers. Information related to interaction among such a network of value chain participants, including content usage auditing, content usage consequence, and CI specification, can be securely, variably reported to parties having right to such information, through, at least in part, use of such CMPSs, for example, as described in the aforementioned Ginter '712 patent application regarding the information reporting functioning of VDE nodes.

In one embodiment, shown in FIG. 24, CMPS 2401 consists of special-purpose hardware and resident software or firmware. These include the following:

(a) One or more processors or microcontrollers e.g. CPU 2402. CPU 2402 controls the overall processing of CMPS 2401, including execution of any necessary software.

(b) One or more external communications ports, e.g., Port 2403. Port 2403 communicates with External Network 2404, which may include LANs, WANs or distributed networks such as the Internet. External communications ports may also include one or more IEEE 1394 serial bus interfaces.

(c) Memory 2405. Types of memories which may be included in Memory 2405-- and examples of the information they may store -- are the following:

i. ROM 2406. ROM 2406 may include any information which is permanently stored in CMPS 2401, such as (1) CMPS Operating System 2407 and/or CMPS BIOS 2408, (2) Rules/Controls 2409 which are permanently stored in the CMPS; (3) Control Primitives 2410 which may be used to build rules or controls; (4) Keys 2411 associated with the CMPS, including a Public/Private Key Pair; (5) one or more Certificates 2412 designed to identify CMPS 2401 and/or the device, including version information; (6) Hardware Signature Information 2413 used to check for tampering (e.g., a hashed signature reflecting the expected hardware state of the device).

ii. RAM 2414. RAM 2414 may hold current state information needed by CMPS 2401, as well as information temporarily stored by CMPS 2401 for later use. Information stored in RAM 2414 may include the following: (1) Software 2415 currently executing in CPU 2402; (2) CMPOs 2416 which are currently active; (3) Content Object Identification 2417 of those content objects which are currently active (in an MPEG 4 system this would constitute, for example, an identification of active AVOs); (4) Rules 2418 which are currently active; (5) State Information 2419 regarding the current state of use of content, including an identification of any higher-order organization (in an MPEG-4 system this would constitute an identification of the scene descriptor tree and the current

state of composition and rendering); (6) Stored Exhaust Information 2420 relating to use and/or the user, designed for external transmission; (7) Updated Budget Information 2421; (8) Content 2422; (9) Active Content Class Information 2423; and (10) Active User Identification 2424, including identification characteristic information.

iii. NVRAM 2425 (e.g., flash memory). This type of memory may hold information which is persistent but changeable, including at least some: (1) Budget Information 2426; (2) User Information 2427, such as identification, credit card numbers; preferred clearinghouses and other Commerce Utility Systems; (3) User Preferences 2428, such as preferences, profiles, and/or attribute information; and (4) Appliance Information 2429, such as attribution and/or state information.

The types of information described above and stored in CMPS Memory 2405 may be stored in alternative of the above memory types, for example, certain budget information may be located in ROM, information regarding specific one or more clearinghouses may be stored in ROM, certain active information may be moved into NVRAM, etc.

Budget information may include stored budgets made up of, for example:

(1) electronic cash;

(2) pre-authorized uses (e.g., based on a prepayment, the user has the right to watch 12 hours of programming).

(3) Security budgets related to patterns reflecting abnormal and/or unauthorized usage, for example, as described in the incorporated Shear patent, wherein such budgets restrict and/or report certain cumulative usage conduct.

(4) electronic credit, including credit resulting from usage events such as attention to promotional material and/or the playing of multiple works from one or more classes of works (e.g., certain publisher's works) triggering a credit or cash refund event and/or a discount on future playing of one or more of such publisher's works, such as other works provided by such publisher.

User information may include the following types of information for one or more authorized users of the Commerce Appliance:

(1) Name, address, telephone number, social security number or other identifier

(2) Information used to authenticate the user, which may include a user selected password and/or biometric data, such as fingerprints, retinal data, etc.

(3) User public/private key pair

(4) User attribute and/or profiling information.

iv. Removable Memory 2430. This may include any type of removable memory storage device, such as smart cards, floppy disks or DVD disks. If the commerce appliance is designed to play content received on removable memory devices (e.g., a DVD player), that capability may be used for purposes of the CMPS.

Memory 2405 may include a protected database, in which certain control, budget, audit, security, and/or cryptographic information is stored in secure memory, with complete information stored in an encrypted fashion in unsecure memory.

(d) Encryption/Decryption Engine 2431. CMPS 2401 must include a facility for decrypting received information, including content and CMPOs and/or other. CMPS 2401 may also include a facility for encrypting information if such information is to be transmitted outside the secure boundaries of CMPS 2401. This may include exhaust sent to clearinghouses or other external repositories; and content sent across unsecured buses for usage, such as content sent across IEEE 1394 Serial Bus 2432 to a computer central processing arrangement or to a viewing device such as a monitor, wherein a receiving CMPS may be employed to control such content's usage, including, for example, decrypting such content, as appropriate. Encryption/Decryption Engine 2431 may include a Random Number Generator 2433 used for the creation of keys or key pairs that can be used to identify and assure the uniqueness of CMPSs and support the opening of secure communication channels between such secure content control secure encryption/decryption arrangements.

(e) Secure Clock/Calendar 2434. CMPS 2401 may include Secure Clock/Calendar 2434 designed to provide absolute information regarding the date and time of day, information regarding elapsed absolute time, and/or relative timing information used to determine the elapsed time of operations performed by the system. Secure Clock/Calendar 2434 may include Battery Back Up 2435. It may further include Sync Mechanism 2436 for synchronization with outside timing information, used to recover the correct time in the event of a power loss, and/or to check for tampering.

(f) Interface 2437 to blocks used for content rendering and display. This interface is used for controlling rendering and display, based on rules, and for obtaining feedback information, which may be used for budgeting purposes or for providing information to outside servers (e.g., information on which content was actually displayed, which choices the user invoked, etc.) In the case of an MPEG-4 player such as is shown in

FIG. 23, this may include control over Commerce Appliance circuitry which handles, for example, buffering, the scene descriptor graph, AVO decode, object descriptors and composite and rendering (e.g., Control Lines 2310, 2311 and 2312).

Feedback Path 2313 from Composite and Render block 2309 may allow CMPS 2302 to determine whether and when content has actually been released to the viewer. For example, Composite and Render block 2309 can issue a start event to CMPS 2302 when an AVO object is released for viewing, and can issue a stop event to CMPS 2302 when the AVO object is no longer being viewed.

Feedback from Composite and Render block 2309 may also be used to detect tampering, by allowing CMPS 2302 to match the identification of the objects actually released for viewing with the identification of the objects authorized for release. Start and end time may also be compared with the expected elapsed time, with a mismatch possibly indicative of the occurrence of an unauthorized event.

In one embodiment, the following protocol may be used for feedback data:

start <id>, T, <instance number><clock time><rendering options>

Sent if elementary stream <id> is reachable in the SD-graph at time *T*, but not at time *T-1*.

end <id>, T, <instance number><clock time><rendering options>

T constitutes presentation time, clock time constitutes the wall clock time, including day and date information, and rendering options may include such information as QoS and rate of play (e.g., fast forward).

Sent if elementary stream **<id>** is reachable in the SD-graph at time *T-1* but not at time *T*. A SD-graph stream is reachable if, during traversal of the SD-graph for display update, the renderer encounters a node that the SD-graph update stream <id> created or modified. This implies that all nodes in the tree need an update history list. This list need not be as large as the number of streams. Further, it can be labeled to indicate if the CMPS will be watching for stream, if not labeled it will not record them. An AV elementary stream is reachable if the stream's content was rendered.

For SD-graph update streams, the object instance number is ignored. For AV streams, the instance number can be used to disambiguate the case where the display shows two or more instances of the same data stream simultaneously. Instance numbers do not have to count up. In this case, they are simply a unique id that allows the CMPS to match a start event with an end event.

In a second embodiment, CMPS 2302 may include some special purpose hardware in combination with general purpose hardware which is also used for other functions of the

device. In this embodiment, care must be taken to ensure that commercially trusted CMPS functions are performed in a secure and tamper-resistant manner, despite the use of general purpose hardware. Each of the elements recited above may include dedicated CMPS functions and general purpose device functions:

(a) CPU/microcontroller. This may include one or more devices. If more than one device is included (e.g., a CPU and a DSP, a math coprocessor or a commerce coprocessor), these devices may be included within the same package, which may be rendered tamper-resistant, or the devices may communicate on a secure bus. The CPU may include two modes: a secure CMPS mode, and an unsecure general purpose mode. The secure CMPS mode may allow addressing of secure memory locations unavailable to the processor in general purpose mode. This may be accomplished, for example, by circuitry which remaps some of the available memory space, so that, in unsecure mode, the CPU cannot address secure memory locations.

(b) External communications ports. If the device, for example, a Commerce Appliance, is capable of receiving content or other information through a communications port (e.g., a cable connection, an Internet connection), this communications port can be used for CMPS purposes. In such a case, CMPS accesses to the external communications port is preferably designed to avoid or minimize interference with the use of such port for receipt of content.

(c) Memory. In some applications and embodiments, it is possible to operate a Commerce Appliance without NVRAM, wherein information that may be needed for CMPS operation that would employ NVRAM would be loaded into RAM, as required. ROM, RAM and NVRAM may be shared between CMPS uses and general uses. This can be accomplished in any of the following ways, or in a combination of these ways: (1) Some memory space may be rendered off-limits to general purpose uses, for example by remapping; (2) the entirety of the memory may be rendered secure, so that even portions of the memory being used for non-secure purposes cannot be observed or changed except in a secure and authorized manner; (3) CMPS information may be stored in an encrypted fashion, though this requires at least some RAM to be secure, since the CMPS will require direct access to unencrypted information stored in RAM.

(d) Encryption/decryption engine. Encryption and decryption functions, including key generation, may be handled by special purpose software running on a general purpose processor arrangement, particularly, for example, a floating point processor or DSP arrangement. That processor arrangement may also be used for purposes of decompressing and displaying content and/or for handling watermarking/fingerprinting

insertion and/or reading. Alternatively, the device may include native encryption and decryption functions. For example, various emerging standards may require at least some degree of encryption and decryption of content designed to be passed across unsecure buses within and among devices such as DVD players, such as the "five company proposal" and other IEEE 1394 related initiatives. Circuitry designed to perform such encryption and decryption may also be usable for CMPS applications.

(e) Secure clock/calendar. The underlying device may already require at least some clock information. MPEG-4, for example, requires the use of clock information for synchronization of Elementary Streams. A secure CMPS clock can also be used for such purposes.

In a third embodiment, CMPS 2302 can be primarily software designed to run on a general purpose device which may include certain minimal security-related features. In such a case, CMPS 2302 may be received in the same channel as the content, or in a side-band channel. An I-CMPO and/or other CI may specify a particular type of CMPS, which Commerce Appliance 2301 must either have or acquire (e.g., download from a location specified by the I-CMPO), or CMPS 2302 may be included, for example, with an I-CMPO.

A software CMPS runs on the CPU of the Commerce Appliance. This approach may be inherently less secure than the use of dedicated hardware. If the Commerce Appliance includes secure hardware, the software CMPS may constitute a downloadable OS and/or BIOS which customizes the hardware for a particular type of commerce application.

In one embodiment, a software CMPS may make use of one or more software tamper resistance means that can materially "harden" software. These means include software obfuscation techniques that use algorithmic means to make it very difficult to reverse engineer some or all of a CMPS, and further make it difficult to generalize from a reverse engineering of a given one or more CMPS. Such obfuscation is preferably independent of source code and object code can be different for different CMPSs and different platforms, adding further complexity and separation of roles. Such obfuscation can be employed "independently" to both CI, such as an CMPO, as well as to some or all of the CMPS itself, thus obscuring both the processing environment and executable code for a process. The approach is also applicable for integrated software and hardware implementation CMPS implementations described above. Other tamper resistance means can also be employed, including using "hiding places" for storing certain state information in obscure and unexpected locations, such as locations in NV memory used for other purposes, and data hiding techniques such as watermarking/fingerprinting.

Association of CMPS With a Commerce Appliance

A CMPS may be permanently attached to a particular device, or may be partially or fully removable. A removable CMPS may include software which is securely loaded into a Commerce Appliance, and/or removable hardware. A removable CMPS may be personalized to one or more particular users, including user keys, budget information, preferences, etc., thereby allowing different users to use the same Commerce Appliance without commingling budgets and/or other rights, etc.

A CMPS may be designed for operation with certain types of content and/or for operation with certain types of business models. A Commerce Appliance may include more than one type of CMPS. For example, a Commerce Appliance designed to accept and display content pursuant to different standards may include one CMPS for each type of format. In addition, a Commerce Appliance may include a CMPS provided by a particular provider, designed to preferentially display certain types of content and to preferentially bill for such content through a particular channel (e.g., billing to one or more particular credit cards and/or using a particular one or more clearinghouses).

Source of Rules

The CMPS must recognize those rules which are to be applied to particular content. Such rules may be received by the CMPS from a variety of sources, depending on the particular embodiment used:

(a) CMPO. The rules may be included within a CMPO (e.g., CMPO 2303) and/or other CI. The CMPO and/or other CI may be incorporated within a content object or stream (as, e.g., a header on an MPEG-4 ES), and/or may be contained within a dedicated content object or stream encoded and received as per the underlying standard (e.g., an MPEG-4 CMPO ES), and/or may be received outside the normal content stream, in which event it may not be encoded as per the underlying standard (e.g., a CMPS received as an encrypted object through a sideband channel).

(b) CMPS. Rules may be permanently and/or persistently stored within a CMPS, e.g., Rules 2409. A CMPS may include default rules designed to handle certain situations, for example, where no CMPO and/or other necessary CI is received (e.g., content encoded under an earlier version of the standard which did not incorporate CMPOs, including MPEG-4 version 1). Complete rules which are stored within the CMPS may be directly or indirectly invoked by a CMPO and/or other CI. This may occur through the CI identifying particular rules through a pointer, and/or it may occur through the CI identifying itself and the general class of control it requires, with the CMPS then applying particular rules specific to that CMPS.

Rule "primitives" may also be stored within the CMPS (e.g., Control Primitives 2410). The CMPO and/or other CI may invoke these primitives by including a sequence of macro-type commands, each of which triggers a sequence of CMPS primitives.

(c) User. The user may be given the ability to create rules relating to the particular user's preferences. Such rules will generally be allowed to further restrict the use of content, but not to expand the use of content beyond that which would otherwise be allowed. Examples include: (a) rules designed to require that certain types of content (e.g., adult movies) only be accessible after entry of a password and/or only to certain CMPS users (e.g. adults, not children, as, for example, specified by parents and/or a societal body such as a government agency); (b) rules designed to require that only particular users be allowed to invoke operations requiring payment beyond a certain limit and/or aggregate payment over a certain amount.

The user may be allowed to create templates of rules such as described in the aforementioned Ginter '333 patent application (and incorporated herein). In addition, a CMPS arrangement, and/or a particular CMPO and/or other CI, may restrict the rules the user is allowed to specify. For example, a CI may specify that a user can copy a work, but cannot add rules to the work restricting the ability of a recipient to make additional copies (or to be able to view, but only after a payment to the first user). User supplied one or more rules may govern the use of – including privacy restrictions related to -- payment, audit, profiling, preference, and/or any other kind of information (e.g., information result as a consequence of the use of a CMPS arrangement, including, for example, use of secured content). Such user supplied one or more rules can be associated with the user and/or one or more Commerce Appliances in a user arrangement, whether or not the information is aggregated according to one or more criteria, and whether or not user and/or appliance identification information is removed during aggregation and/or subsequent reporting, distribution, or any other kind of use.

The ability to allow the user to specify rules allows the CMPS to subsume (and thereby replace) V-chips, since a parent can use content rating information to specify precisely what types of information each viewer will be allowed to watch (e.g., violent content can only be displayed after entry of a certain password and/or other identifier, including, for example, insertion of a removable hardware card (smart or rights card) possessed by a user).

(d) External network source. The rules may be stored on an external server. Rules may be addressed and downloaded by the CMPS if necessary (e.g., either the CMPO and/or other CI and/or the CMPS contains a pointer to certain rules location(s), such as one

or more URLs). In addition, content providers and/or clearinghouses may broadcast rules designed for general applicability. For example, a content provider might broadcast a set of rules providing a discount to any user participating in a promotional event (e.g., by providing certain user information). Such rules could be received by all connected devices, could be received by certain devices identified as of interest by the content provider (e.g., all recent viewers of a particular program, as identified by exhaust information provided by the CMPS to a clearinghouse and/or all members having certain identity characteristics such as being members of one or more classes) and/or could be posted in central locations.

Example Embodiment

In one embodiment, a set of MPEG-4 Elementary Streams may make up a work. The Elementary Streams may be encrypted and multiplexed together to form an Aggregate Stream. One or more CMPOs may be present in such stream, or may otherwise be associated with the stream. Options are as follows:

1. Content may be streamed or may be received as static data structures.

2. A Work may be made up of a single stream or data structure, or of many separately addressable streams or data structures, each of which may constitute an Object.

3. If a Work is made up of separately addressable streams or data structures, those streams or data structures may be multiplexed together into an Aggregate Stream, or may be received separately.

4. If streams or data structures are multiplexed together into an Aggregate Stream, the streams or data structures may be encrypted prior to such multiplexing. The Aggregate Stream itself may be encrypted, whether or not the underlying streams or data structures are encrypted. The following possibilities therefore exist: (a) individual streams/data structures are unencrypted (in the clear), the Aggregate Stream is unencrypted; (b) individual streams/data structures are unencrypted prior to multiplexing, the Aggregate Stream is encrypted following multiplexing; (c) individual streams/data structures are encrypted prior to multiplexing, the Aggregate Stream is not encrypted following multiplexing; or (d) individual streams/data structures are encrypted prior to multiplexing, the Aggregate Stream is encrypted following multiplexing.

5. A CMPO may be associated with a channel (CCMPO), a work (MCMPO) or an individual Object (CMPO).

6. A CMPO may be received prior to the controlled data, may be received contemporaneously with the data, or may be received after the data (in which event use of the data must wait until the CMPO has been received).

7. A CMPO may be received as part of an Aggregate Stream or separately.

8. If a CMPO is received as part of the Aggregate Stream, it may be multiplexed together with the individual streams or data structures, or may constitute a separate stream or data structure.

9. If a CMPO is multiplexed within the Aggregate Stream, it may be encrypted or nonencrypted. If encrypted, it may be encrypted prior to multiplexing, and/or encrypted after multiplexing, if the entire Aggregate Stream is encrypted.

10. If a CMPO is received as part of the Aggregate Stream, it may be (a) a part of the stream or data structure which holds the content (e.g., a header); (b) a separate stream or data structure encoded pursuant to the same format as the streams or data structures which hold the content (e.g., an MPEG-4 ES) or (c) a separate stream or data structure encoded under a different format designed for CMPOs.

11. If a CMPO is a part of the stream or data structure which holds the content, it may be (a) a header which is received once and then persistently maintained for control of the content; (b) a header which is received at regular intervals within the stream or data structure; or (c) data distributed throughout the stream or data structure.

These various scenarios give rise to different requirements for demultiplexing and decryption of the CMPOs. FIG. 25 illustrates the following embodiment:

1. Aggregate Stream 2501 is made up of multiplexed ESs (e.g., ES 2502 and 2503). A combination of such ESs makes up a single work. Aggregate Stream 2501 is generated by a cable aggregator and received by a user's set-top box as one of a number of channels.

2. CCMPOs 2504 corresponding to each channel are sent along the cable in Header 2505 at regular intervals (e.g., once per second). When the set-top box is turned on, it polls each channel, and downloads all current CCMPOs. These are stored persistently, and are changed only if a new CCMPO is received which differs from prior CCMPOs.

3. When the user selects a channel, the set-top box addresses the associated CCMPO. The CCMPO may specify, for example, that content in this particular channel may only be accessed by subscribers to the channel. A CMPS within the set-top box accesses a user profile persistently stored in NVRAM and determines that the user is a subscriber. The CMPS deems the CCMPO rule to have been satisfied.

4. The CMPS obtains an identifier for the MCMPO associated with the work (video) currently streaming on the channel and a key for the MCMPO. If works are received serially on the channel (e.g., a television channel in which one work is provided at a time), the received MCMPO identifier may include don't care bits so that it can address any MCMPO currently on the channel.

5. The CMPS begins demuxing of Aggregate Stream 2501 (this may occur in parallel with the preceding step), and obtains the MCMPO, which is encoded into an ES multiplexed within the Aggregate Stream (e.g., MCMPO 2506). Although each ES within Aggregate Stream 2501 has been encrypted, Aggregate Stream 2501 was not encrypted following multiplexing. This allows the CMPS to demultiplex Aggregate Stream 2501 without decrypting the entire Aggregate Stream.

6. The CMPS identifies the ES which constitutes the MCMPO (e.g., ES 2503). The CMPS downloads one complete instance of MCMPO 2506 into an internal buffer, and uses the key received from CCMPO 2504 to decrypt MCMPO 2506.

7. The CMPS determines which rules are applied by MCMPO 2506. MCMPO 2506 might, for example, include a rule stating that the user can view the associated work with advertisements at a low fee, but must pay a higher fee for viewing the work without advertisements.

8. The CMPS generates an options menu, and displays that menu on the screen for the user. The menu specifies the options, including the cost for each option. Additional options may be specified, including payment types.

9. The user uses a remote control pointing device to choose to view the work at a lower cost but with advertisements. The user specifies that payment can be made from an electronic cash budget stored in the CMPS.

10. The CMPS subtracts the specified amount from the budget persistently stored in NVRAM, and generates and encrypts a message to a server associated with the cable. The message transfers the required budget to the server, either by transferring electronic cash, or by authorizing a financial clearinghouse to transfer the amount from the user's account to the cable provider's. This message may be sent immediately, or may be buffered to be sent later (e.g., when the user connects the device to the Internet). This step may be taken in parallel with decryption of the content.)

11. The CMPS obtains from MCMPO 2506 a set of keys used to decrypt the Elementary Streams associated with the work (e.g., ES 2502). The CMPS also obtains identifiers for the specific ESs to be used. Since the user has indicated that advertisements are to be included, the MCMPO identifies ESs associated with the advertisements, and identifies a Scene Descriptor Graph which includes advertisements. A Scene Descriptor Graph which does not include advertisements is not identified, and is not passed through by the CMPS.

12. The CMPS passes the decrypted ESs to the MPEG-4 buffers. The normal process of MPEG-4 decoding, compositing and rendering then takes place. The Composite

and Render block outputs Start and Stop events for each object released for viewing. The CMPS monitors this information and compares it to the expected events. In particular, the CMPS confirms that the advertisements have been released for viewing, and that each operation has occupied approximately the expected amount of time.

In another embodiment, a set-top box containing a CMPS (e.g., CMPS 2302 from FIG. 23) may have a cable input (e.g., carrying M4 Bit Streams 2314 and CMPOs 2303). The cable may carry multiple channels, each made up of two sub-channels, with one sub-channel carrying MPEG-4 ESs (e.g., M4 Bit Streams 2314), and the other sub-channel carrying CMPOs (e.g., CMPOs 2303). The sub-channel carrying CMPOs 2303 could be routed directly to CMPS 2302, with the ES channel being routed to a decryption block (operating under control of the CMPS, e.g., CR&D 2315), and then to the MPEG-4 buffers (e.g., buffers associated with Scene Descriptor Graph 2306, AVO Decode 2307 and Object Descriptors 2308). In this case, if the ESs are not encrypted, they proceed unchanged through the decryption block and into the buffers. This may occur, for example, if the ESs are being broadcast for free, with no restrictions, and/or if they are public domain information, and/or they were created prior to inclusion of CMPOs in the MPEG-4 standard.

Such an embodiment might include timing synchronization information in the CMPO sub-channel, so that CMPOs can be synchronized with the associated ESs.

The concept of incorporating two separate streams, one consisting of control information and connected directly to the CMPS, and the other consisting of ESs, may support a high degree of modularization, such that the formats of CMPOs, and particular types of CMPS's, may be changed without alteration to the underlying ES format. For example, it may be possible to change the CMPO format without the necessity for reformatting content ESs. To take another example, it may be possible to upgrade a Commerce Appliance by including a new or different CMPS, without the necessity for any changes to any of the circuitry designed to demultiplex, composite and render the content ESs. A user might obtain a CMPS on a smart card or other removable device, and plug that device into a Commerce Appliance. This could be done to customize a Commerce Appliance for a particular application or for particular content.

CMPS Interface to a CE Device

A CMPS may be designed to present a standardized interface between the general-purpose functionality of a consumer electronics device and any relevant CMPOs and/or other CI and protected content. For example, a CMPS could be designed to accept CI and encrypted ESs, and output decrypted ESs into the device's buffers. In such a case, the

manufacturer of the device would be able to design the device in compliance with the specification (e.g., MPEG-4), without concern about commerce-related extensions to the standard, which extensions might differ from provider to provider. All such extensions would be handled by the CMPS.

Initialization

1. Initialization of the CMPS.

A CMPS may be used to identify the capabilities of the Commerce Appliance in which a CMPS is installed. A CMPS permanently associated with a particular Commerce Appliance may have such information designed-in when the CMPS is initially installed (e.g., stored in ROM 2406 shown in FIG.24). A CMPS which is removable may be used to run an initialization operation in order to obtain information about the device's capabilities. Such information may be stored in a data structure stored in NVRAM 2425. Alternatively, some or all of such information may be gathered each time the device is turned on, and stored in RAM 2414.

For example, a DVD player may or may not contain a connection to an external server and/or process. A CMPO and/or other CI stored on a DVD (and/or any other format optical disk) inserted into a DVD (or any other format optical disk) player may include rules predicated on the possibility of outputting information to a server (e.g., content is free if user identification information is output), or may require a direct connection in order, for example, to download keys used to decrypt content. In such a case, the CMPS arrangement may determine the hardware functionality which is expected by or required by the CMPO, and compare that to the hardware actually present. If the CMPS determines that the CMPO and/or other CI requires a network connection , and that the DVD player does not include such a connection, the CMPS may take a variety of steps, including: (1) if the network connection is required for some options but not others, causing only those options which are possible to be displayed to the user; (2) informing the user that necessary hardware is missing; or (3) causing a graceful rejection of the disk, including informing the user of the reason for the rejection.

To take another example, a CMPO and/or other CI may include a business model which allows the user to choose among quality levels (or other forms of variations of a given work, for example, longer length and/or greater options), with a higher price being charged if the user selects a higher level of quality (e.g., music may be played at low resolution for free, but requires a payment in order to be played at a higher resolution). In such a case, the Commerce Appliance may not include loudspeakers which are capable of outputting sound at the higher resolution. The CMPS arrangement preferably identifies this situation, and either eliminates the higher resolution output as an option for the user, or

informs the user that this option costs more but provides no additional benefit given the Commerce Appliance's current functionality or given the Commerce Appliance not being docked in a user arrangement that provides higher quality loudspeakers.

If the Commerce Appliance may be hooked up to external devices (e.g., loudspeakers, display, etc.), the CMPS will require some mechanism for identifying and registering such devices. Each device may be used to make standard ID and capability information available at all times, thereby allowing the CMPS to poll all connected devices at regular intervals, including, for example, authenticating CMPS arrangements within one or more of each such connected devices. Using another approach, all devices could be used to output CMPS identification information upon power-on, with later connected devices being used to output such information upon establishment of the connection. Such identification information may take the form, for example, of authentication information provided under the "five company arrangement", such authentication methods are herein incorporated by reference.

As discussed earlier, a Commerce Appliance may be connected to multiple devices each containing its own CMPS arrangement (e.g., a DVD player may be connected to a digital TV) In such cases, the CMPSs must be able to initiate secure communication (e. g., using a scheme, for example, like the "five company proposal" for IEEE 1394 serial bus) and determine how the CMPSs will interact with respect to content communication between CMPSs and, in certain embodiments, regarding cooperative governance of such content such as describing in the incorporated Shear patent application. In one embodiment, the first CMPS arrangement to receive content might govern the control process by downloading an initial CMPO and/or other CI, and display one or more of the rules to the user, etc. The second CMPS arrangement might recognize that it has no further role to play, either as a result of a communication between the two CMPS arrangements, or as a result of changes to the content stream created by the first CMPS arrangement (which decrypted the content, and may have allowed demuxing, composition and rendering, etc.)

The relationship between upstream and downstream CMPSs arrangements may be complicated if one device handles certain aspects of MPEG-4 rendering, and the other handles other aspects. For example, a DVD player might handle demuxing and buffering, transferring raw ESs to a digital TV, which then handles composition and rendering, as well as display. In such a case, there might be no back-channel from the composition and rendering block to the upstream CMPS arrangement. CMPS arrangements are preferably designed to handle stand-alone cases (a DVD (or any other optical disk) player with a CMPS arrangement attached to a dumb TV with no CMPS), multiple CMPS arrangement

cases in which one CMPS arrangement handles all of the processing (a DVD (or other optical disk) player which handles everything through composition and rendering, with a video stream output to the digital TV (in one non-limiting example, via an IEEE 1349 serial bus) (that output stream would be encrypted as per the "five company proposal" for copy protection using IEEE 1394 serial bus transmission)) and/or shared processing between two or more CMPSs arrangements regarding some, or in certain cases, all, of such processing.

2. Initialization of a particular content stream.

The CMPS may be designed so that it can accept initialization information which initializes the CMPS for a particular content stream or channel. This header, which may be a CMPO and/or other CI, may contain information used by the CMPS to locate and/or interpret a particular content stream as well as CI associated with that stream. This initial header may be received through a sideband channel, or may be received as a CI ES such as a CMPO ES.

In one example, shown in FIG. 26, Header CMPO 2601 may include the following information:

(a) Stream/Object/CMPO ID 2602, which identifies the content streams/objects governed by Header CMPO 2601 and/or identification of CMPOs associated with each such content stream or object.

In one embodiment, Header CMPO 2601 identifies other CMPOs which contain rules and keys associated with particular content streams. In another embodiment, Header CMPO 2601 directly controls all content streams, by incorporating the keys and rules associated with such streams. In the latter case, no other CMPOs may be used.

In one embodiment, Header CMPO 2601 may be one or more CMPOs, CCMPOs, MCMPOs, and/or other CI.

(b) One or CMPO Keys 2603 for decrypting each identified CMPO.

(c) Work-Level Control 2604, consisting of basic control information associated with the work as a whole, and therefore potentially applicable to all of the content streams which make up the work. This basic control information may include rules governing the work as a whole, including options to be presented to the user.

(d) In one embodiment of this embodiment, a header CMPO may be updatable to contain User/Site Information 2605 regarding a particular user or site currently authorized to use certain content, as well as one or more rule sets under which the user has gained such authorization. A header CMPO associated with a work currently being viewed may be stored in RAM or NVRAM. This may include updated information. In one

embodiment, the CMPO may also store header CMPOs for certain works viewed in the past. In one embodiment, header CMPOs may be stored in non-secure memory, with information sufficient to identify and authenticate that each header CMPO had not been changed.

In one such header CMPO embodiment of this embodiment, the header CMPO operates as follows:

(a) The header CMPO is received by a CMPS arrangement. In the case of previously unreceived content which has now become available, the header CMPO may be received at an input port. In the case of content which is already available, but is not currently being used (e.g., a set-top box with 500 channels, of which either 0 or 1 are being displayed at any given time), CCMPOs for each channel may be buffered by the CMPS arrangement for possible use if the user invokes particular content (e.g., switches to a particular channel).

In either case, the header CMPO must include information which allows a CMPS arrangement to identify it as a header CMPO.

(b) The CMPS arrangement obtains business-model information held in the clear in the header CMPO. Business-model information may include, for example, a statement that content can be viewed for free if advertisements are included, or if the user authorizes Nielson-type information, user and/or audience measurement information, for example, content may be output to a server or otherwise copied once, but only at a price.

(c) The CMPS arrangement either accepts the business model, if the user has authorized it to accept certain types of models (e.g., the user has programmed the CMPS arrangement to always accept play with advertisements for free), rejects the business model, if the user has instructed that the particular model always be rejected, or displays the business model to the user (e.g., by presenting options on the screen).

(d) If a business model has been accepted, the CMPS arrangement then decrypts the remainder of the header CMPO. If the Commerce Appliance contains a live output connection to an external server (e.g., Internet connection, back-channel on a set-top box, etc.), and if latency problems are handled, decryption of these keys can be handled by communicating with the external server, each side authenticating the other, establishment of a secure channel, and receipt of a key from the server. If the Commerce Appliance is not at least occasionally connected to an external server, decryption may have to be based on one or more keys securely stored in the Commerce Appliance.

(e) Once a header CMPO has been decrypted, the CMPS arrangement acquires information used to identify and locate the streams containing the content, and

keys which are used to decrypt either the CMPOs associated with the content, or to directly decrypt the content itself.

(f) In one embodiment of this header embodiment, the header CMPO may contain a data structure for the storage of information added by the CMPS arrangement. Such information may include the following:

(1) Identification of user and/or Commerce Appliance and/or CMPS arrangement. In this embodiment, such information may be stored in a header CMPO in order to provide an audit trail in the event the work (including the header CMPO) is transferred (this only works if the header CMPO is transferred in a writable form). Such information may be used to allow a user to transfer the work to other Commerce Appliances owned by the user without the payment of additional cost, if such transfers are allowed by rule information associated with the header CMPO. For example, a user may have a subscription to a particular cable service, paid for in advance by the user. When a CMPS arrangement downloads a header CMPO from that cable service, the CMPS arrangement may store the user's identification in the header CMPO. The CMPS arrangement may then require that the updated header CMPO be included if the content is copied or transferred. The header CMPO could include a rule stating that, once the user information has been filled in, the associated content can only be viewed by that user, and/or by Commerce Appliances associated with that user. This would allow the user to make multiple copies of the work, and to display the work on multiple Commerce Appliances, but those copies could not be displayed or used by non-authorized users and/or on non-authorized Commerce Appliances. The header CMPO might also include a rule stating that the user information can only be changed by an authorized user (e.g., if user 1 transfers the work to user 2, user 2's CMPS arrangement can update the user information in the header CMPO, thereby allowing user 2 to view the work, but only if user 2 is also a subscriber to the cable channel).

(2) Identification of particular rules options governing use. Rule sets included in header CMPOs may include options. In certain cases, exercise of a particular option might preclude later exercise of a different option. For example, a user might be given the choice to view an unchanged work for one price, or to change a work and view the changed work for a higher price. Once the user decides to change the work and view the changed work, this choice is preferably stored in the header CMPO, since the option of viewing the original unchanged work at the lower price is no longer available. The user might have further acquired the right, or may now be presented with the option for the right, to further distribute the changed work at a mark-up in cost resulting in third party

derived revenue and usage information flowing to both the user and the original work stakeholder(s).

(3) Historical usage information. The header CMPO may include information relating to the number and types of usages. For example, if the underlying work is copied, the header CMPO may be updated to reflect the fact that a copy has been made, since a rule associated with the work might allow only a single copy (e.g., for backup and/or timeshifting purposes). To take another example, a user might obtain the right to view a work one time, or for a certain number of times. The header CMPO would then be updated to reflect each such use.

Usage information may be used to determine if additional uses are authorized by rules associated with the header CMPO. Such information may also be used for audit purposes. Such information may also be provided as usage information exhaust, reported to an external server. For example, a rule may specify that a work may be viewed for free, but only if historical usage information is downloaded to a server.

Content Management Protection Objects (CMPO)

The Content Management and Protection Object ("CMPO") is a data structure which includes information used by the CMPS to govern use of certain content. A CMPO may be formatted as a data structure specified by a particular standard (e.g., an MPEG-4 ES), or may be formatted as a data structure not defined by the standard. If the CMPO is formatted as a data structure specified by the standard, it may be received in the channel utilized by the standard (e.g., as part of a composite MPEG-4 stream) or may be received through some other, side-band method. If the CMPO is formatted as a data structure not specified by the relevant standard, it is provided and decoded using some side-band method, which may include receipt through the same port as formatted content and/or may include receipt through a separate port.

Content may be controlled at virtually any level of granularity. Three exemplary levels will be discussed herein: "channel," "work," and "object."

A "channel" represents an aggregation of works. The works may be available for selection by the user (e.g., a web site, or a video library) or may be received serially (e.g., a cable television channel).

A "work" represents a single audio-visual, textual or other work, intended to be consumed (viewed, read, etc.) by a user as an integrated whole. A work may, for example, be a movie, a song, a magazine article, a multimedia product such, for example, as sophisticated videogame. A work may incorporate other works, as, for example, in a multimedia work which incorporates songs, video, text, etc. In such a case, rights may be

associated

An "object" represents a separately addressable portion of a work. An object may be, for example, an individual MPEG-4 AVO, a scene descriptor graph, an object descriptor, the soundtrack for a movie, a weapon in a videogame, or any other logically definable portion.

Content may be controlled at any of these levels (as well as intermediate levels not discussed herein). The preferred embodiment mechanism for such control is a CMPO or CMPO arrangement (which comprises one or more CMPOs, and if plural, then plural, cooperating CMPOs). CMPOs and CMPO arrangements may be organized hierarchically, with a Channel CMPO arrangement imposing rules applicable to all contained works, a MCMPO or an SGCMPO imposing rules applicable to all objects within a work, and a CMPO arrangement imposing rules applicable to a particular object.

In one embodiment, illustrated in FIG. 27, a CMPS may download CCMPO 2701. CCMPO 2701 may include one or more Rules 2702 applicable to all content in the channel, as well as one or more Keys 2703 used for decryption of one or more MCMPOs and/or SGCMPOs. MCMPO 2704 may include Rules 2705 applicable to a single work and/or works, one or more classes and/or more users and/or user classes, and may also include Keys 2706 used to decrypt CMPOs. CMPO 2707 may include Rules 2708 applicable to an individual object, as well as Key 2709 used to decrypt the object.

As long as all objects are subject to control at some level, there is no requirement that each object be individually controlled. For example, CCMPO 2701 could specify a single rule for viewing content contained in its channel (e.g., content can only be viewed by a subscriber, who is then might be free to redistribute the content with no further obligation to the content provider). In such a case, rules would not necessarily be used for MCMPOs (e.g. Rules 2705), SGCMPOs, or CMPOs (e.g., Rules 2708). In one embodiment, MCMPOs, SGCMPOs, and CMPOs could be dispensed with, and CCMPO 2701 could include all keys used to decrypt all content, or could specify a location where such keys could be located. In another embodiment, CCMPO 2701 would supply Key 2703 used to decrypt MCMPO 2704. MCMPO 2704 might include keys used to decrypt CMPOs (e.g., Keys 2706), but might include no additional Rules 2705. CMPO 2707 might include Key 2709 used to decrypt an object, but might include no additional Rules 2708. In certain embodiments, there may be no SGCMPOs.

A CMPO may be contained within a content data structure specified by a relevant standard (e.g., the CMPO may be part of a header in an MPEG-4 ES.) A CMPO may be contained within its own, dedicated data structure specified by a relevant standard (e.g., a

CMPO ES). A CMPO may be contained within a data structure not specified by any content standard (e.g., a CMPO contained within a DigiBox).

A CCMPO may include the following elements:

(a) ID 2710. This may take the following form: <channel ID>< CMPO type><CMPO ID><version number>. In the case of hierarchical CMPO organization (e.g., CCMPOs controlling MCMPOs controlling CMPOs), CMPO ID 2711 can include one field for each level of the hierarchy, thereby allowing CMPO ID 2711 to specify the location of any particular CMPO in the organization. ID 2710 for a CCMPO may, for example, be 123-000-000. ID 2712 for a MCMPO of a work within that channel may, for example, be 123-456-000, thereby allowing the specification of 1,000 MCMPOs as controlled by the CCMPO identified as "123." CMPO ID 2711 for a CMPO associated with an object within the particular work may, for example, be 123-456-789, thereby allowing the specification of 1,000 CMPOs as associated with each MCMPO.

This method of specifying CMPO IDs thereby conveys the exact location of any CMPO within a hierarchy of CMPOs. For cases in which higher levels of the hierarchy do not exist (e.g., a MCMPO with no associated CCMPO), the digits associated with that level of the hierarchy may be specified as zeroes.

(b) Rules 2702 applicable to all content in the channel. These may be self-contained rules, or may be pointers to rules obtainable elsewhere. Rules are optional at this level.

(c) Information 2713 designed for display in the event the user is unable to comply with the rules (e.g., an advertisement screen informing the user that a subscription is available at a certain cost, and including a list of content available on the channel).

(d) Keys 2703 for the decryption of each MCMPO controlled by this CCMPO. In one embodiment, the CCMPO includes one or more keys which decrypt all MCMPOs. In an alternate embodiment, the CCMPO includes one or more specific keys for each MCMPO.

(e) A specification of a CMPS Type (2714), or of hardware/software necessary or desirable to use the content associated with this channel.

The contents of a MCMPO may be similar to those of a CCMPO, except that the MCMPO may include rules applicable to a single work, and may identify CMPOs associated with each object.

The contents of each CMPO may be similar to those of the MCMPO, except that the CMPO may include rules and keys applicable to a single object.

The contents of an SGCMPO may be similar to those of the CCMPO, except that the MCMPO may include rules applicable to only certain one or more classes of rights, certain one or more classes of works, and/or to one or more certain classes of users and/or user arrangements (e.g. CMPO arrangements and/or their devices).

In another embodiment, shown in FIG. 28, CMPO Data Structure 2801 may be defined as follows:

CMPO Data Structure 2801 is made up of elements. Each element includes a self-contained item of information. The CMPS parses CMPO Data Structure, one element at a time.

Type Element 2802 identifies the data structure as a CMPO, thereby allowing the CMPS to distinguish it from a content ES. In an exemplary embodiment, this element may include 4 bits, each of which may be set to "1" to indicate that the data structure is a CMPO.

The second element is CMPO Identifier 2803, which is used to identify this particular CMPO and to convey whether the CMPO is part of a hierarchical organization of CMPOs and, if so, where this CMPO fits into that organization.

CMPO Identifier 2803 is divided into four sub-elements, each of three bits. These are shown as sub-elements A, B, C and D. The first sub-element (2803 A) identifies the CMPO type, and indicates whether the CMPO is governed or controlled by any other CMPO:

100: this is a top-level CMPO (associated with a channel or an aggregation of works) and is not controlled by any other CMPO.

010: this is a mid-level CMPO (associated with a particular work) and is not controlled by any other CMPO.

110: this is a mid-level CMPO, and is controlled by a top-level CMPO.

001: this is a low-level CMPO (associated with an object within a work) and is not controlled by any other CMPO. This case will be rare, since a low-level CMPO will ordinarily be controlled by at least one higher-level CMPO.

011: this is a low-level CMPO, and is controlled by a mid-level CMPO, but not by a top-level CMPO.

111: this is a low-level CMPO, and is controlled by a top-level CMPO and by a mid-level CMPO.

The second sub-element of CMPO ID 2803 (sub-element B) identifies a top-level CMPO. In the case of a top-level CMPO, this identifier is assigned by the creator of the CMPO. In the case of a mid-level or low-level CMPO which is controlled by a top-level

CMPO, this sub-element contains the identification of the top-level CMPO which performs such control. In the case of a mid-level or low-level CMPO which is not controlled by a top-level CMPO, this sub-element contains zeroes.

The third sub-element of CMPO ID 2803 (sub-element C) identifies a mid-level CMPO. In the case of a top-level CMPO, this sub-element contains zeroes. In the case of a mid-level CMPO, this sub-element contains the identification of the particular CMPO. In the case of a low-level CMPO which is controlled by a mid-level CMPO, this sub-element contains the identification of the mid-level CMPO which performs such control. In the case of a low-level CMPO which is not controlled by a mid-level CMPO, this sub-element contains zeroes.

The fourth sub-element of CMPO ID 2803 (sub-element D) identifies a low-level CMPO. In the case of a top-level or mid-level CMPO, this sub-element contains zeroes. In the case of a low-level CMPO, this sub-element contains the identification of the particular CMPO.

Following the identifier element is Size Element 2804 indicating the size of the CMPO data structure. This element contains the number of elements (or bytes) to the final element in the data structure. This element may be rewritten if alterations are made to the CMPO. The CMPS may use this size information to determine whether the element has been altered without permission, since such an alteration might result in a different size. For such purposes, the CMPS may store the information contained in this element in a protected database. This information can also be used to establish that the entire CMPO has been received and is available, prior to any attempt to proceed with processing.

Following Size Element 2804 are one or more Ownership/Control Elements containing ownership and chain of control information (e.g., Ownership/Control Elements 2805, 2806 and 2807). In the first such element (2805), the creator of the CMPO may include a specific identifier associated with that creator. Additional participants may also be identified in following elements (e.g., 2806, 2807). For example, Element 2805 could identify the creator of the CMPO, Element 2806 could identify the publisher of the associated work and Element 2807 could identify the author of the work.

A specific End Element 2808 sequence (e.g., 0000) indicates the end of the chain of ownership elements. If this sequence is encountered in the first element, this indicates that no chain of ownership information is present.

Chain of ownership information can be added, if rules associated with CMPO 2801 permit such additions. If, for example, a user purchases the work associated with CMPO 2801, the user's identification may be added as a new element in the chain of ownership

elements (e.g., a new element following 2807, but before 2808). This may be done at the point of purchase, or may be accomplished by the CMPS once CMPO 2801 is encountered and the CMPS determines that the user has purchased the associated work. In such a case, the CMPS may obtain the user identifier from a data structure stored by the CMPS in NVRAM.

Following the ownership element chain are one or more Handling Elements (e.g., 2809, 2810) indicating chain of handling. These elements may contain the identification of any CMPS which has downloaded and decoded CMPO 2801, and/or may contain the identification of any user associated with any such CMPS. Such information may be used for audit purposes, to allow a trail of handling in the event a work is determined to have been circulated improperly. Such information may also be reported as exhaust to a clearinghouse or central server. Chain of handling information preferably remains persistent until reported. If the number of elements required for such information exceeds a specified amount (e.g., twenty separate user identifiers), a CMPS may refuse to allow any further processing of CMPO 2801 or the associated work until the CMPS has been connected to an external server and has reported the chain of handling information.

The last element in the chain of handling elements (e.g., 2811) indicates the end of this group of elements. The contents of this element may, for example, be all zeroes.

Following the chain of handling elements may be one or more Certificate Elements (e.g., 2812, 2813) containing or pointing to a digital certificate associated with this CMPO. Such a digital certificate may be used by the CMPS to authenticate the CMPO. The final element in the digital certificate chain is all zeroes (2814). If no digital certificate is present, a single element of all zeroes exists in this location.

Following the Certificate Elements may be a set of Governed Object Elements (e.g., 2815, 2816, 2817, 2818) specifying one or more content objects and/or CMPOs which may be governed by or associated with CMPO 2801. Each such governed object or CMPO is identified by a specific identifier and/or by a location where such object or CMPO may be found (e.g., these may be stored in locations 2815 and 2817). Following each such identifier may be one or more keys used to decrypt such CMPO or object (e.g., stored in locations 2816 and 2818). The set of identifiers/keys ends with a termination element made up of all zeroes (2819).

Following the set of elements specifying identifiers and/or keys may be a set of Rules Elements (e.g., 2820, 2821, 2822) specifying rules/controls and conditions associated with use of the content objects and/or CMPOs identified in the Governed Objects chain (e.g., locations 2815 and 2817). Exemplary rules are described below. Elements may

contain explicit rules or may contain pointers to rules stored elsewhere. Conditions may include particular hardware resources necessary to use associated content objects or to satisfy certain rules, or particular types of CMPS's which are necessary or preferred for use of the associated content objects.

Following the rules/controls and conditions elements may be a set of Information Elements 2823 containing information specified by the creator of the CMPO. Among other contents, such information may include content, or pointers to content, programming, or pointers to programming.

The CMPO ends with Final Termination Element 2824.

In one embodiment, the rules contained in Rules Elements 2820-2822 of CMPO 2801 may include, for example, the following operations:

(1) Play. This operation allows the user to play the content (though not to copy it) without restriction.

(2) Navigate. This allows the user to perform certain types of navigation functions, including fast forward/rewind, stop and search. Search may be indexed or unindexed.

(3) Copy. Copy may be allowed once (e.g., time-shifting, archiving), may be allowed for a specified number of times and/or may be allowed for limited period of time, or may be allowed for an unlimited period of time, so long as other rules, including relevant budgets, are not violated or exceeded. A CMPS arrangement may be designed so that a Copy operation may cause an update to an associated CMPO (e.g., including an indication that the associated content has been copied, identifying the date of copying and the site responsible for making the copy), without causing any change to any applicable content object, and in particular without requiring that associated content objects be demuxed, decrypted or decompressed. In the case of MPEG-4, for example, this may require the following multi-stage demux process:

(i) the CMPS arrangement receives a Copy instruction from the user, or from a header CMPO.

(ii) CMPO ESs associated with the MPEG-4 stream which is to be copied are separated from the content stream in a first demux stage.

(iii) CMPOs are decrypted and updated by the CMPS arrangement. The CMPOs are then remuxed with the content ESs (which have never been demuxed from each other), and the entire stream is routed to the output port without further alteration.

This process allows a copy operation to take place without requiring that the content streams be demuxed and decrypted. It requires that the CMPS arrangement include

two outputs: one output connected to the digital output port (e.g., FIG. 23 line 2316, connecting to Digital Output Port 2317), and one output connected to the MPEG-4 buffers (e.g., FIG. 23, lines 2310, 2311, 2312), with a switch designed to send content to one output or the other (or to both, if content is to be viewed and copied simultaneously) (e.g., Switch 2319). Switch 2319 can be the only path to Digital Output Port 2317, thereby allowing CMPS 2302 to exercise direct control over that port, and to ensure that content is never sent to that port unless authorized by a control. If Digital Output Port 2317 is also the connector to a digital display device, CMPS 2302 will also have to authorize content to be sent to that port even if no copy operation has been authorized.

In one example embodiment, the receiving device receiving the information through Digital Output Port 2317 may have to authenticate with the sending device (e.g., CMPS 2302). Authentication may be for any characteristic of the device and/or one or more CMPSs used in conjunction with that device. Thus, for example, a sending appliance may not transmit content to a storage device lacking a compatible CMPS.

In another non-limiting example, CMPS 2302 can incorporate session encryption functionality (e.g., the "five company arrangement") which establishes a secure channel from a sending interface to one or more external device interfaces (e.g., a digital monitor), and provided that the receiving interface has authenticated with the sending interface, encrypts the content so that it can only be decrypted by one or more authenticated 1394 device interfaces. In that case, CMPS 2302 would check for a suitable IEEE 1394 serial bus interface , and would allow content to flow to Digital Output Port 2317 only if (a) an authorized Play operation has been invoked, a secure channel has been established with the device and the content has been session-encrypted, or (b) an authorized Copy or Retransmit operation has been invoked, and the content has been treated as per the above description (i.e., the CMPO has been demuxed, changed and remuxed, the content has never been decrypted or demuxed).

This is only possible if CMPOs are separately identifiable at an early demux stage, which most likely requires that they be stored in separate CMPO ESs. If the CMPOs are stored as headers in content ESs, it may be impossible to identify the CMPOs prior to a full demux and decrypt operation on the entirety of the stream.

(4) Change. The user may be authorized to change the content.

(5) Delete. This command allows the user to delete content which is stored in the memory of the Consumer Appliance. This operation operates on the entire work. If the user wishes to delete a portion of a work, the Change operation must be used.

(6) Transfer. A user may be authorized to transfer a work to a third party.

This differs from the Copy operation in that the user does not retain the content or any rights to the content. The Transfer operation may be carried out by combining a Copy operation and a Delete operation. Transfer may require alteration of the header CMPO associated with the work (e.g., adding or altering an Ownership/Control Element, such as Elements 2805-2807 of FIG. 28), so as to associate rights to the work with the third party.

These basic operations may be subject to modifications, which may include:

i. Payment. Operations may be conditioned on some type of user payment. Payment can take the form of cash payment to a provider (e.g., credit card, subtraction from a budget), or sending specified information to an external site (e.g., Nielson-type information).

ii. Quality of Service. Operations may specify particular quality of service parameters (e.g., by specifying a requested QoS in MPEG-4), including: requested level of decompression, requested/required types of display, rendering devices (e.g., higher quality loudspeakers, a particular type of game controller).

iii. Time. Operations may be conditioned such that the operation is only allowed after a particular time, or such that the price for the operation is tied to the time (e.g., real-time information at a price, delayed information at a lower price or free, e.g., allowing controlled copies but only after a particular date).

iv. Display of particular types of content. Operations may be conditioned on the user authorizing display of certain content (e.g., the play operation may be free if the user agrees to allow advertisements to be displayed).

In all of these cases, a rule may be modified by one or more other rules. A rule may specify that it can be modified by other rules or may specify that it is unmodifiable. If a rule is modifiable, it may be modified by rules sent from other sources. Those rules may be received separately by the user or may be aggregated and received together by the user.

Data types which may be used in an exemplary MPEG-4 embodiment may include the following:

a. CMP Data Stream.

The CMP-ds is a new elementary stream type that has all of the properties of an elementary stream including its own CMPO and a reference in the object descriptors. Each CMP-ds stream has a series of one or more *CMP Messages*. A CMP_Message has four parts:

1. Count: [1...*n*] CMPS types supported by this IP ES. Multiple CMPS systems may be supported, each identified by a unique *type*. (There may have

to be a central registry of types.)

2. CMPS_type_identifiers: [1...*n*] identifiers, each with an offset in the stream and a length. The offset points to the byte in the CMPO where the data for that CMPS type is found. The length is the length in bytes of this data.

3. Data segments: One segment for each of the *n* CMPS types encoded in a format that is proprietary to the CMPS supplier.

4. CMP_Message_URL: That references another CMP_Message. (This is in keeping with the standard of using URLs to point to streams.)

b. CMPO.

The CMPO is a data structure used to attach detailed CMP control to individual elementary streams. Each **CMPO** contains:

1. CMPO_ID: An identifier for the content under control. This identifier must *uniquely* identify an elementary stream.

2. CMPO_count: [1...*n*] CMPS types supported by this **CMPO**.

3. CMPS_type_identifiers: [1...*n*] identifiers, each with an offset in the stream and a length. The offset points to the byte in the CMPO where the data for that CMPS type is found. The length is the length in bytes of this data.

4. Data segments: *n* data segments. Each data segment is in a format that is proprietary to the CMPS supplier.

5. CMPO_URL: An optional URL that references an additional CMPO that adds information to the information in this CMPO. (This is a way of dynamically adding support for new CMPSs.)

c. Feedback Event

The feedback events come in two forms: start and end. Each feedback event contains three pieces of information:

1. Elementary_stream_ID

2. Time: in presentation time

3. Object_instance_number

User Interface.

Commerce Appliance 2301 may include User Interface 2304 designed to convey control-related information to the user and to receive commands and information from the user. This interface may include special purpose displays (e.g., a light which comes on if a current action requires payment), special purpose buttons (e.g., a button which accepts the payment or other terms required for display of content), and/or visual information presented on screen.

Example of Operation in an MPEG-4 Context

1. User selects a particular work or channel. The user may, for example, use a remote control device to tune a digital TV to a particular channel.

2. Selection of the channel is communicated to a CMPS arrangement, which uses the information to either download a CCMPO or to identify a previously downloaded CCMPO (e.g., if the CMPS arrangement is contained in a set-top box, the set-top box may automatically download CCMPOs for every channel potentially reachable by the box).

3. The CMPS arrangement uses the CCMPO to identify rules associated with all content found on the channel. For example, the CCMPO may specify that content may only be viewed by subscribers, and may specify that, if the user is not a subscriber, an advertisement screen should be put up inviting the user to subscribe.

4. Once rules specified by the CCMPO have been satisfied, the CCMPO specifies the location of a MCMPO associated with a particular work which is available on the channel. The channel CMPO may also supply one or more keys used for decryption of the MCMPO.

5. The CMPS arrangement downloads the MCMPO. In the case of an MPEG-4 embodiment, the MCMPO may be an Elementary Stream. This Elementary Stream must be identifiable at a relatively early stage in the MPEG-4 decoding process.

6. The CMPS arrangement decrypts the MCMPO, and determines the rules used to access and use the content. The CMPS arrangement presents the user with a set of options, including the ability to view for free with advertisements, or to view for a price without advertisements.

7. The user selects view for free with advertisements, e.g., by highlighting and selecting an option on the screen using a remote control device.

8. The CMPS arrangement acquires one or more keys from the MCMPO and uses those keys to decrypt the ESs associated with the video. The CMPS arrangement identifies two possible scene descriptor graphs, one with and one without advertisements. The CMPS arrangement passes the scene descriptor graph with advertisements through, and blocks the other scene descriptor graph.

9. The CMPS arrangement monitors the composite and render block, and checks to determine that the advertisement AVOs have actually been released for viewing. If the CMPS arrangement determines that those AVOs have not been released for viewing, it puts up an error or warning message, and terminates further decryption.

CMPS Rights Management In Provider And Distribution Chains

In addition to consumer arrangements, in other embodiments one or more CMPSs

may be used in creating, capturing, modifying, augmenting, animating, editing, excerpting, extracting, embedding, enhancing, correcting, fingerprinting, watermarking, and/or rendering digital information to associate rules with digital information and to enforce those rules throughout creation, production, distribution, display and/or performance processes.

In one non-limiting example, a CMPS, a non-exhaustive example of which may include a least a secure portion of a VDE node as described in the aforementioned Ginter et al., patent specification, is incorporated in video and digital cameras, audio microphones, recording, playback, editing, and/or noise reduction devices and/or any other digital device. Images, video, and/or audio, or any other relevant digital information may be captured, recorded, and persistently protected using at least one CMPS and/or at least one CMPO. CMPSs may interact with compression/decompression, encryption/decryption, DSP, digital to analog, analog to digital, and communications hardware and/or software components of these devices as well.

In another non-exhaustive example, computer animation, special effects, digital editing, color correcting, noise reduction, and any other applications that create and/or use digital information may protect and/or manage rights associated with digital information using at least one CMPS and/or at least one CMPO.

Another example includes the use of CMPSs and/or CMPOs to manage digital assets in at least one digital library, asset store, film and/or audio libraries, digital vaults, and/or any other digital content storage and management means.

In accordance with the present applications, CMPSs and/or CMPOs may be used to manage rights in conjunction with the public display and/or performance of digital works. In one non-exhaustive example, flat panel screens, displays, monitors, TV projectors, LCD projectors, and/or any other means of displaying digital information, may incorporate at least one hardware and/or software CMPS instance that controls the use of digital works. A CMPS may allow use only in conjunction with one or more digital credentials, one example of which is a digital certificate, that warrant that use of the digital information will occur in a setting, location, and/or other context for public display and/or performance. Non-limiting examples of said contexts include theaters, bars, clubs, electronic billboards, electronic displays in public areas, or TVs in airplanes, ships, trains and/or other public conveyances. These credentials may be issued by trusted third parties such as certifying authorities, non-exhaustive examples of which are disclosed in the aforementioned Ginter '712 patent application.

Additional MPEG-4 Embodiment Information

This work is based on the MPEG-4 description in the version 1 Systems Committee Draft (CD), currently the most complete description of the evolving MPEG-4 standard.

This section presents the structural modifications to the MPEG-4 player architecture and discusses the data lines and the concomitant functional changes. Figure 23 shows the functional components of the original MPEG-4 player. Content arrives at Player 2301 packaged into a serial stream (e.g., MPEG-4 Bit Stream 2314). It is demultiplexed via a sequence of three demultiplexing stages (e.g., Demux 2305) into elementary streams. There are three principle types of elementary streams: AV Objects (AVO), Scene Descriptor Graph (SDG), and Object Descriptor (OD). These streams are fed into respective processing elements (e.g., AVO Decode 2307, Scene Descriptor Graph 2306, Object Descriptors 2308). The AVOs are the multimedia content streams such as audio, video, synthetic graphics and so on. They are processed by the player's compression/coding subsystems. The scene descriptor graph stream is used to build the scene descriptor graph. This tells Composite and Render 2309 how to construct the scene and can be thought of as the "script." The object descriptors contain description information about the AVOs and the SD-graph updates.

To accommodate a CMPS (e.g., CMPS 2302) and to protect content effectively, the player structure must be modified in several ways:

- Certain data paths must be rerouted to and from the CMPS
- Certain buffers in the SDG, AVO decode and Object descriptor modules must be secured
- Feedback paths from the user and the composite and render units to the CMPS must be added

In order for CMPS 2302 to communicate with the MPEG-4 unit, and for it to effectively manage content we must specify the CMPO structure and association protocols and we must define the communication protocols over the feedback systems (from the compositor and the user.)

The structural modifications to the player are shown in Figure 23. The principal changes are:

- All elementary streams are now routed through CMPS 2302.
- Direct communication path between Demux 2305 and CMPS 2302.
- A required "Content Release and Decrypt" Module 2315 in CMPS 2302.

- The addition of a feedback loop (e.g., Line 2313) from Composite and Render 2309 to CMPS 2302.
- Bi-directional user interaction directly with the CMPS 2302, through Line 2316.

Furthermore, for M4v2P, CMP-objects are preferably associated with all elementary streams. Elementary streams that the author chooses not to protect are still marked by an "unprotected content" CMPO. The CMPOs are the primary means of attaching rules information to the content. Content here not only refers to AVOs, but also to the scene descriptor graph. Scene Descriptor Graph may have great value and will thus need to be protected and managed by CMPS 2302.

The direct path from Demux 2305 to CMPS 2302 is used to pass a CMPS specific header, that potentially contains business model information, that communicates business model information at the beginning of user session. This header can be used to initiate user identification and authentication, communicate rules and consequences, and initiate up-front interaction with the rules (selection of quality-of-service (QoS), billing, etc.) The user's communication with CMPS 2302 is conducted through a *non-standardized* channel (e.g., Line 2316). The CMPS designer may provide an independent API for framing these interactions.

Feedback Path 2313 from Composite and Render block 2309 serves an important purpose. The path is used to cross check that the system actually presented the user with a given scene. Elementary streams that are processed by their respective modules may not necessarily be presented to the user. Furthermore, there are several fraud scenarios wherein an attacker could pay once and view multiple times. The feedback path here allows CMPS 2302 to cross check the rendering and thereby perform a more accurate accounting. This feedback is implemented by forcing the Composite and Render block 2309 to issue a *start event* that signals the initiation of a given object's rendering that is complemented by a *stop event* upon termination. The feedback signaling process may be made optional by providing a CMP-notification flag that may be toggled to indicate whether or not CMPS 2302 should be notified. All CMPOs would be required to carry this flag.

The final modification to the structure is to require that the clear text buffers in the AVO, SDG and Object Descriptor processors and in the Composite-and-Render block be secured. This is to prevent a pirate from stealing content in these buffers. As a practical matter, this may be difficult, since tampering with these structures may well destroy synchronization of the streams. However, a higher state of security would come from placing these buffers into a protected processing environment.

CMPS 2302 *governs* the functioning of Player 2301, consistent with the following:

- Communication mechanism between CMPS 2302 and the MPEG-4 player (via CMPOs)
- A content release and decryption subsystem
- Version authentication subsystem
- Sufficient performance so as not to interfere with the stream processing in the MPEG-4 components

CMPS 2302 may have a bi-directional side-channel that is external to the MPEG-4 player that may also be used for the exchange of CMP information. Furthermore, the CMPS designer may choose to provide a user interface API that provides the user with the ability to communicate with the content and rights management side of the stream management (e.g., through Line 2316).

Encrypted content is decrypted and released by CMPS 2302 as a function of the rules associated with the protected content and the results of user interaction with CMPS 2302. Unencrypted content is passed through CMPS 2302 and is governed by associated rules and user interaction with CMPS 2302. As a consequence of these rules and user interaction, CMPS 2302 may need to transact with the SDG and AVO coding modules (e.g., 2310, 2311) to change scene structure and/or the QoS grade.

Ultimately, the CMPS designer may choose to have CMPS 2302 generate audit trail information that may be sent to a clearinghouse authority via CMPS Side Channel Port 2318 or as encrypted content that is packaged in the MPEG-4 bit stream.

The MPEG-4 v1 Systems CD uses the term "object" loosely. In this document, "object" is used to specifically mean a data structure that flows from one or more of the data paths in Figure 23.

Using multiple SD-graph update streams, each with its own CMPO, allows an author to apply arbitrarily specific controls to the SD-graph. For example, each node in the SD-graph can be created or modified by a separate SD-graph update stream. Each of these streams will have a distinct CMPO and ID. Thus, the CMPS can release and decrypt the creation and modification of each node and receive feedback information for each node individually. The practical implications for controlling release and implementing consequences should be comparable to having a CMPO on each node of the SD-graph, without the costs of having a CMPO on each SD-graph node.

Principles consistent with the present invention may be illustrated using the following examples:

In the first example, there is a bilingual video with either an English or French soundtrack. The user can choose during playback to hear either the English or French. The

basic presentation costs $1. If the French soundtrack is presented there is a $0.50 surcharge. If the user switches back and forth between French and English, during a single viewing of the presentation, the $0.50 surcharge will occur only once.

In this example, there will be four elementary streams:

The Scene Description Graph Update stream will have a CMPO. The CMPO will imply a $1.00 fee associated with the use of the content. The scene description graph displays the video, English audio and puts up a button that allows the user to switch to French. If the user clicks that button, the English stops, the French picks up from that point and the button changes to a switch-to-English button. (Optionally, there may be a little dialog at the beginning to allow the user to select the initial language. This is all easy to do in the SD graph.)

The Video Stream with the CMPO will say that it can only be released if the scene description graph update stream above is released.

The English Audio Stream will be similar to the Video stream.

The French Audio Stream will be similar to the Video stream but there is a $.50 charge it if is seen in the feedback channel. (The CMPS must to not count twice if the user switches between the two in a single play of the presentation.)

An important requirement is that the ID for the SD-graph update stream appears in the feedback path (e.g., Feedback Path 2313). This is so CMPS 2302 knows when the presentation stops and ends so that CMPS 2302 can correctly bill for the French audio.

The rules governing the release of the video and audio streams may include some variations. The rules for these streams, for example, may state something like "if you don't see the id for the scene description graph update stream X in the feedback channel, halt release of this stream." If the main presentation is not on the display, then the video should not be. This ties the video to this one presentation. Using the video in some other presentation would require access to the original video, not just this protected version of it.

In a second example, an author wants to have a presentation with a free attract sequence or "trailer". If the user clicks the correct button the system moves into the for-fee presentation, which is organized as a set of "acts".

Multiple SD-graph update streams may update a scene description graph. Multiple SD-graph update streams may be open in parallel. The time stamps on the ALUs in the streams are used to synchronize and coordinate.

The trailer and each act are represented by a separate SD-graph update stream with a separate CMPO. There is likely an additional SD-graph update stream that creates a simple

root node that is invisible and silent. This node brings in the other components of the presentation as needed.

The foregoing description of implementations of the invention has been presented for purposes of illustration and description. It is not exhaustive and does not limit the invention to the precise form disclosed. Modifications and variations are possible in light of the above teachings or may be acquired from practicing of the invention. For example, the described implementation includes software but the present invention may be implemented as a combination of hardware and software or in hardware alone. The invention may be implemented with both object-oriented and non-object-oriented programming systems. The scope of the invention is defined by the claims and their equivalents.

We claim:

1. A streaming media player providing content protection and digital rights management, including:

a port configured to receive a digital bit stream, the digital bit stream including:

content which is encrypted at least in part, and

a secure container including control information for controlling use of the content, including at least one key suitable for decryption of at least a portion of the content; and

a control arrangement including:

means for opening secure containers and extracting cryptographic keys, and

means for decrypting the encrypted portion of the content.

2. The player of Claim 1 in which the digital bit stream includes at least two sub-streams which have been muxed together, at least one of the sub-streams including compressed information, and

wherein the player further includes:

a demux designed to separate and route the sub-streams;

a decompression unit configured to decompress at least one of the sub-streams, the decompression unit and the demux being connected by a pathway for the transmission of information; and

a rendering unit designed to process decompressed content information for rendering.

3. The player of Claim 2, further including:

a stream controller operatively connected to the decompression unit, the stream controller including decryption functionality configured to decrypt at least a portion of a sub-stream and pass the decrypted sub-stream to the decompression unit.

4. The player of Claim 3, further including:

a path between the control arrangement and the stream controller to enable the control arrangement to pass at least one key to the stream controller for use with the stream controller's decryption functionality.

5. The player of Claim 4, further including:

a feedback path from the rendering unit to the control arrangement to allow the control arrangement to receive information from the rendering unit regarding the identification of objects which are to be rendered or have been rendered.

6. The player of Claim 1, wherein the digital bit stream is encoded in MPEG-4 format.

7. The player of Claim 1, wherein the digital bit stream is encoded in MP3 format.

8. The player of Claim 4, wherein the control arrangement contains a rule or rule set associated with governance of at least one sub-stream or object.

9. The player of Claim 8, wherein the rule or rule set is delivered from an external source.

10. The player of Claim 9, wherein the rule or rule set is delivered as part of the digital bit stream.

11. The player of Claim 8, wherein the rule or rule set specifies conditions under which the governed sub-stream or object may be decrypted.

12. The player of Claim 8, wherein the rule or rule set governs at least one aspect of access to or use of the governed sub-stream or object.

13. The player of Claim 12, wherein the governed aspect includes making copies of the governed sub-stream or object.

14. The player of Claim 12, wherein the governed aspect includes transmitting the governed sub-stream or object through a digital output port.

15. The player of Claim 14, wherein the rule or rule set specifies that the governed sub-stream or object can be transferred to a second device, but rendering of the governed sub-stream or object must be disabled in the first device prior to or during the transfer.

16. The player of Claim 15, wherein the second device includes rendering capability, lacks at least one feature present in the streaming media player, and is at least somewhat more portable than the streaming media player.

17. The player of Claim 11, wherein the control arrangement contains at least two rules governing access to or use of the same governed sub-stream or object.

18. The player of Claim 17, wherein a first of the two rules was supplied by a first entity, and the second of the two rules was supplied by a second entity.

19. The player of Claim 18, wherein the first rule controls at least one aspect of operation of the second rule.

20. The player of Claim 12, wherein the governed aspect includes use of at least one budget.

21. The player of Claim 12, wherein the governed aspect includes a requirement that audit information be provided.

22. The player of Claim 1, wherein the control arrangement includes tamper resistance.

23. A digital bit stream including:

content information that is compressed and at least in part encrypted; and

a secure container including

governance information for the governance of at least one aspect of access to or use of at least a portion of the content information; and

a key for decryption of at least a portion of the encrypted content information.

24. The digital bit stream of Claim 23, wherein the content information is encoded in MPEG-4 format.

25. The digital bit stream of Claim 23, wherein the content information is encoded in MP3 format.

26. A method of rendering a protected digital bit stream including:

receiving the protected digital bit stream,

passing the protected digital bit stream to a media player,

the media player reading first header information identifying a plugin used to process the protected digital bit stream, the first header information indicating that a first plugin is required;

the media player calling the first plugin;

the media player passing the protected digital bit stream to the first plugin;

the first plugin decrypting at least a portion of the protected digital bit stream;

the first plugin reading second header information identifying a second plugin necessary in order to render the decrypted digital bit stream;

the first plugin calling the second plugin;

the first plugin passing the decrypted digital bit stream to the second plugin;

the second plugin processing the decrypted digital bit stream, the processing including decompressing at least a portion of the decrypted digital bit stream;

the second plugin passing the decrypted and processed digital bit stream to the media player; and

the media player enabling rendering of the decrypted and processed digital bit stream,

whereby the first plugin may be used in an architecture not designed for multiple stages of plugin processing.

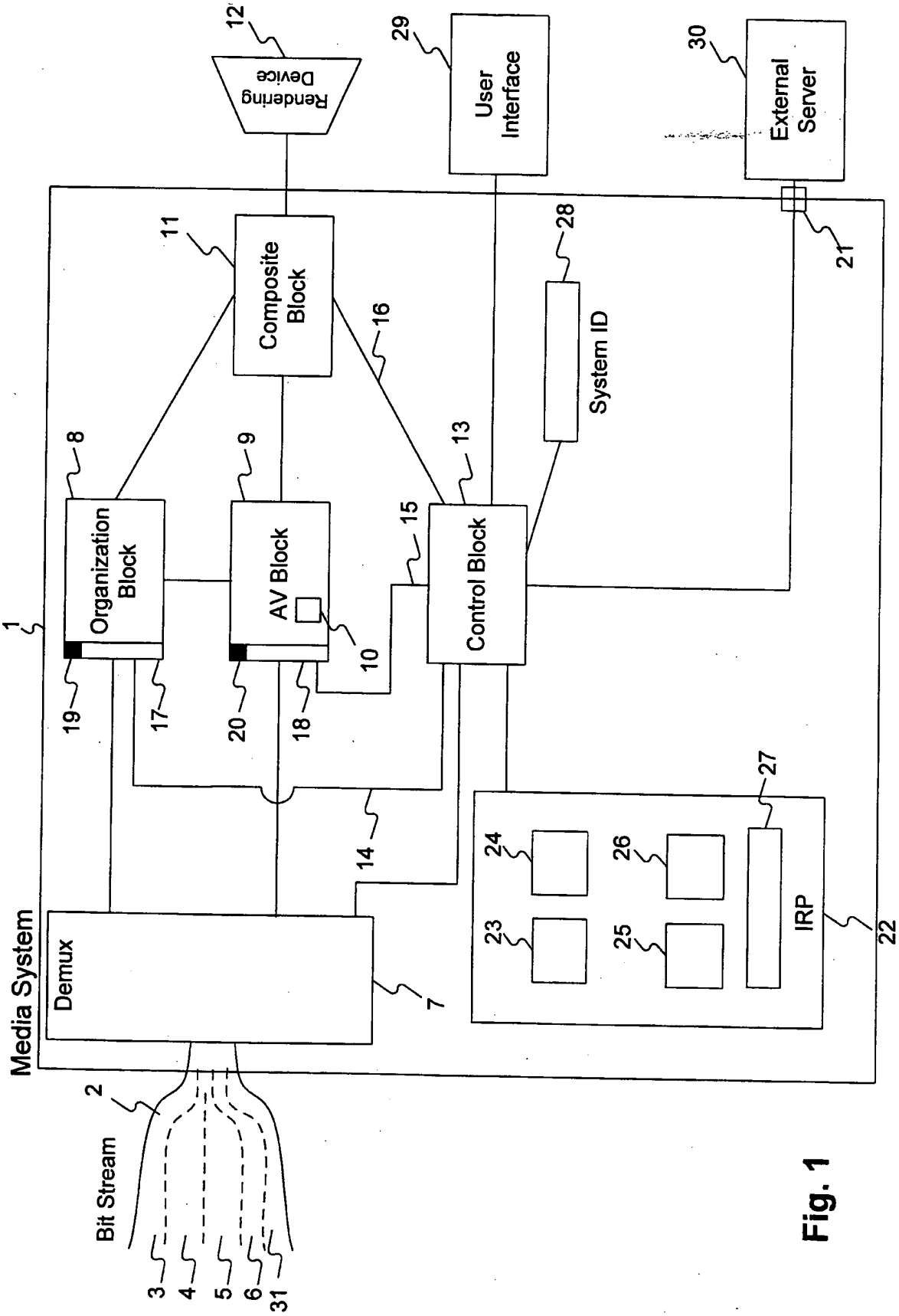

Media System
1
Rendering Device
12
User Interface
29
External Server
30
21
Composite Block
11
16
System ID
28
Organization Block
8
AV Block
9
10
Control Block
13
15
19
17
20
18
14
Demux
7
24
26
27
23
25
IRP
22
Bit Stream
2
3
4
5
6
31

Fig. 1

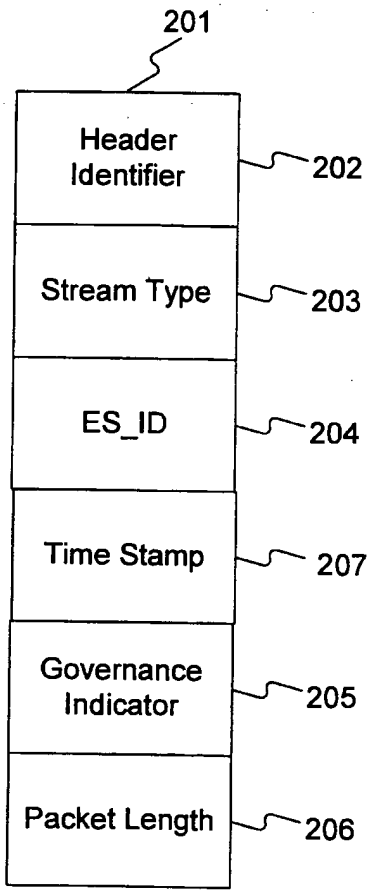

201
Header Identifier
202
Stream Type
203
ES_ID
204
Time Stamp
207
Governance Indicator
205
Packet Length
206

Fig. 2

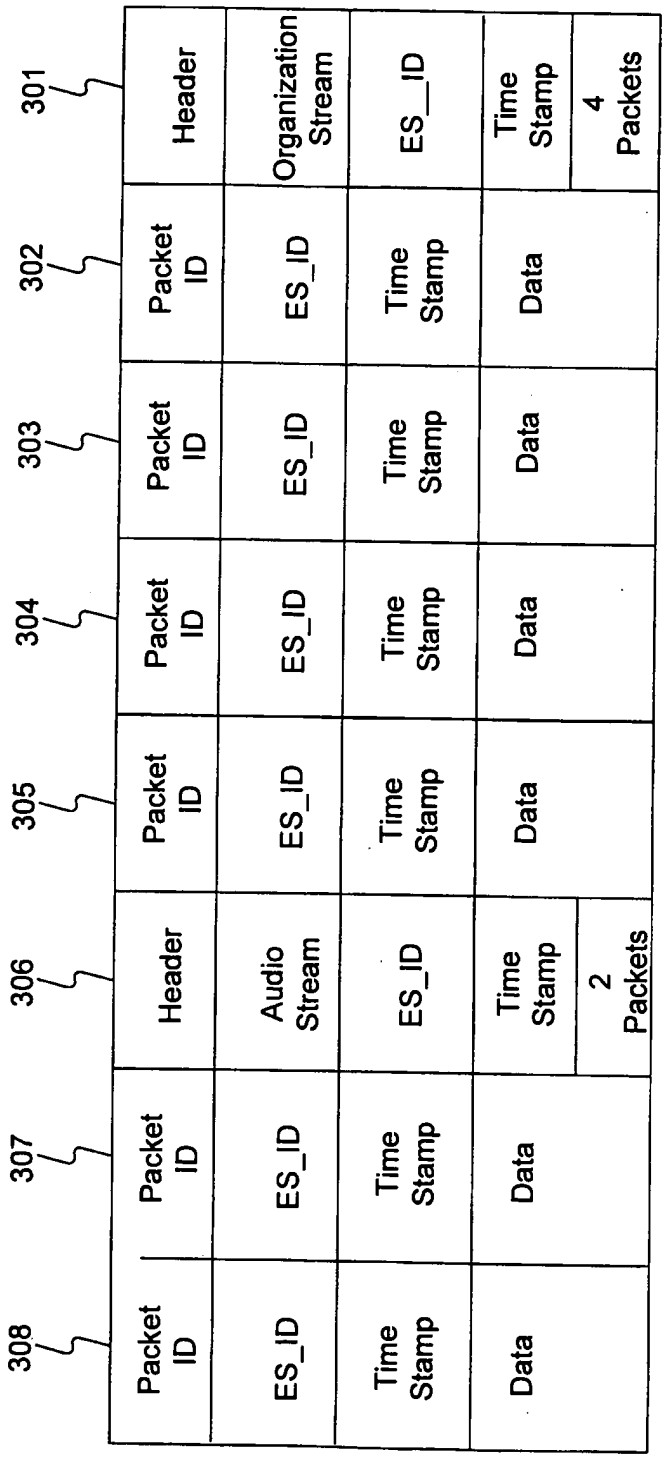

301
302
303
304
305
306
307
308
Header
Packet ID
Packet ID
Packet ID
Packet ID
Packet ID
Header
Packet ID
Packet ID
ES_ID
Time Stamp
Data
Organization Stream
ES_ID
Time Stamp
4 Packets
Audio Stream
ES_ID
Time Stamp
2 Packets

Fig. 3

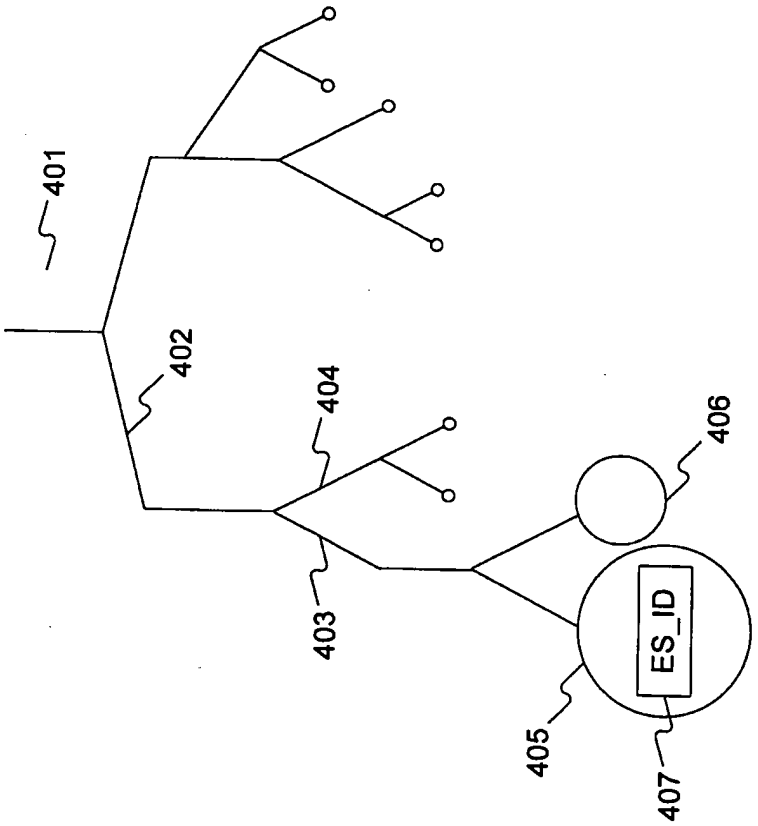


Fig. 4

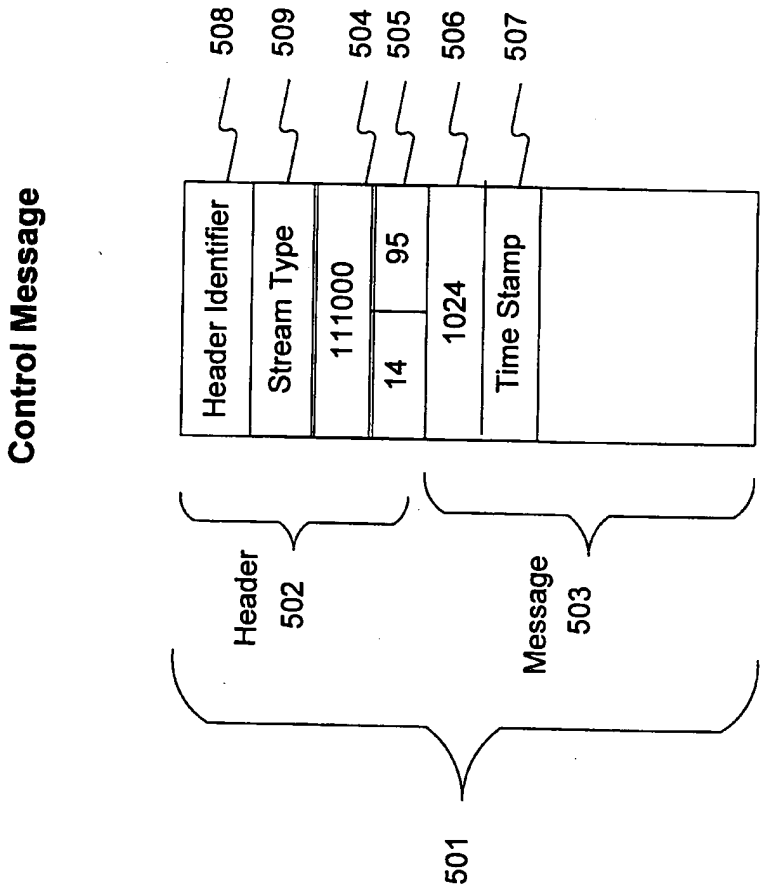

Control Message
508
509
504
505
506
507
Header Identifier
Stream Type
111000
95
14
1024
Time Stamp
Header
502
Message
503
501

Fig. 5

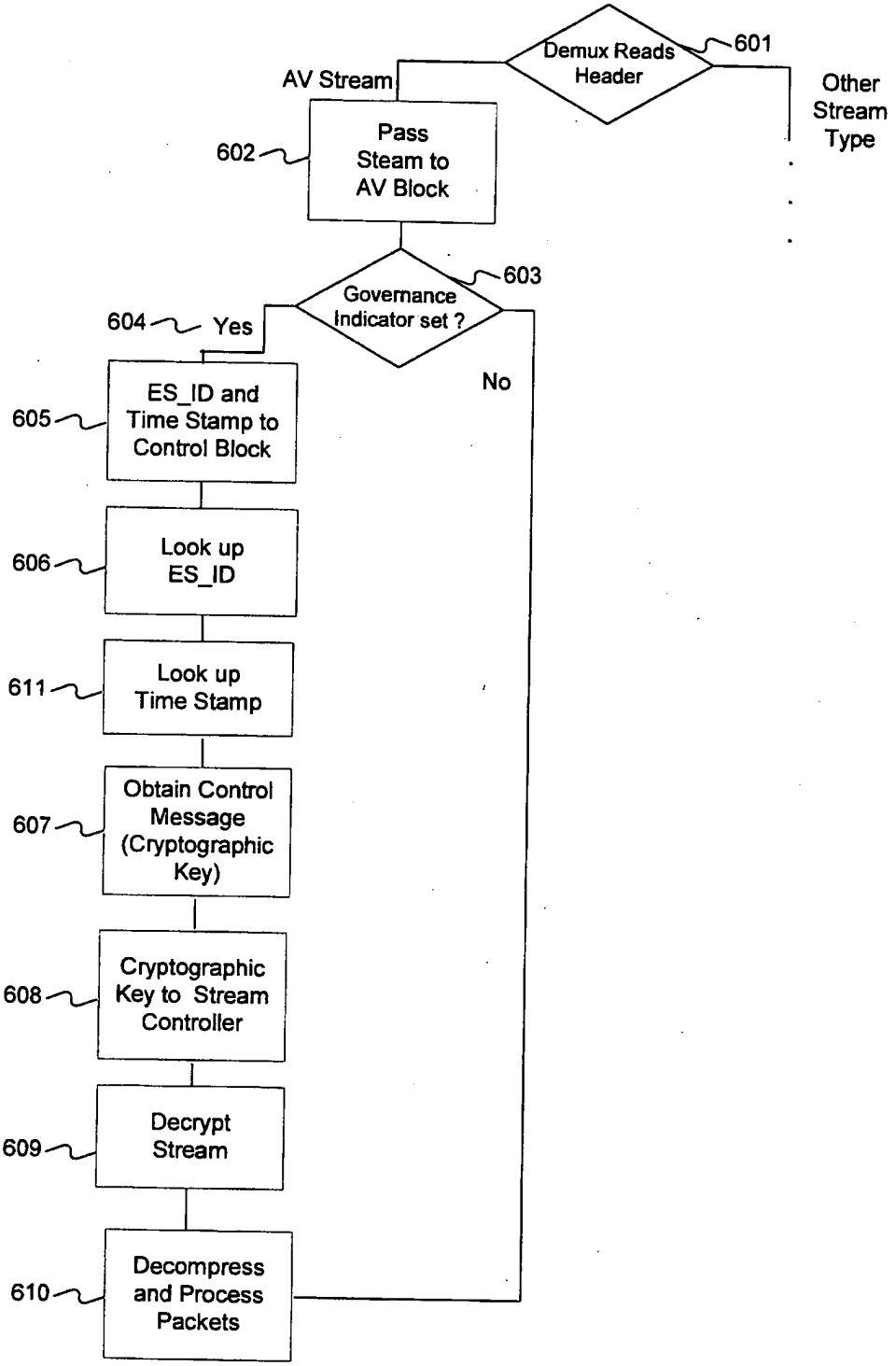

Demux Reads Header
601
AV Stream
Other Stream Type
602
Pass Steam to AV Block
603
Governance Indicator set ?
604
Yes
No
605
ES_ID and Time Stamp to Control Block
606
Look up ES_ID
611
Look up Time Stamp
607
Obtain Control Message (Cryptographic Key)
608
Cryptographic Key to Stream Controller
609
Decrypt Stream
610
Decompress and Process Packets

Fig. 6

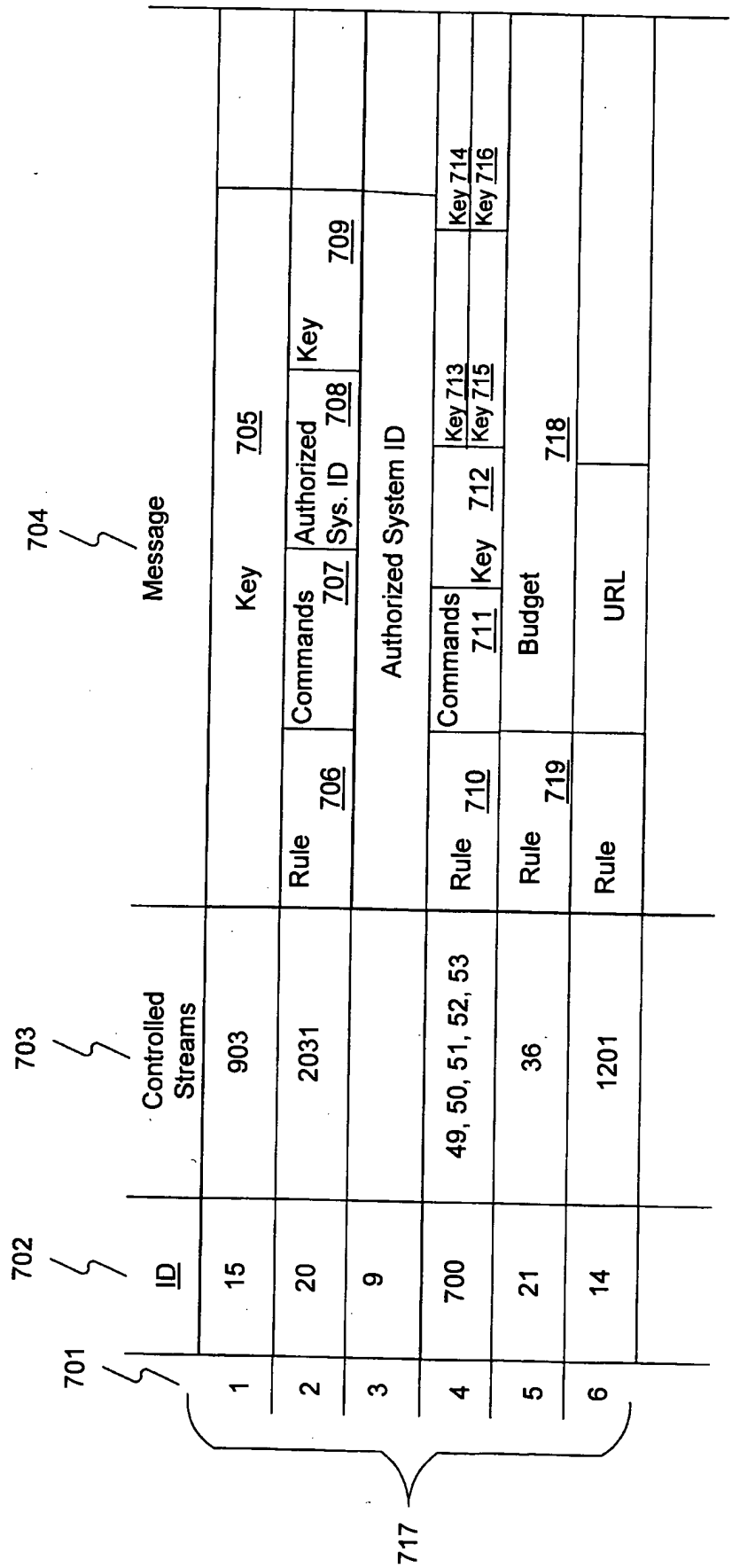

701
702
703
704
717
ID
Controlled Streams
Message
Key 705
Rule 706
Commands 707
Authorized Sys. ID 708
Key 709
Authorized System ID
Commands 711
Key 712
Key 713
Key 715
Key 714
Key 716
Rule 710
Budget 718
Rule 719
URL
Rule
1
2
3
4
5
6
15
20
9
700
21
14
903
2031
49, 50, 51, 52, 53
36
1201

Fig. 7

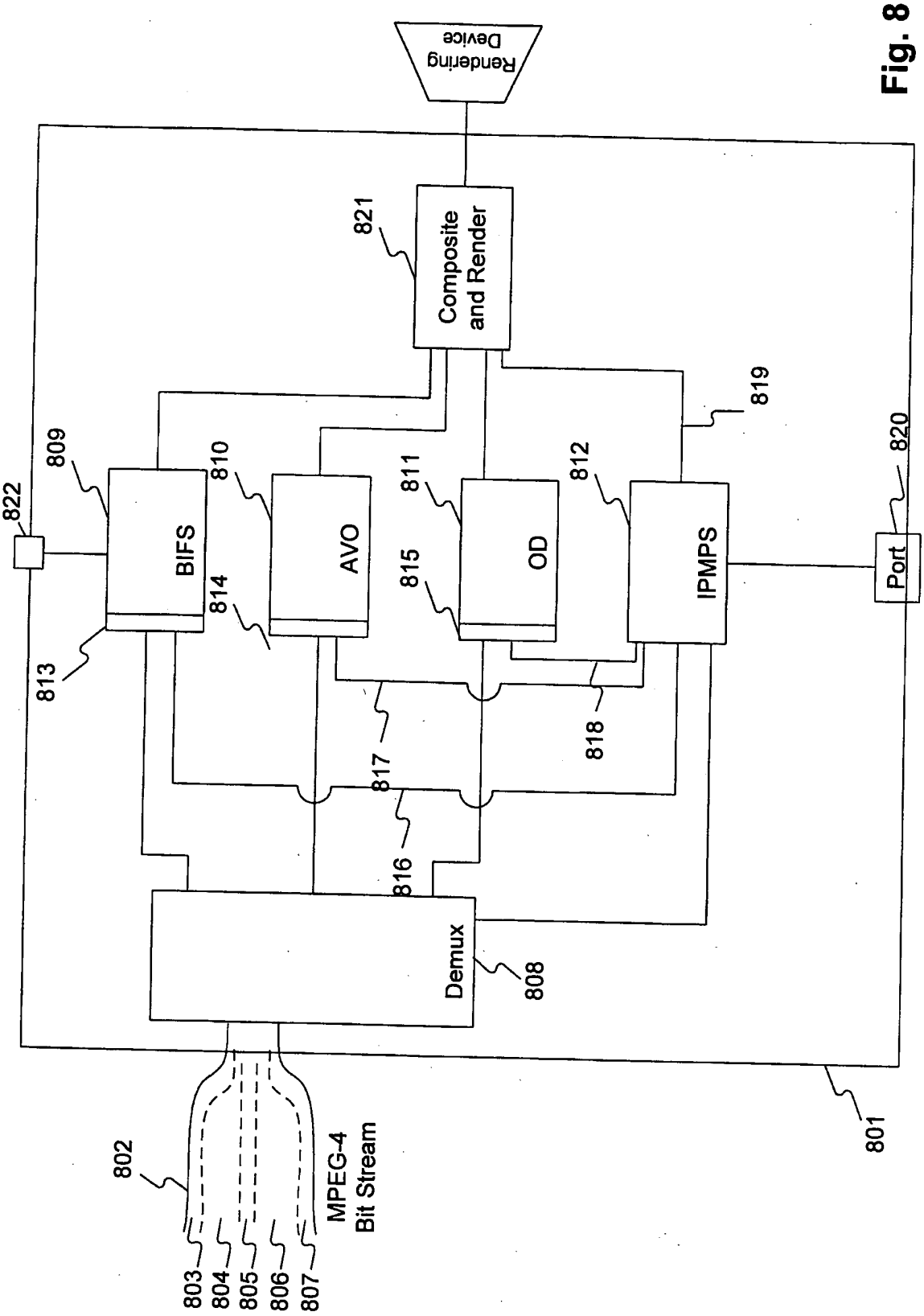

Rendering Device
Composite and Render
821
819
820
809
810
811
812
822
BIFS
AVO
OD
IPMPS
Port
814
815
813
817
818
816
Demux
808
801
802
MPEG-4 Bit Stream
803
804
805
806
807

Fig. 8

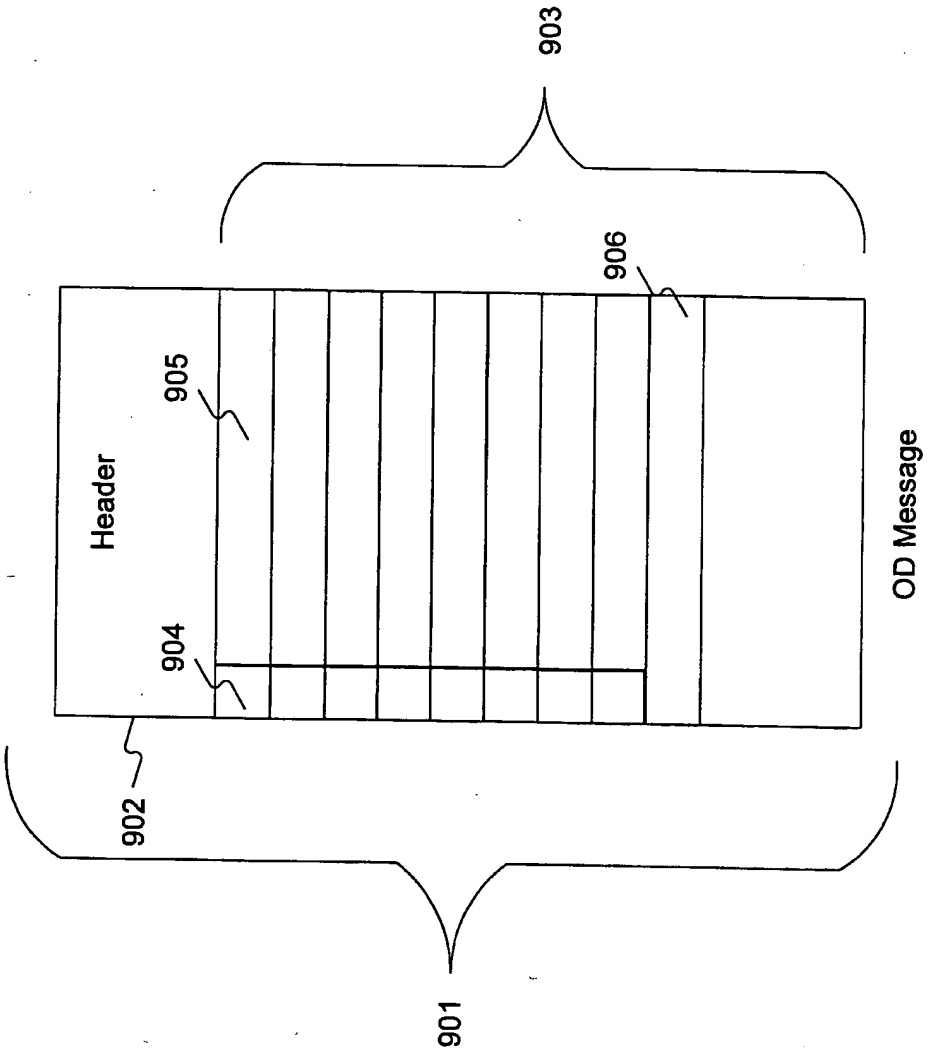


Fig. 9

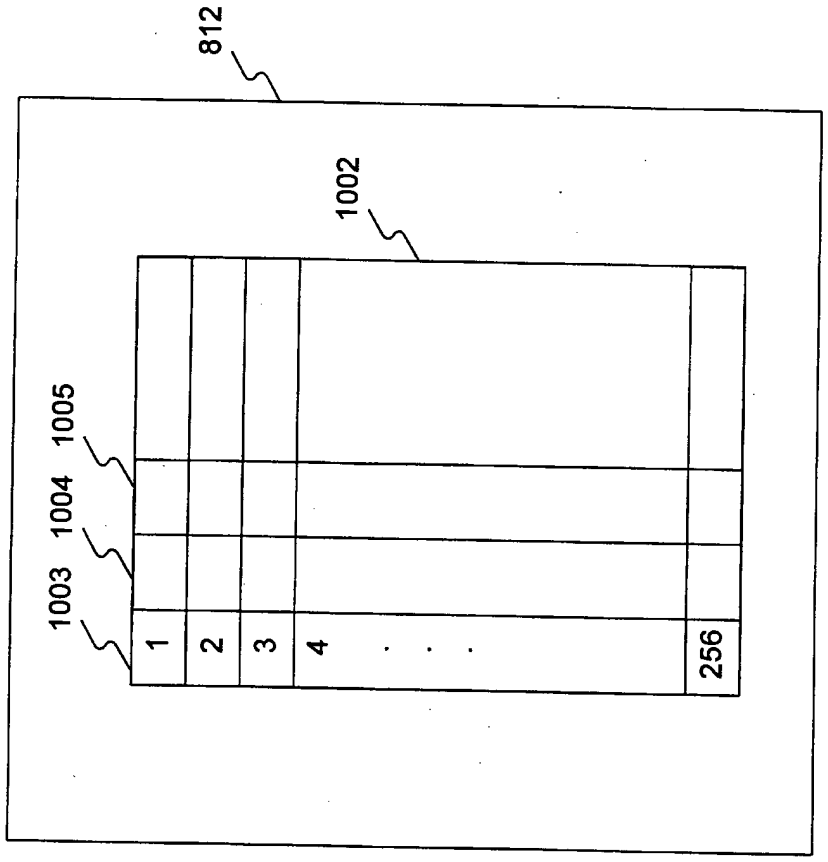


Fig. 10

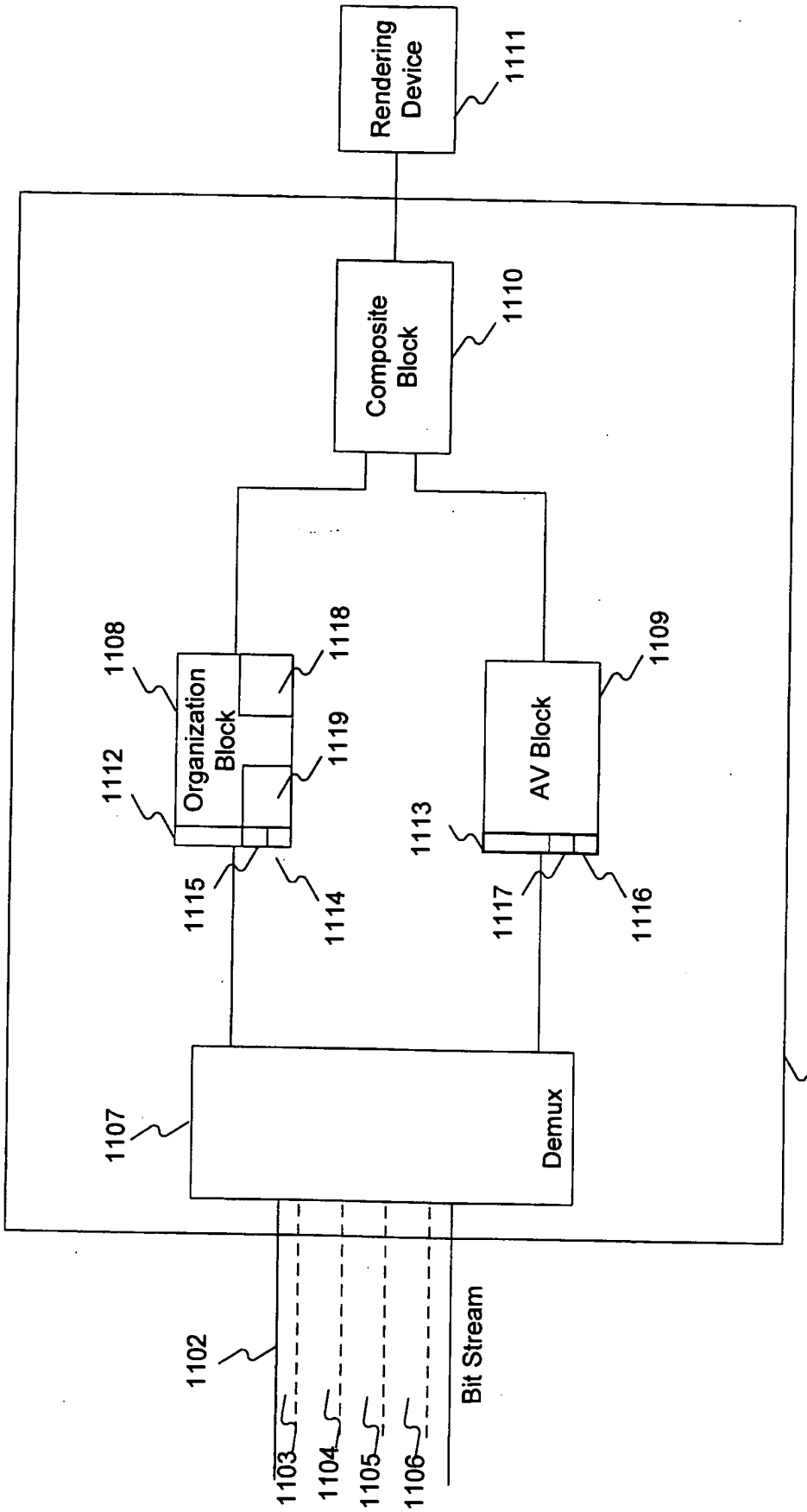

Rendering Device
1111
Composite Block
1110
Organization Block
1108
1118
1119
1112
1115
1114
AV Block
1109
1113
1117
1116
Demux
1107
1101
1102
Bit Stream
1103
1104
1105
1106

Fig. 11

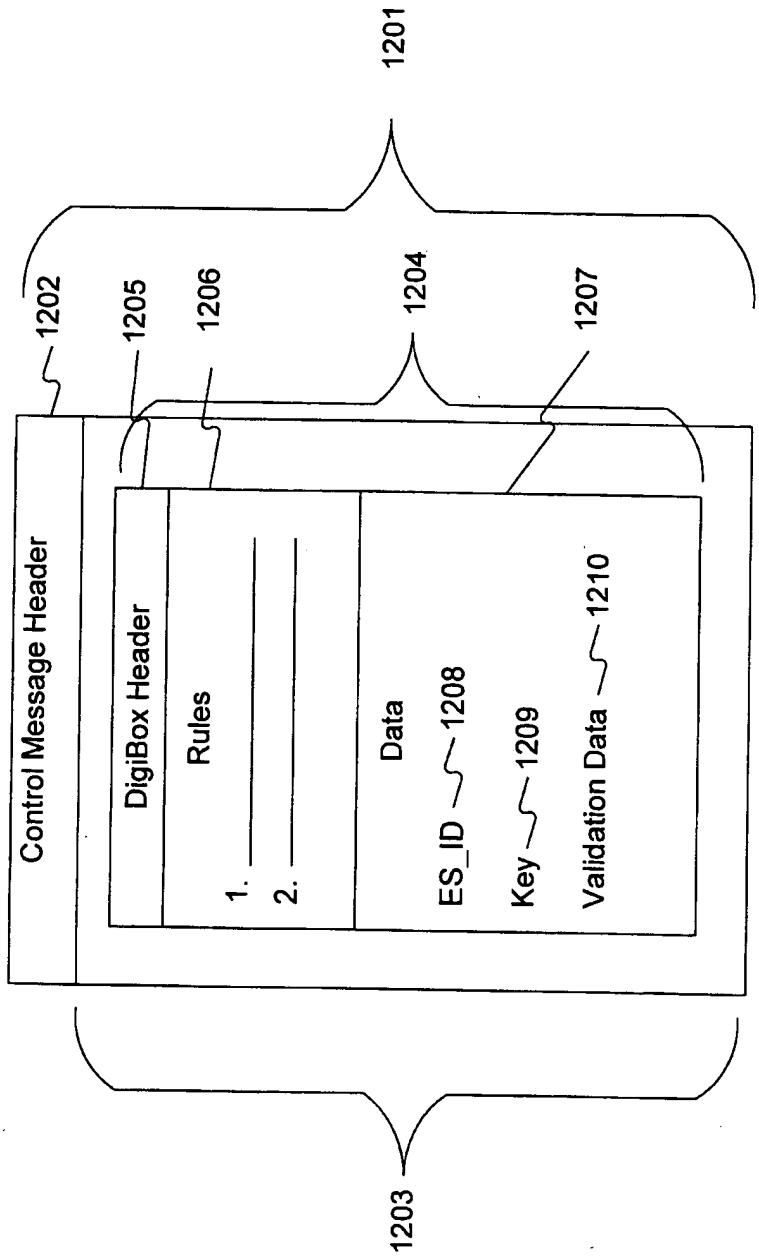

1201
1202
1205
1206
1204
1207
Control Message Header
DigiBox Header
Rules
1.
2.
Data
ES_ID
1208
Key
1209
Validation Data
1210
1203

Fig. 12

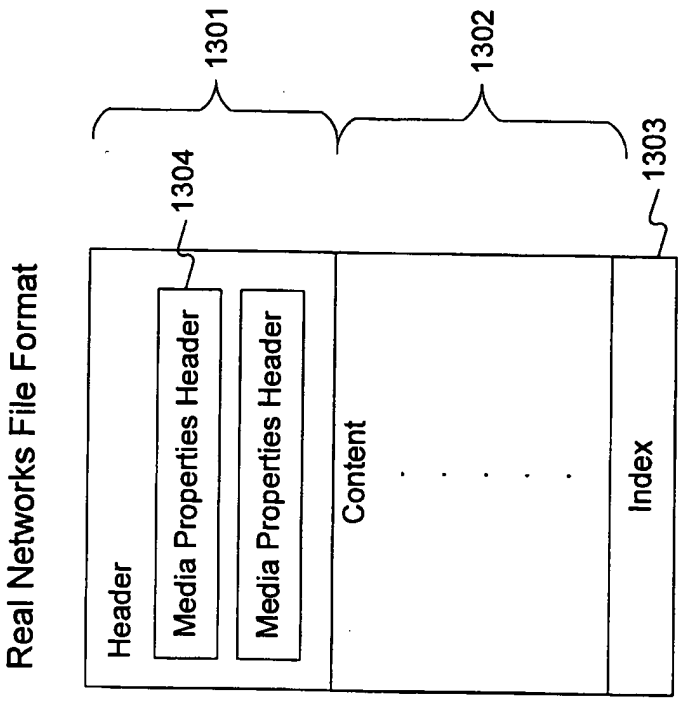

Real Networks File Format
Header
Media Properties Header
Media Properties Header
Content
Index
1301
1302
1303
1304

Fig. 13

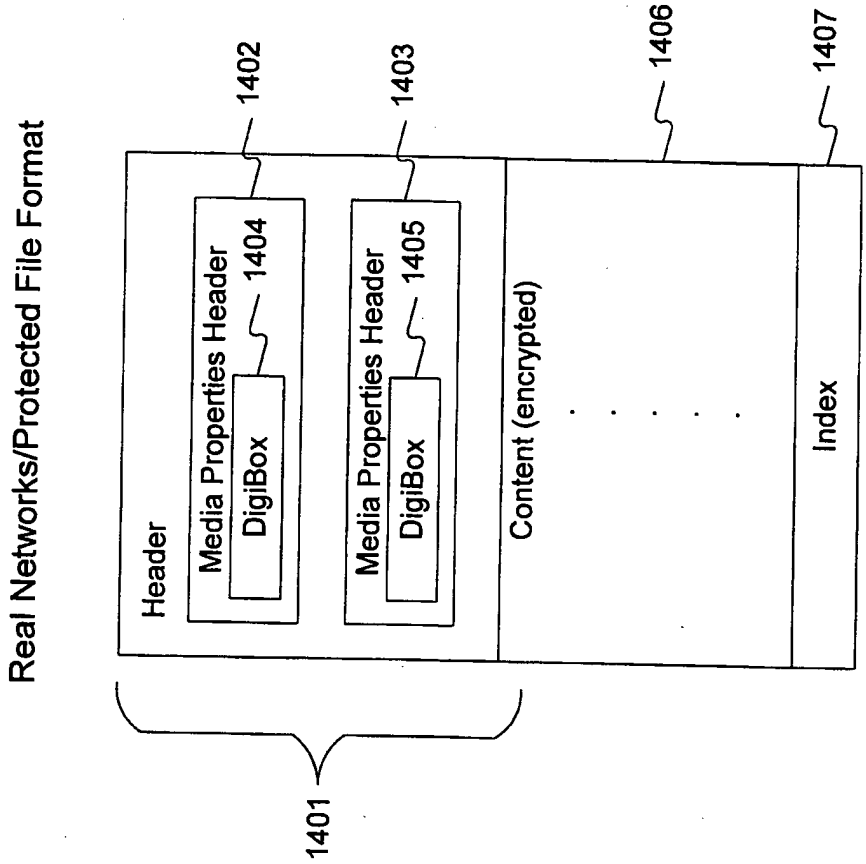

Real Networks/Protected File Format
1401
Header
Media Properties Header
DigiBox
1404
1402
Media Properties Header
DigiBox
1405
1403
Content (encrypted)
1406
Index
1407

Fig. 14

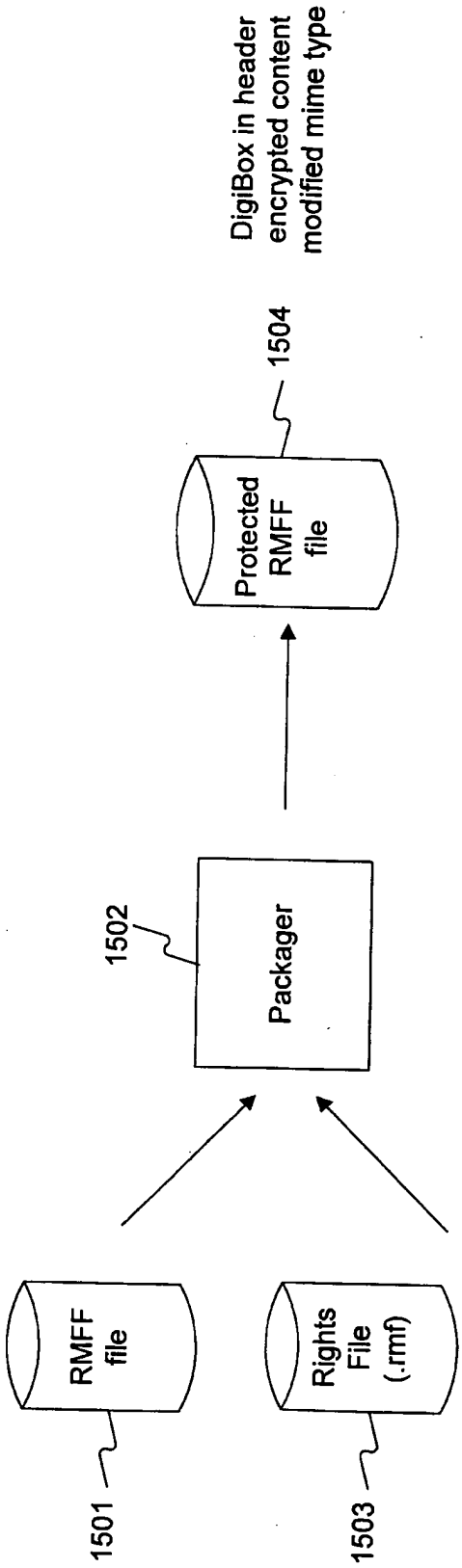

DigiBox in header
encrypted content
modified mime type
1504
Protected RMFF file
1502
Packager
RMFF file
Rights File (.rmf)
1501
1503

Fig. 15

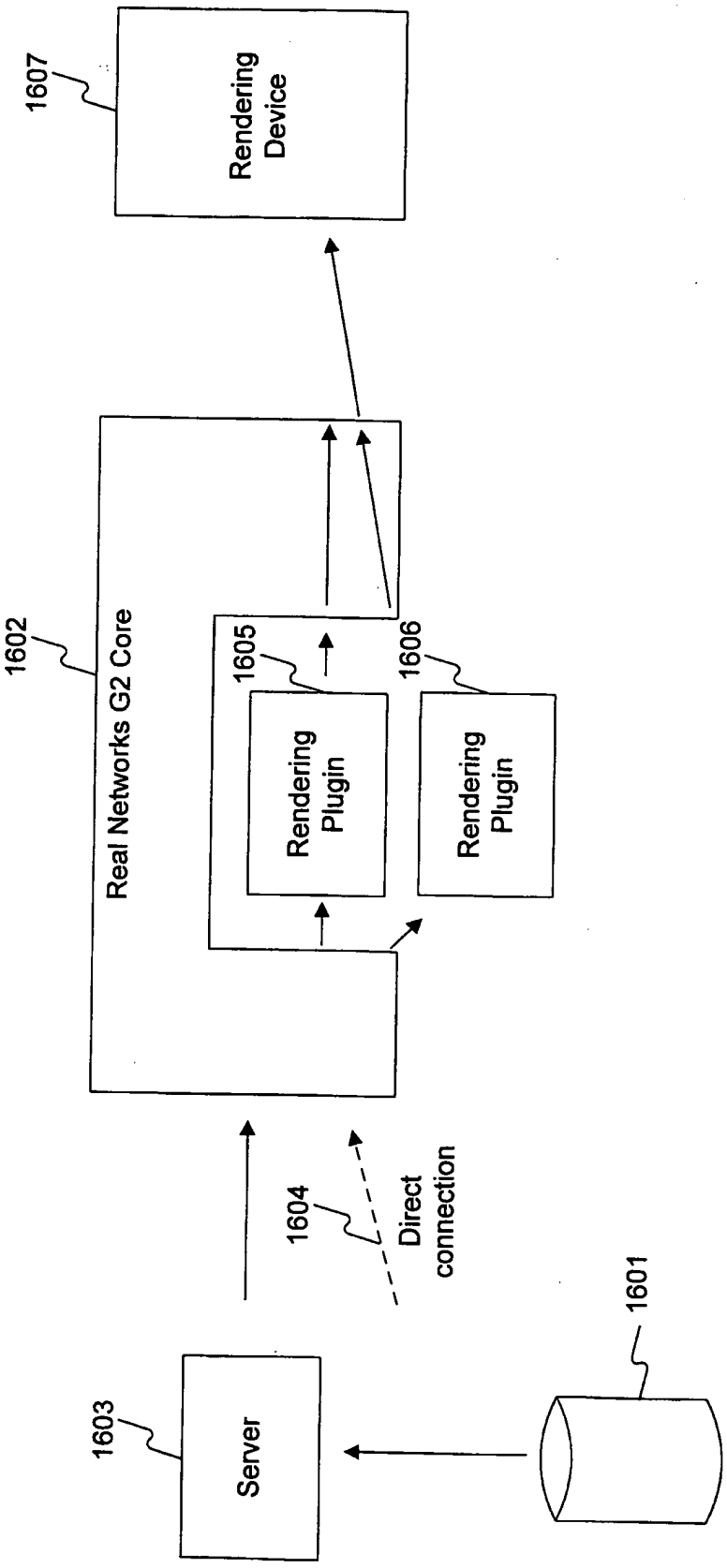

1607
Rendering Device
1602
Real Networks G2 Core
1605
Rendering Plugin
1606
Rendering Plugin
1604
Direct connection
1601
1603
Server

Fig. 16

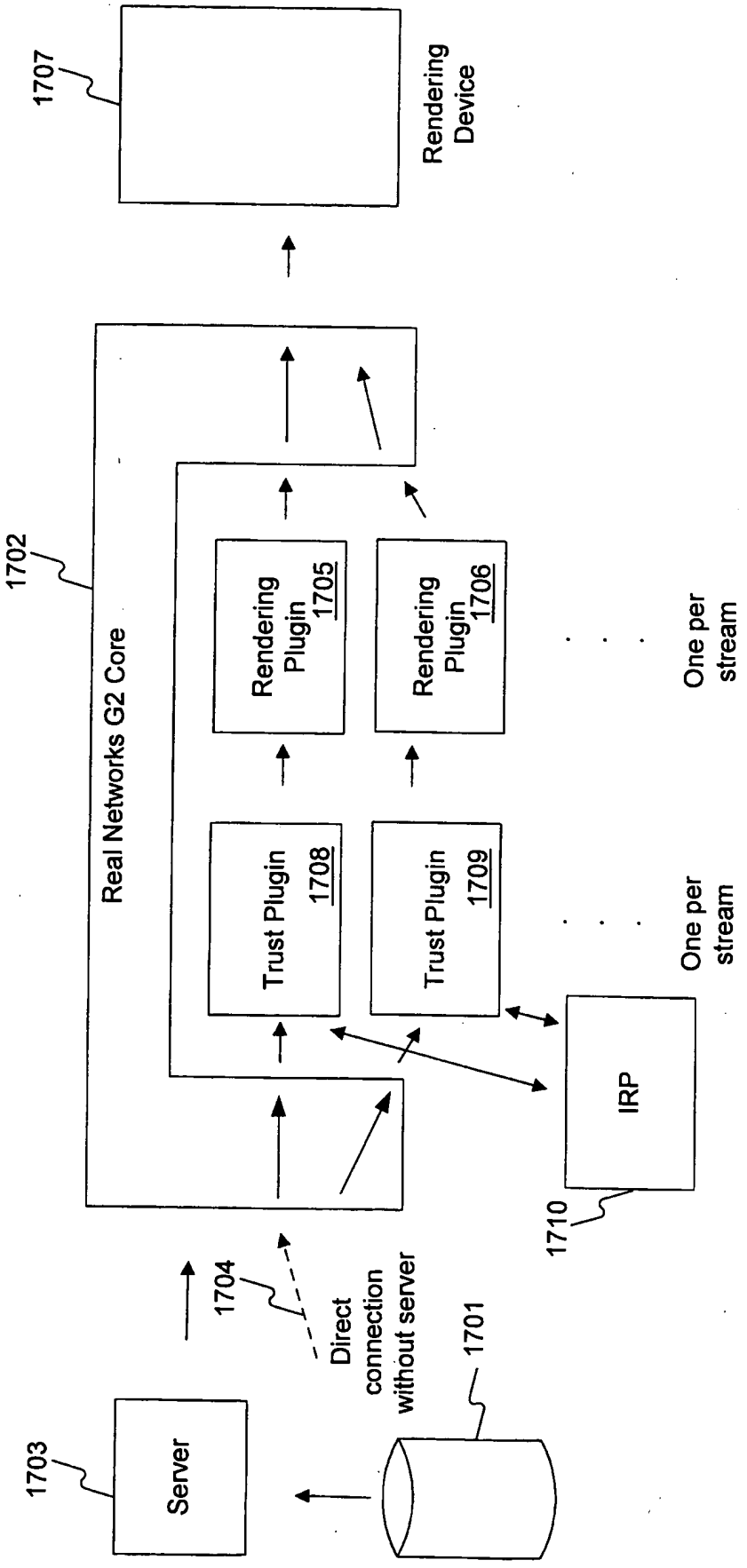

1707
Rendering Device
1702
Real Networks G2 Core
Rendering Plugin 1705
Rendering Plugin 1706
One per stream
Trust Plugin 1708
Trust Plugin 1709
One per stream
IRP
1710
1704
Direct connection without server
1701
1703
Server

Fig. 17

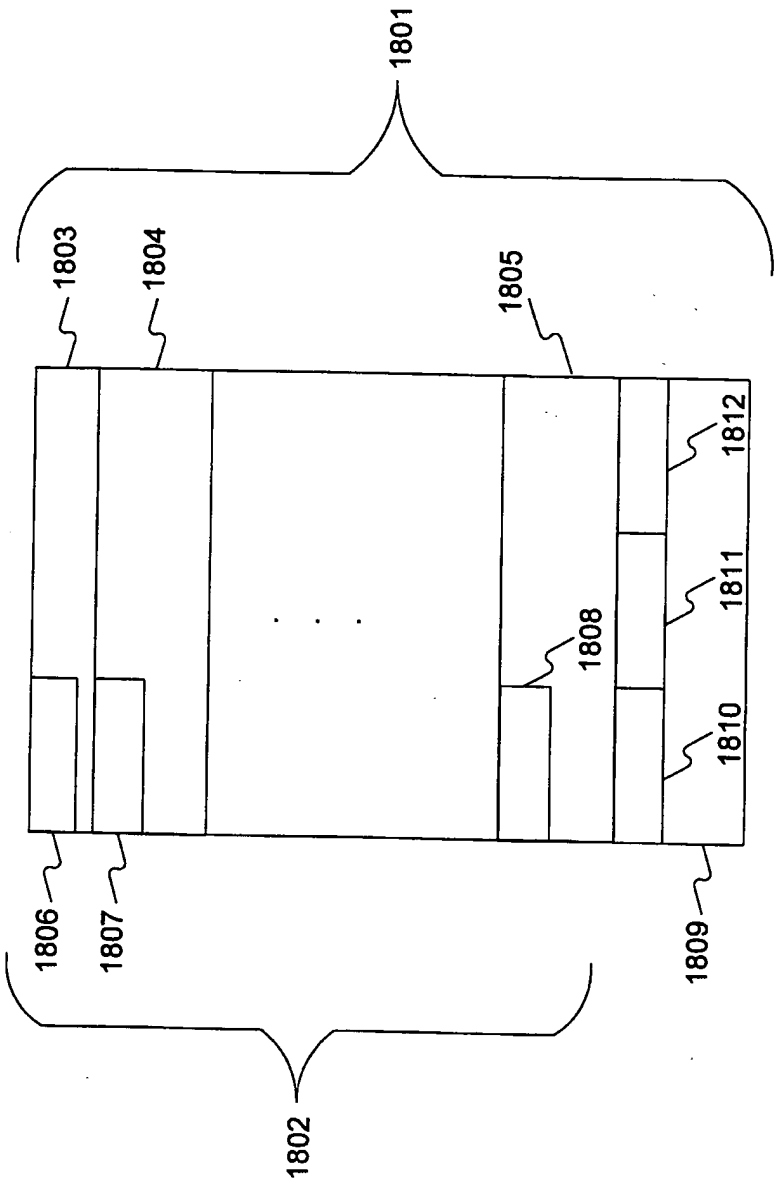
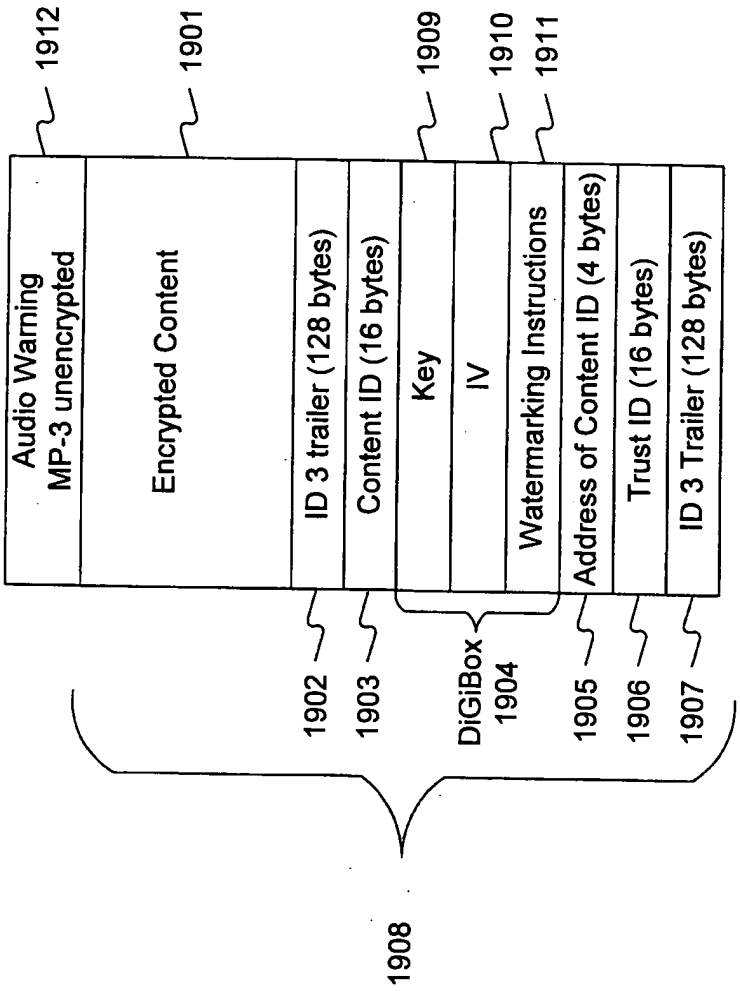


Fig. 18



Protected MP3 Format

Fig. 19

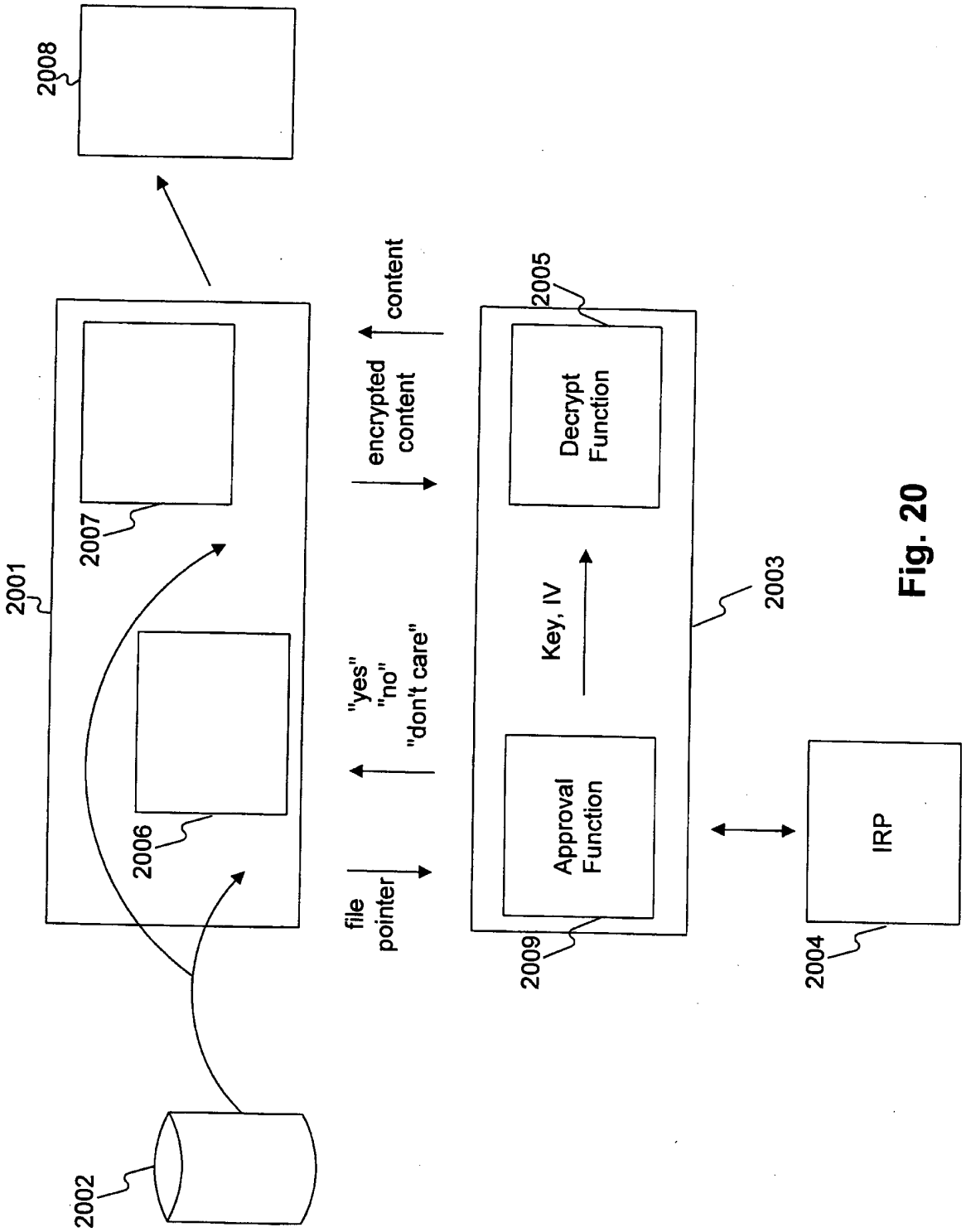

2008
2001
2007
2006
2002
file pointer
"yes"
"no"
"don't care"
encrypted content
content
2005
Decrypt Function
Key, IV
Approval Function
2009
2003
IRP
2004

Fig. 20

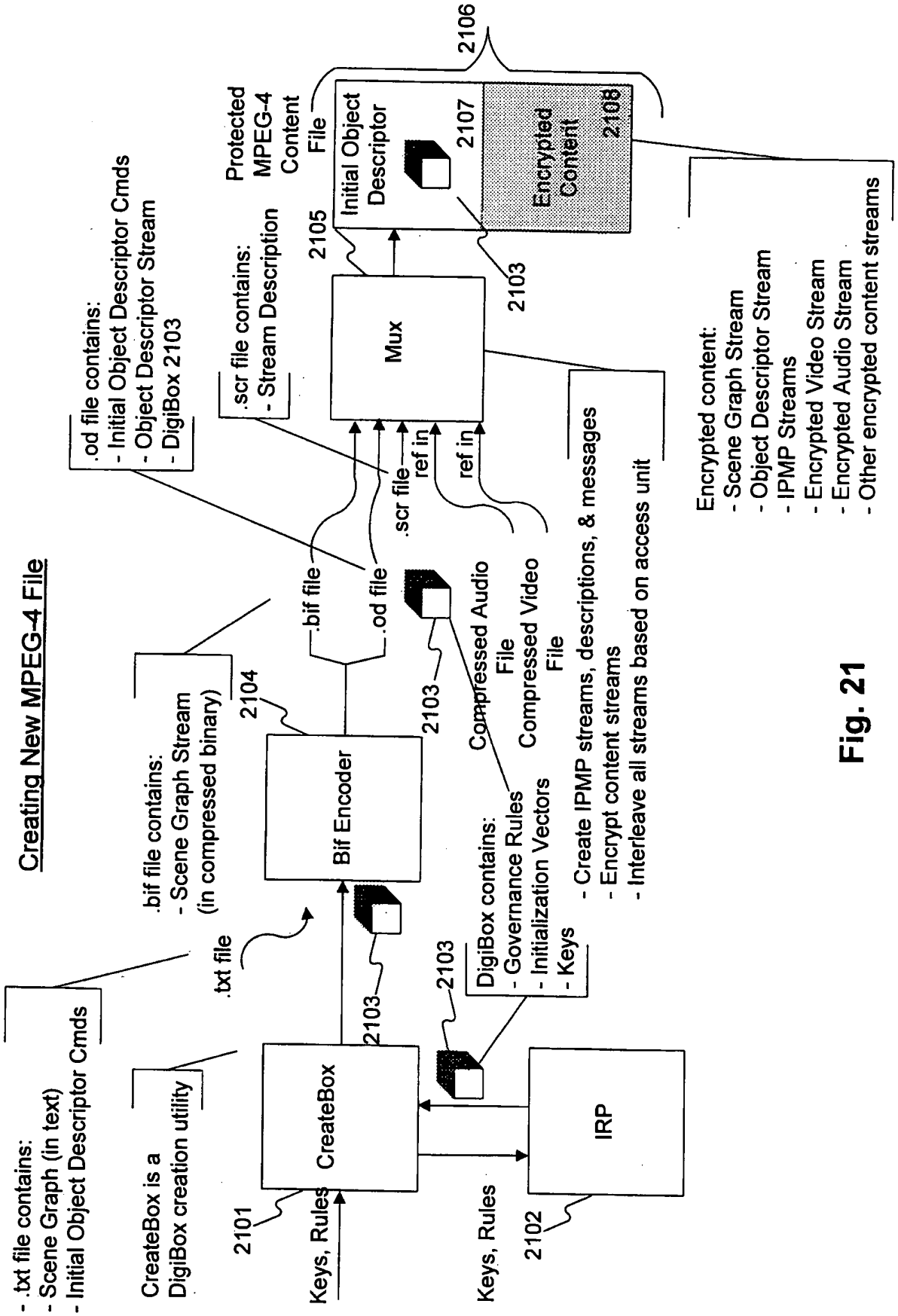

Creating New MPEG-4 File
- .txt file contains:
- Scene Graph (in text)
- Initial Object Descriptor Cmds
CreateBox is a DigiBox creation utility
CreateBox
IRP
Keys, Rules
Keys, Rules
2101
2102
2103
2103
2103
.txt file
.bif file contains:
- Scene Graph Stream
(in compressed binary)
2104
Bif Encoder
DigiBox contains:
- Governance Rules
- Initialization Vectors
- Keys
- Create IPMP streams, descriptions, & messages
- Encrypt content streams
- Interleave all streams based on access unit
.bif file
.od file
.scr file
ref in
ref in
Compressed Audio File
Compressed Video File
.od file contains:
- Initial Object Descriptor Cmds
- Object Descriptor Stream
- DigiBox 2103
.scr file contains:
- Stream Description
Mux
2103
2105
Initial Object Descriptor
2107
Encrypted Content
2108
Protected MPEG-4 Content File
2106
Encrypted content:
- Scene Graph Stream
- Object Descriptor Stream
- IPMP Streams
- Encrypted Video Stream
- Encrypted Audio Stream
- Other encrypted content streams

Fig. 21

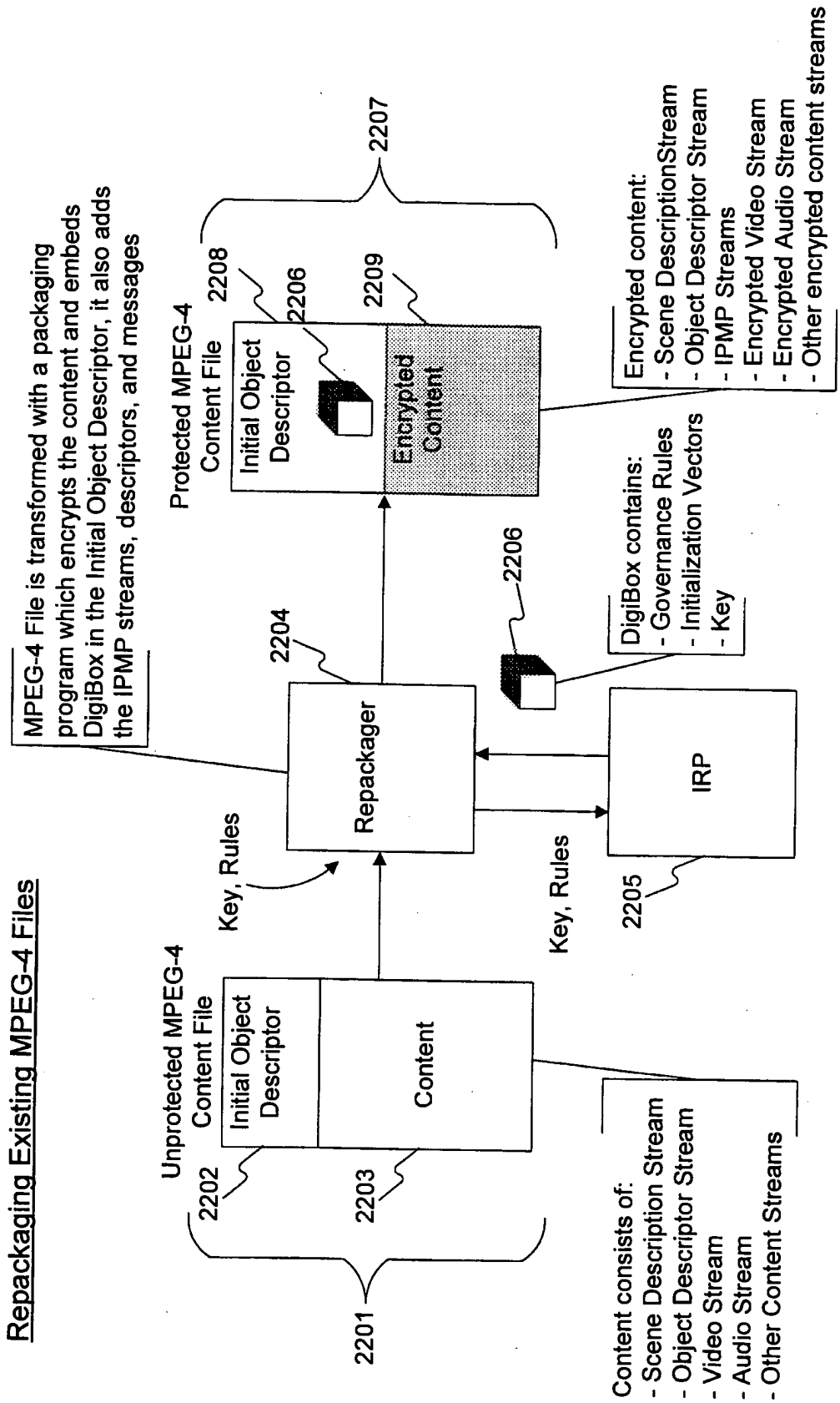

Repackaging Existing MPEG-4 Files
Unprotected MPEG-4 Content File
Initial Object Descriptor
Content
2202
2203
2201
Content consists of:
- Scene Description Stream
- Object Descriptor Stream
- Video Stream
- Audio Stream
- Other Content Streams
Key, Rules
Repackager
2204
MPEG-4 File is transformed with a packaging program which encrypts the content and embeds DigiBox in the Initial Object Descriptor, it also adds the IPMP streams, descriptors, and messages
Protected MPEG-4 Content File
Initial Object Descriptor
Encrypted Content
2208
2206
2209
2207
2206
DigiBox contains:
- Governance Rules
- Initialization Vectors
- Key
Key, Rules
IRP
2205
Encrypted content:
- Scene DescriptionStream
- Object Descriptor Stream
- IPMP Streams
- Encrypted Video Stream
- Encrypted Audio Stream
- Other encrypted content streams

Fig. 22

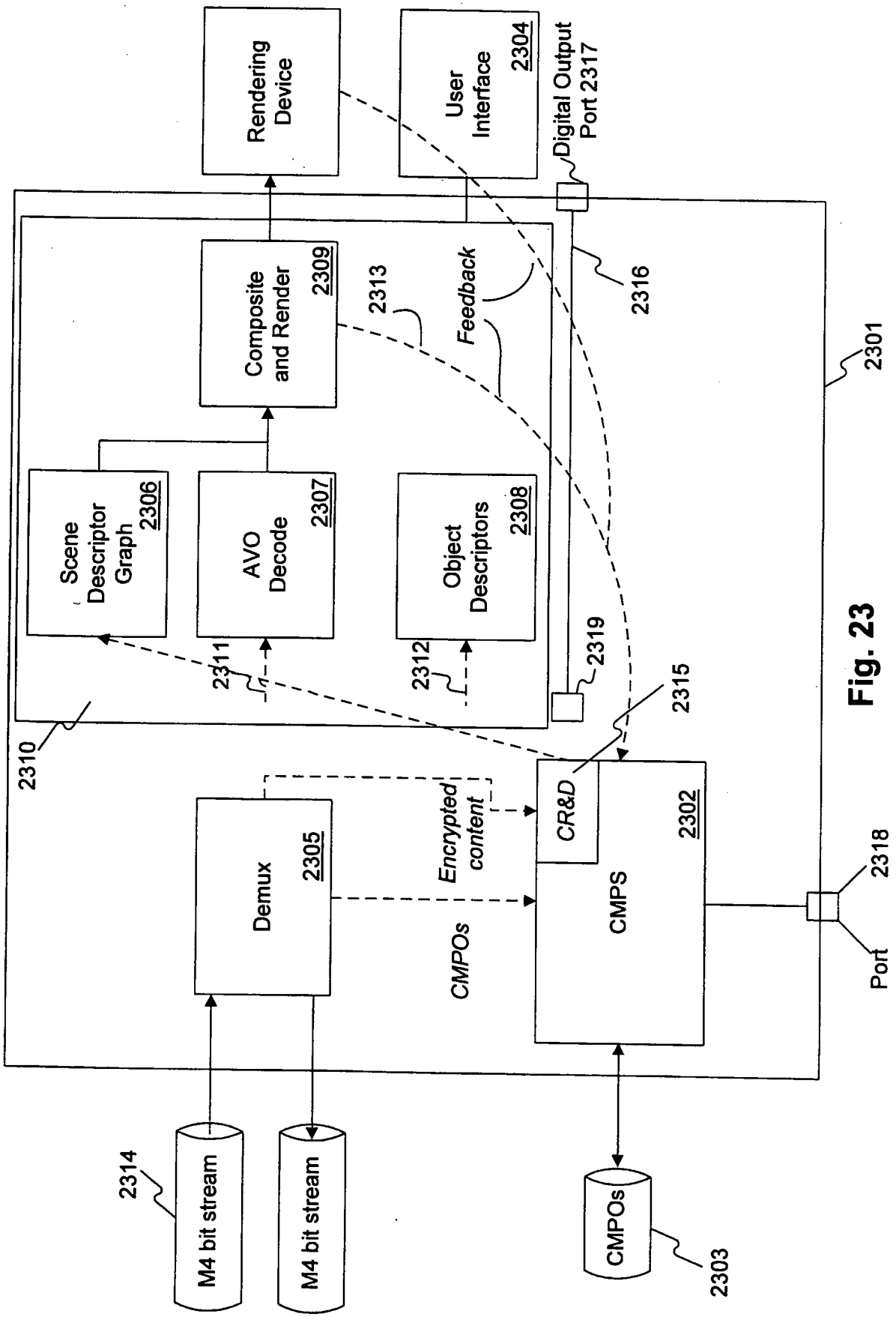

Rendering Device
User Interface 2304
Digital Output Port 2317
Composite and Render 2309
2313
Feedback
2316
2301
Scene Descriptor Graph 2306
AVO Decode 2307
Object Descriptors 2308
2311
2312
2319
2315
2310
CR&D
2302
Demux 2305
Encrypted content
CMPOs
CMPS
2318
Port
2314
M4 bit stream
M4 bit stream
CMPOs
2303

Fig. 23

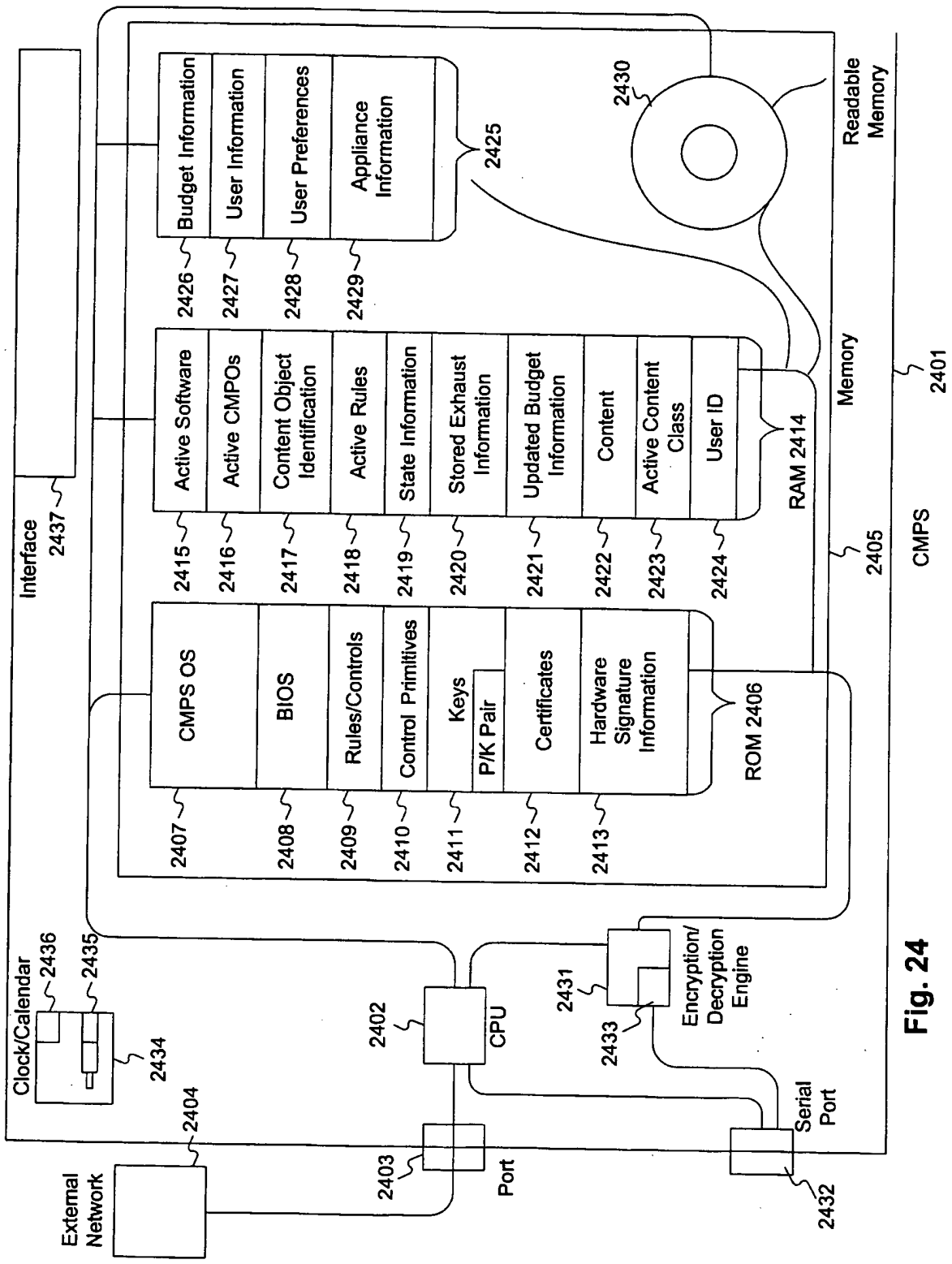

Budget Information
User Information
User Preferences
Appliance Information
2425
2426
2427
2428
2429
2430
Readable Memory
Active Software
Active CMPOs
Content Object Identification
Active Rules
State Information
Stored Exhaust Information
Updated Budget Information
Content
Active Content Class
User ID
RAM 2414
Memory
2401
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2405
CMPS
Interface
2437
CMPS OS
BIOS
Rules/Controls
Control Primitives
Keys
P/K Pair
Certificates
Hardware Signature Information
ROM 2406
2407
2408
2409
2410
2411
2412
2413
Clock/Calendar
2436
2435
2434
2404
External Network
2403
Port
2402
CPU
2431
2433
Encryption/ Decryption Engine
Serial Port
2432

Fig. 24

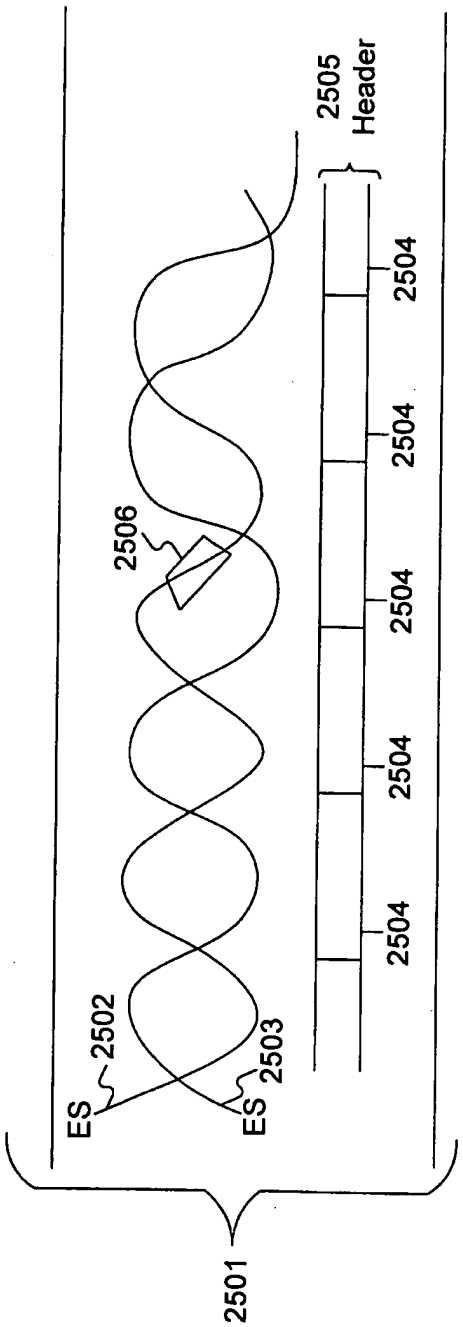

2501
ES
2502
ES
2503
2506
2505
Header
2504
2504
2504
2504
2504
Aggregate Stream

Fig. 25

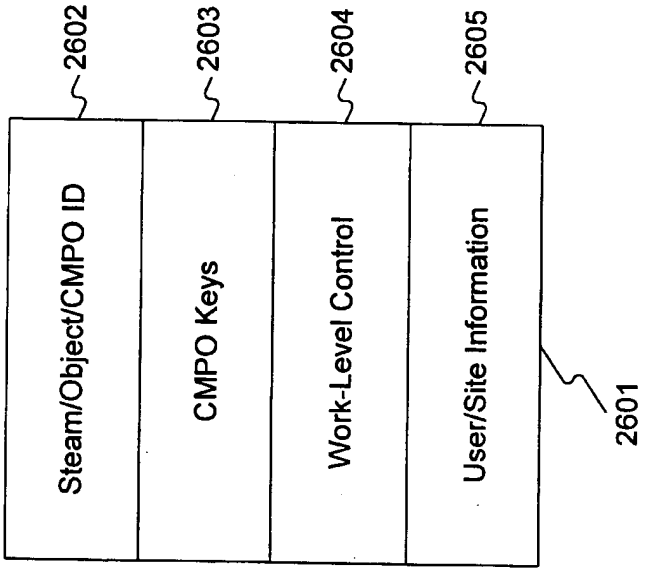

2602
2603
2604
2605
Steam/Object/CMPO ID
CMPO Keys
Work-Level Control
User/Site Information
2601

Fig. 26

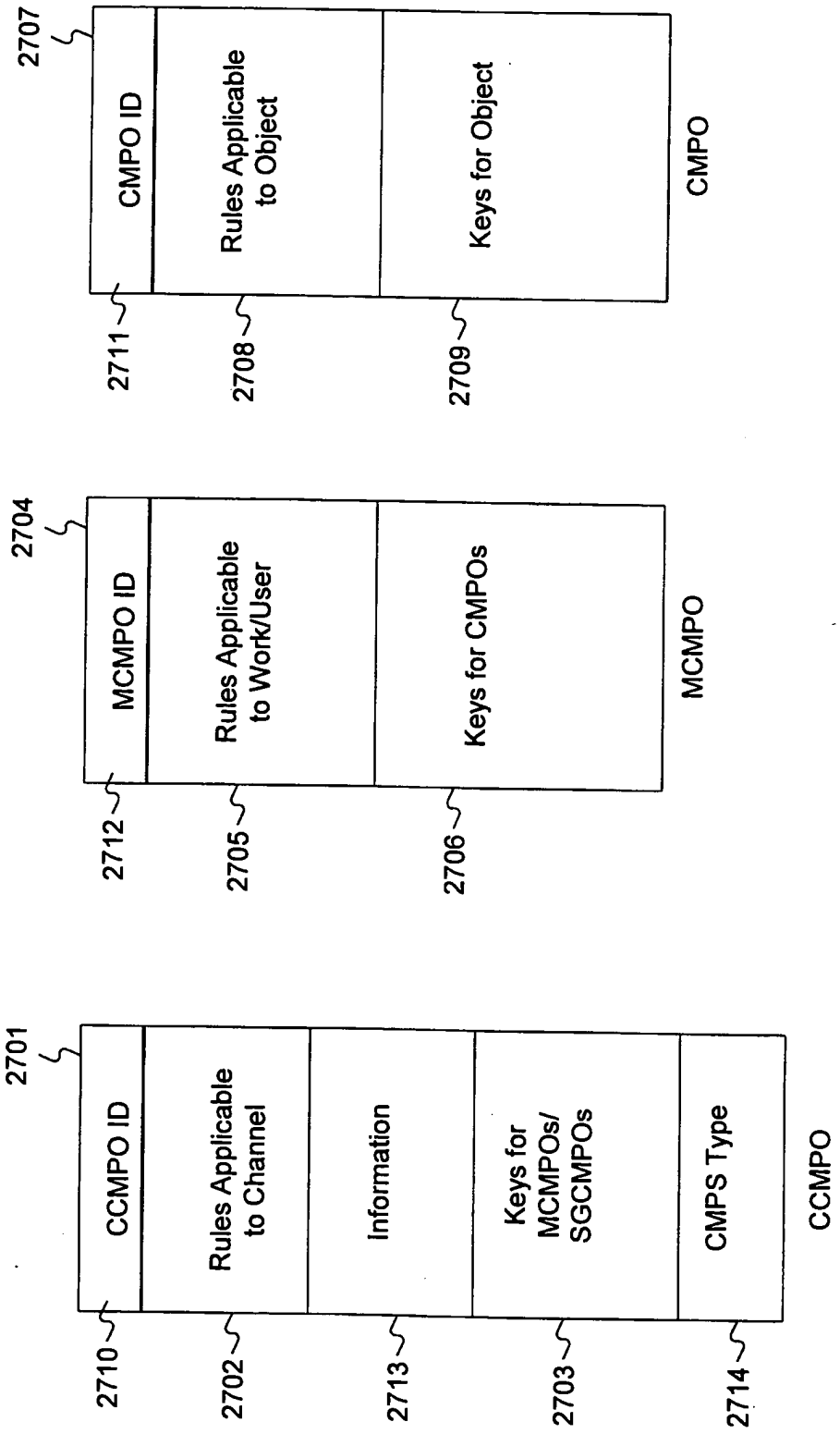

2707
CMPO ID
Rules Applicable to Object
Keys for Object
CMPO
2711
2708
2709
2704
MCMPO ID
Rules Applicable to Work/User
Keys for CMPOs
MCMPO
2712
2705
2706
2701
CCMPO ID
Rules Applicable to Channel
Information
Keys for MCMPOs/ SGCMPOs
CMPS Type
CCMPO
2710
2702
2713
2703
2714

Fig. 27

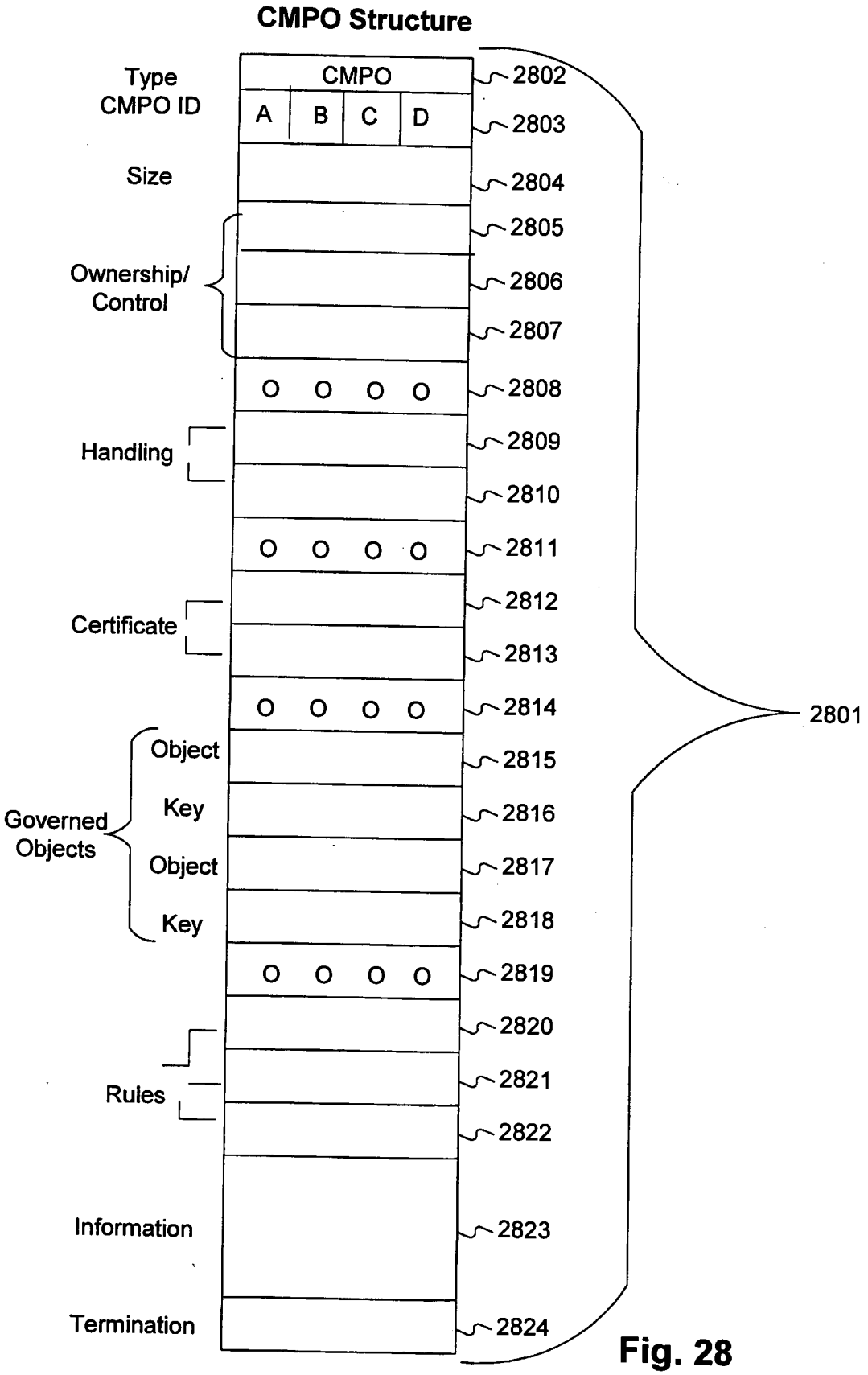

CMPO Structure
Type
CMPO ID
CMPO
A
B
C
D
Size
Ownership/
Control
O O O O
Handling
O O O O
Certificate
O O O O
Object
Key
Governed
Objects
Object
Key
O O O O
Rules
Information
Termination
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2801

Fig. 28

INTERNATIONAL SEARCH REPORT

Inter ɔnal Application No
PCT/US 99/05734

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 H04N7/167 G06F1/00

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 H04N G06F G11B

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	EP 0 763 936 A (LG ELECTRONICS INC) 19 March 1997	1-4, 6-14, 17-20, 22-25
A	see abstract see column 6, line 27 - column 8, line 4 see column 9, line 6 - column 11, line 43 see column 16, line 47 - column 18, line 38 see figures 4,6A,6B,7 see figures 10,16	5,15,16, 21,26
	--- -/--	

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
1 July 1999	09/07/1999

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Hampson, F

1

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Inte: ɔnal Application No
PCT/US 99/05734

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
X	EP 0 714 204 A (LG ELECTRONICS INC) 29 May 1996	1-4, 6-14, 22-26
A	see abstract see page 6, line 14 - page 8, line 45 see figures 7,19A,B,20	5,15-21
A	EP 0 715 246 A (XEROX CORP) 5 June 1996 see abstract see page 3, line 46 - page 8, line 27 see figures 1-3,4A,4B	1-26
A	WO 97 25816 A (SONY CORP ;INOUE HAJIME (US); LEE CHUEN CHIEN (US); SONY ELECTRONI) 17 July 1997 see abstract see page 7, line 10 - page 10, line 7 see figures 2,3	1-26
A	EP 0 800 312 A (MATSUSHITA ELECTRIC IND CO LTD) 8 October 1997 see abstract see column 50, line 20 - column 51, line 53	1,6,7, 23-25

1

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Information on patent family members

Inter ional Application No
PCT/US 99/05734

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 0763936 A	19-03-1997	CN 1150738 A	28-05-1997
		JP 9093561 A	04-04-1997
		US 5799081 A	25-08-1998
EP 0714204 A	29-05-1996	CN 1137723 A	11-12-1996
		JP 8242438 A	17-09-1996
		US 5757909 A	26-05-1998
EP 0715246 A	05-06-1996	US 5638443 A	10-06-1997
		JP 8263439 A	11-10-1996
WO 9725816 A	17-07-1997	AU 1344097 A	01-08-1997
		CN 1209247 A	24-02-1999
		EP 0882357 A	09-12-1998
		US 5889919 A	30-03-1999
EP 0800312 A	08-10-1997	WO 9714249 A	17-04-1997
		CN 1168054 A	17-12-1997
		EP 0789361 A	13-08-1997
		JP 10079174 A	24-03-1998

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 1/00 — A1

(11) International Publication Number: **WO 99/60461**

(43) International Publication Date: 25 November 1999 (25.11.99)

(21) International Application Number: PCT/GB98/03828

(22) International Filing Date: 18 December 1998 (18.12.98)

(30) Priority Data:
09/080,030 15 May 1998 (15.05.98) US

(71) Applicant: INTERNATIONAL BUSINESS MACHINES CORPORATION [US/US]; New Orchard Road, Armonk, NY 10504 (US).

(71) Applicant *(for MC only)*: IBM UNITED KINGDOM LIMITED [GB/GB]; North Harbour, Portsmouth, P.O. Box 41, Hampshire PO6 3AU (GB).

(72) Inventors: BERSTIS, Viktors; 5194 Cuesta Verde, Austin, TX 78746 (US). HIMMEL, Maria, Azua; 6403 Rain Creek Parkway, Austin, TX 78759 (US).

(74) Agent: BOYCE, Conor; IBM United Kingdom Limited, Intellectual Property Law, Hursley Park, Winchester, Hampshire SO21 2JN (GB).

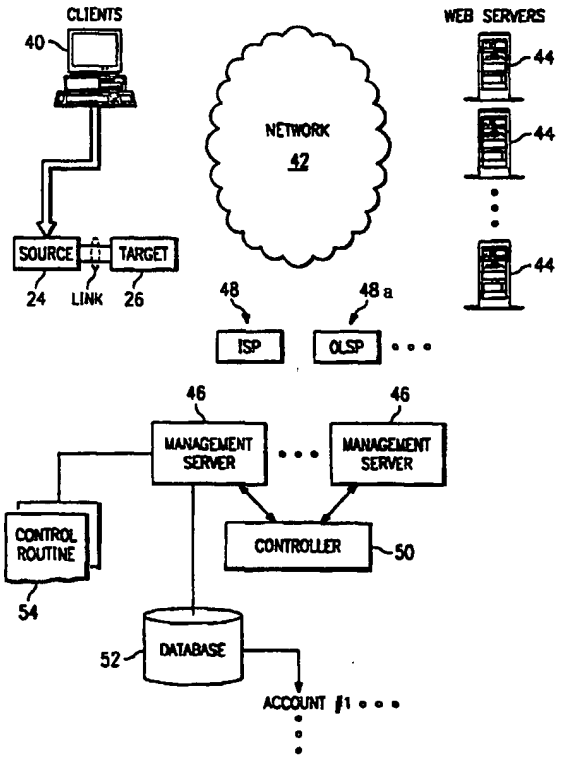
(81) Designated States: CN, CZ, IL, IN, JP, KR, PL, SG, European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.

(54) Title: ROYALTY COLLECTION METHOD AND SYSTEM FOR USE OF COPYRIGHTED DIGITAL MATERIALS ON THE INTERNET

(57) Abstract

A method, system and computer program product to facilitate royalty collection with respect to online distribution of electronically published material over a computer network. In one embodiment, a method for managing use of a digital file (that includes content subject to copyright protection on behalf of some content provider) begins by establishing a count of a number of permitted copies of the digital file. In response to a given protocol, a copy of the digital file is then selectively transferred from a source to a target. Thus, for example, the source and target may be located on the same computer with the source being a disk storage device and the target being a rendering device (e.g., a printer, a display, a sound card or the like). The method logs an indication each time the digital file is transferred from the source to a target rendering device, and the count is decremented upon each transfer. When the count reaches a given value (e.g., zero), the file is destroyed or otherwise prevented from being transferred from the source device. The indications logged are transferred to a management server to facilitate payment of royalties to the content provider.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

ROYALTY COLLECTION METHOD AND SYSTEM FOR USE OF COPYRIGHTED DIGITAL MATERIALS ON THE INTERNET

BACKGROUND OF THE INVENTION

Technical Field

The present invention relates generally to managing collection of royalties for electronically-published material distributed over a computer network.

Description of the Related Art

The World Wide Web is the Internet's multimedia information retrieval system. In the Web environment, client machines effect transactions to Web servers using the Hypertext Transfer Protocol (HTTP), which is a known application protocol providing users access to files (e.g., text, graphics, images, sound, video, etc.) using a standard page description language known as Hypertext Markup Language (HTML). HTML provides basic document formatting and allows the developer to specify "links" to other servers and files. In the Internet paradigm, a network path to a server is identified by a so-called Uniform Resource Locator (URL) having a special syntax for defining a network connection. Use of an HTML-compatible browser (e.g., Netscape Navigator or Microsoft Internet Explorer) at a client machine involves specification of a link via the URL. In response, the client makes a request to the server (sometimes referred to as a "Web site") identified in the link and, in return, receives in return a document or other object formatted according to HTML.

One of the technical advantages of the World Wide Web is the ease with which digital content (e.g., graphics, sound, video, movies and the like) may be transmitted and distributed to many users. Indeed, copying a digital file is as easy as clicking on a computer mouse. Copyright laws afford a copyright owner the exclusive right to reproduce the copyrighted work in copies, to distribute such copies, and to publicly perform and display the work. Each time a digital file is transferred over the Internet and copied onto a user s memory, the copyright owner's exclusive reproduction right is implicated (and possibly violated). Likewise, transmission of the copyrighted work over the physical wire is tantamount to a distribution. Indeed, in an open system (e.g., a personal computer accessing the World Wide Web through an Internet Service Provider (ISP)), copies of copyrighted materials can undergo unlimited further copying and transmission without the ability of the owner to collect appropriate compensation (e.g., royalties).

Many publishers or other content providers naturally are hesitant to make their copyrighted works available over the Internet due to the ease with which these materials may be copied and widely disseminated without adequate compensation. Presently, Internet commerce remains highly unregulated, and there is no central authority for managing collection and allocation of content provider royalties. Moreover, while publishers and content rights societies and organizations are attempting to address the legal and logistical issues, the art has yet to develop viable technical solutions.

One technique that has been proposed involves wrapping a copyrighted work in a copy protection "environment" to facilitate charging users for use of that information obtained from the Internet or World Wide Web. This approach, called COPINET, links a copyright protection mechanism with a copyright management system, and it is described in *Charging, paying and copyright - information access in open networks*, Bennett et al., 19th International Online Information Meeting Proceedings, Online Information 1995 pp. 13-23 (Learned Information Europe Ltd.). Publishers in such a system can determine an appropriate level of protection while monitoring use and managing the chain of rights. This approach is also said to provide protection for digital material even after delivery to the user workstation. In particular, copyright material is "wrapped" (by encryption) and "unwrapped" as a result of a specific authorization provided by a trusted subsystem. Material thus is only "visible" to the environment and thus any subsequent user actions, such as "save" or "copy", result in the protected material, or material derived from it, remaining in a protected state when outside the environment.

Although the above-described approach provides some advantages, it does not address the problem of managing the collection of royalties and/or the allocating of such payments to content providers. Moreover, it is not an accepting solution in the context of an open PC architecture such as implemented in the public Internet. It also requires the use of a separate trusted subsystem to generate the authorizations for particular content transfers, which is undesirable.

Other known techniques for managing use of content over the Internet typically involve electronic "wallets" or smart cards. Known prior art systems of this type are illustrated, for example, in U.S. Patent Nos. 5,590,197 and 5,613,001. These systems involve complex hardware and encryption schemes, which are expensive and difficult to implement in practice. They are not readily adaptable to provide general royalty payment schemes for Internet content usage.

Thus, there remains a need to provide improved methods and systems for collecting royalties on the Internet as a result of use of copyrighted content.

The present invention solves this important problem.

SUMMARY OF THE INVENTION

An object of this invention is to enable a pair of "certified" devices (e.g., a storage device and a rendering device) to operate within the context of a given security protocol and thereby manage copies of a digital file and associated copy control information.

Still another object of this invention is to enable a copyright proprietor to maintain a degree of control over copyrighted content even after that content has been fetched from a server and downloaded to a client machine, e.g., in a Web client-server environment.

A particular object of the present invention is to manage the number of copies of a digital file that may be made within a Web appliance having a secure disk storage and that is connectable to the Internet using a dialup network connection.

A still further object of this invention is to restrict a number of copies of a digital file that may be made at a given Web client machine connected to the World Wide Web.

It is yet another object of this invention to enable a publisher of an electronic document to control the number of copies of such document that may be made on the Internet by permitted users.

It is a more general object of this invention to manage permissible use of copyrighted content on the Internet and World Wide Web.

It is still another more general object of this invention to manage collection of information to facilitate payment of appropriate compensation to content providers and publishers arising from use of their copyrighted content on the Internet.

Another object of this invention is to manage the charging of users for information obtained from the Internet or World Wide Web.

A still further object of this invention is to facilitate royalty collection as a result of electronically published material distributed

online over a computer network (e.g., the public Internet, an intranet, an extranet or other network).

One embodiment of the invention is a method for managing copies of a digital file, which includes content subject to copyright protection, on behalf of some content provider (e.g., an author, publisher or other). It is assumed that a given usage scheme has been established with respect to the file as defined in copy control information associated with the file. Thus, for example, the copy control information may define a set of payment options including, without limitation, prepayment (for "n" copies), pay-per-copy (as each copy is made), IOU (for copies made offline), or some other payment option. The copy control information may also include other data defining how the file is managed by the scheme including: a count of the number of permitted copies, a count of the number of permitted pay-per-copy versions, copyright management information, payee information, an expiration date (after which copying is no longer permitted), and the like.

The present invention assumes the existence of a pair of devices, a "source" and a "target", that have been or are certified to use the scheme. Typically, the "source" is a storage device while the "target" is a rendering device. An illustrative storage device may be disk storage, system memory, or the like. An illustrative rendering device may be a printer, a display, a sound card or the like. The source and target devices may both be storage devices (e.g., a Web server and a client disk storage). In either case, each of the devices comprising the pair is "certified" (typically upon manufacture) to operate under a given security protocol. Under the protocol, the devices include appropriate circuitry and/or software, as the case may be, to facilitate the establishment of a secure link between the storage and rendering devices. Each device requires the other to validate itself and thus prove that the device can be trusted to manage the content (namely, the digital file) sought to be protected.

When the technique is implemented in an "open" client-server environment, hardware devices (e.g., microcontrollers) preferably are used in the storage and rendering devices to facilitate generation of the secure link. When the technique is implemented in a "closed" Web appliance environment, the secure link may be established and managed using software resident in the control routines associated with the storage and rendering devices. The secure link may be established and managed in software under such conditions because, in the Web appliance environment, it is possible to readily disable the secure link in the event of tampering with the appliance housing or other circuitry. Regardless of the environment, the secure link is first established

between the "certified" storage and rendering devices. Thereafter, the digital file, together with at least part of its copy control information, is transferable between the storage and rendering devices in accordance with the particular usage and payment scheme being utilized. Thus, for example, if a prepayment scheme is implemented and an expiration date (associated therewith) has not occurred, a given number of copies of the file may be transferred between the storage and rendering devices. The prepayment funds are collected at a central location and then redistributed to the copyright proprietor or some third party.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a representative system in which the present invention is implemented;

Figure 2 is a simplified block diagram of a source device and a target device connected by a channel over which a digital file is transferred according to the present invention;

Figure 3 is an illustrative example of a source device connected to a set of target rendering devices in a client computer;

Figure 4 is a block diagram of a representative copyright management system according to the present invention;

Figure 5 is a flowchart of a preferred method of managing a digital file according to the present invention;

Figure 6A is pictorial representation of a data processing system unit connected to a conventional television set to form a "Web" appliance;

Figure 6B is a pictorial representation of a front panel of the data processing system unit;

Figure 6C is a pictorial representation of a rear panel of the data processing system unit;

Figure 6D is a pictorial representation of a remote control unit associated with the data processing system unit; and

Figure 7 is a block diagram of the major components of the data processing system unit.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

A representative system in which the present invention is implemented is illustrated in **Figure 1**. A plurality of Internet client machines **10** are connectable to a computer network Internet Service Provider (ISP) **12** via a "resource" such as a dialup telephone network **14**. As is well known, the a dialup telephone network usually has a given, limited number of connections **16a-16n**. ISP **12** interfaces the client machines **10** to the remainder of the network **18**, which includes a plurality of Internet server machines **20**. A client machine typically includes a suite of known Internet tools (e.g., Web browser **13**) to access the servers of the network and thus obtain certain services. These services include one-to-one messaging (e-mail), one-to-many messaging (bulletin board), on-line chat, file transfer and browsing. Various known Internet protocols are used for these services. Thus, for example, browsing is effected using the Hypertext Transfer Protocol (HTTP), which provides users access to multimedia files using Hypertext Markup Language (HTML). The collection of servers that use HTTP comprise the World Wide Web, which is the Internet's multimedia information retrieval system.

As will be described in more detail below, the present invention may be implemented in hardware and/or in software. The software implementation is particularly useful when the client machine is an Internet or Web appliance, such as illustrated in **Figures 6A-6D**. In the case of the software implementation, a client machine has associated therewith a software routine **15** designed to perform one or more of the functions of the digital file copy protection method, as will be described. The software is preferably a client application (although it may be implemented with the browser as a plug-in, or with a client-side proxy, or as a standalone application). Alternatively, the agent is built into the browser, or it is implemented as a Java applet or standalone application. Thus, as used herein, in this particular embodiment, the software **15** is any application running on a client machine **10** that performs the copy protection/royalty management task(s) on behalf of the user(s) of that client according to the present invention.

The discussion which follows primarily uses the words "copying" or "copies" to describe the control of the further exercise of a copyright right for a particular work. The reader should understand that "copying" could include other types of rendering of the work for different devices. That is, "copying" in a printer would entail printing on paper or another substrate. Copying on a display is presenting an image on the screen. Copying in an audio device would be the performance of an audio portion of the work. Each of these devices both storage devices, e.g., hard disks, tapes in CDR, and rendering devices, e.g., prints, display graph, audio

player, movie player, should be equipped with the present invention so that the copies are controlled throughout the systems and networks until their final rendering place.

The present invention is a method for managing copies of a digital file, which includes content subject to copyright protection, on behalf of some content provider (e.g., an author, publisher or other). It is assumed that a given payment scheme has been established with respect to the file. Thus, for example, such payment schemes include, without limitation, prepayment (for "n" copies), pay-per-copy (as each copy is made), IOU (for copies made offline), or some other payment option. In a prepayment option, a user prepays funds for the right to obtain copies of the digital file. In a pay-per-copy (or "pay as you go") option, the user pays for each copy of the digital file when the file is copied. In an IOU scheme, the user makes copies of the digital file (e.g., while the client machine is not connected to the network) and generates an IOU (or many IOUs) that are then submitted to a clearinghouse or other payment entity when the user later goes online. Other payment schemes (such as a combination of the above options) may also be implemented.

The payment scheme is preferably defined in copy control information associated with the file and established by the author, publisher or some other third party. Thus, for example, the copy control information may also include a count of the number of permitted copies, a count of the number of permitted pay-per-copy versions, a count of the number of copies that may be made under an IOU payment option, copyright management information identifying the author, publisher and/or other license or use restrictions, information about a bank or other financial institution that handles use payments and their reconciliation, one or more expiration dates (after which copying is no longer permitted), and the like.

The copy control information associated with a given file thus defines a usage scheme for the file because it includes information that controls how the content may be used, how such use is paid for, over what period the content may be used, and other such information. A particular usage scheme (or some portion thereof) may also be implemented in the devices between which the file is transferred, although preferably such restrictions are defined by the content provider.

According to the present invention as illustrated in **Figure 2**, the present invention assumes the existence of a pair of devices, a "source" **24** and a "target" **26**, that have been or are certified to use the scheme. In particular, devices that implement the inventive scheme preferably include a device certificate that is not accessible (and thus is free from tampering) and stored therein. The certificate evidences that the device

is capable of understanding a given security protocol useful in carrying out the protection scheme. A representative security protocol is CSS, or the Content Scrambling System protocol, available commercially from Matsushita Corp. Thus, for example, if the source device is a disk storage, the device certificate is typically stored inside a secure chip within the device control hardware. Typically, each of the devices is "certified" upon manufacture, although this is not a requirement.

As also illustrated in **Figure 2**, a channel **28** is established between the source and target devices over which copies of a digital file (that is subject to the scheme) are communicated in a secure fashion. Thus, prior to transfer of the digital file, the channel **28** is first established between the devices to ensure that the copy restrictions (such as set forth in the copy control information) may be enforced. Typically, this is accomplished by having each device (in accordance with the security protocol implemented) require the other device (of the pair) to verify that its device certificate is valid. An appropriate message exchange may be used for this purpose as defined in the protocol. Once the secure link has been established, each of the devices can be trusted to control the digital file in accordance with the file's copy control information.

Typically, the "source" **24** is a storage device while the "target" **26** is a rendering device. An illustrative storage device may be disk storage, system memory, or the like. An illustrative rendering device may be a printer, a display, a sound card or the like. The source and target devices may both be storage devices (e.g., a Web server and a client disk storage).

When the technique is implemented in an "open" client-server environment, hardware devices (e.g., microcontrollers) are used in the storage and rendering devices to facilitate generation and management of the secure link. When less security may be tolerated, some of these functions may be implemented in software. When the technique is implemented in a "closed" Web appliance environment (**Figures 6A-6D**), the secure link may be established in whole or in part using software resident in the control routines associated with the storage and rendering devices. The secure link may be established in software under such conditions because, in the Web appliance environment, it is possible to readily disable the secure link in the event of tampering with the appliance housing or other circuitry. Regardless of the environment, the secure link is first established between the "certified" storage and rendering devices. Thereafter, the digital file, together with at least part of its copy control information, is transferable between the storage and rendering devices in accordance with the particular usage scheme defined, for example, by the copy control information. Thus, for example, if a

prepayment scheme is implemented and an expiration date (associated therewith) has not occurred, a given number of copies of the file may be transferred between the storage and rendering devices.

Thus, as illustrated in **Figure 2** in simplified form, the digital file copy protection method and system of the present invention involves a "source" device **24** (or one or more of such devices), and a set of one or more "target" devices **26**a-n connected via the secure channel or link **28**. The physical characteristics of the channel, of course, depend on whether the source and target devices are located in the same machine or are in separate machines connected via a network. In a network connection, the link may be a conventional TCP/IP connection. Channel **28** may be a physically secure channel (such as a https connection), but this is not required as the given security protocol in the certified devices establishes a secure link. According to the invention, once the link is established, one or more digital files are transferred (under the control of a control routine or mechanism) between the certified devices in an predictable, auditable manner so that (a) a controlled number of file transfers can be made, and (b) the precise number of file transfers (and their particular use) may be readily documented to facilitate dissemination of royalties or some such other consideration, typically to providers of such content. Generalizing, prior to transfer of a given digital file (or set of files, or file component) from the source to the target via the secure link, that transfer must first be authorized, and the transfer itself is then capable of being associated with some royalty payment then due to a content provider for use of such file. The scheme thus facilitates implementation of a generalized copyright management/royalty collection and distribution scheme.

As previously mentioned, the source **24** and target **26** may be located on the same computer. **Figure 3** illustrates this particular connection for a disk storage subsystem **24'** and the target rendering devices, namely printer **26a'**, display **26b'** and sound card **26c'**. The illustrated computer is a Web appliance, in which case the secure link may be established (as noted above) using software. Thus, in this example, each source and/or target device includes appropriate control software (part of software **15** as described above) to facilitate creation of the secure channel. Although not meant to be limiting, one convenient mechanism to create the channel involves each of the devices to generate a random number **30**, which numbers are then supplied to a key generation algorithm **32** in a known manner to generate a secret of "private" key **34**. The key **34** may be generated for each digital file to be transferred over the link **28**, or a signal key may be used for a set of such files, or even for a particular browsing session. To create the secure channel, the software resident on the disk storage encrypts the digital file as it leaves the source device.

The target device then decrypts the digital file using the key prior to rendering. In this way, the digital file cannot be readily intercepted as it is being transferred between these devices. As noted above, each of the source and target devices may also include secure chips or other known hardware devices to facilitate or augment such secure transfer of the digital file between the devices.

The particular mechanism for securing the channel between the source and target may be quite varied, and the present invention contemplates the use of any now known or later-developed technique, system or method for securing such communications. Thus, for example, another technique that may be used would be a public key cryptosystem.

Figure 4 is a block diagram illustrating a representative copyright royalty management system implemented according to the present invention. In this system, it is assumed that client computers 40 access the computer network 42 (e.g., the public Internet, an intranet, an extranet, or other computer network) to obtain access to Web-like documents supported on Web servers 44. One or more management servers 46 are connectable to the system via an access provider 48, and a control management server 50 may be used to facilitate scaling of the architecture if required. Control management server 50 may be controlled by a regulatory or rights agency that has responsibility for managing collection and distribution of copyright royalties.

A given management server includes a database 52 and appropriate control routines 54 for establishing a royalty account 55 for content providers. It is envisioned (although not required) that given content providers will subscribe to a royalty collection service implemented by the present invention and perhaps pay a fee (e.g., a commission or service charge) for the service provided. A given content provider thus may subscribe to the service to receive royalty payments for the use of his or her copyrighted content by users of the client machines. To this end, control routines 54 are used to establish an account for each of a set of given content providers, with each account including a representation of a given royalty value (which may be $0 when the account is established). A control routine then adjusts the given royalty value in a given provider account in response to receipt of an indication that a given digital file associated with the given content provider has been transferred from a source 24 to a target rendering device 26 in a given client computer 40. Periodically, the content provider account is adjusted for any service or processing fees, and the remainder of the account is then distributed to the content provider. In the situation where the content provider is willing to allow his or her content (a given digital file) to be used with charges for such use paid later, a given bit may be set in the file's copy

control information indicating such preference. Other data in the copy control information may be used to set or control other content provider preferences with respect to use of the file within the context of the inventive scheme.

Figure 5 is a flowchart of one method of managing royalty account collection with respect to a particular digital file when a prepayment option is utilized. In this representative example, the digital file is an image (i.e. a .jpeg file) having a copyright owned by a given content proprietor or provider. Of course, the principles of the present invention are designed to be implemented collectively with many such digital files, and the following description is thus merely representative of one type of basic payment scheme. The routine assumes initially that a usage or payment account has been established for a given client computer (or a user of that computer). This is step **60** in the flowchart. It is also assumed that a royalty account has been established for the content provider at one of the management servers as previously described. This is step **62** in the flowchart. One of ordinary skill will appreciate that steps **60** and **62** need not be in any particular sequence. Step **60** typically involves the user prepaying some amount of funds into an account from which payments may be withdrawn, although this is not required.

At step **64**, a count is established by a control routine for the particular digital file. Typically, this is a count of a number of permitted copies of the digital file that may be transferred from the source to one or more target devices according to the present invention. This number, as noted above, is typically identified in the file's copy control information. The count is usually a positive integer, which is then decremented (by the control routine) down to zero as permitted or authorized copies are made. Alternatively, of course, the count may begin at zero (or any other arbitrary number), which is then incremented (by the control routine) to the threshold value identified in the copy count information. As noted above, the count may be set by the copyright proprietor, by a system operator, by a Webmaster, by hardware constraints, or by any other party or entity having authority and/or ability to set the count. Under certain circumstances, e.g., where a prepaid user account is used, it may be unnecessary to use an explicit count as the number of copies transferred may simply depend on the royalty assessed per copy. Thus, the "count" as used herein may be expressed explicitly or implicitly. The digital file may be stored on the client already, or it may be available from a Web server or other storage or archive. The particular location from which the digital file is sourced initially does not matter. Step **64** assumes, however, that the image is located already at the source device. If the file is not present at the source, it may be

necessary to obtain it (although, conceptually, the "source" may be broadly construed as the original or initial location of the file).

At step 66, a test is done repeatedly to determine whether a request for the image has been received. If not, the routine cycles on step 66 and waits for such a request. If the outcome of the test at step 66 is positive, then the routine continues at step 68 by testing whether the given client computer (which generated the request) is authorized to effect the transfer. Step 68 may comprise a simple comparison of the user's account balance and the royalty amount to be assessed. If the user's account balance is large enough, the transfer may be allowed. Or, step 68 may simply test whether the count has a value indicating that further copies may be made. More typically, step 68 will require that the count be non-zero (in the situation where the count is positive and decremented to zero) and the user have sufficient funds allocated to pay the royalty assessment for use of the image. The step 68 may also test whether a given expiration date set in the copy count information has past.

If the outcome of the test at step 68 is negative, the transfer is not authorized, and the routine branches to step 70 to so notify the user of the client machine. Such notification may be in the form of an error or "access denied" message or the like. The user may be informed merely that a preset expiration date has passed or that his or her prepaid account is exhausted and requires more funds. If, however, the outcome of the test at step 68 is positive, the digital file may be transferred to the target. The routine then branches to step 72 to initiate the copy transfer. Preferably, all bytes of the file must be transferred before the transfer is considered valid. At step 74, the control routine count is adjusted (e.g., decremented) and/or a given charge is allocated against the user's account. The given charge may be equal to the royalty or use charge, or some fixed percentage thereof (e.g., 105%) reflecting that royalty plus some service charge). At step 76, the appropriate content provider account is adjusted by the amount of the royalty payment (plus or minus appropriate service fees or other charges).

Neither step 74 nor step 76 need occur at the time of the file transfer. Typically, the account adjustments will take place in batch at a given time. Thus, for example, where the Web client is a Web appliance connected to the computer network via a dialup connection, the account information may be transferred to the management server upon establishing a given connection (e.g. perhaps once each day). Other variations regarding the timing of delivery of this information are, of course, within the scope of the present invention.

The present invention thus provides numerous advantages. Certified source and target devices first establish a secure link between themselves. Upon transfer of the file copy between source and target, the control routine records an appropriate indication thereof in the copy count, and the central authority is notified of the transfer of the digital file. Such notification may occur upon transfer of the digital file between the source and target devices, or at some later time (e.g., upon dialup connection of the computer to the network). Royalty accounts are then managed at a central authority; to facilitate distribution of royalties to content owners/publishers. When the copy count reaches the authorized limit (as set in the copy control information), the control routine destroys the file or otherwise prevents further copying of the digital file.

Thus, in one embodiment, the user establishes a "prepaid" account from which royalty or usage payments are drawn against as files are copied/transmitted. The system detects use of the file and, preferably, allows only a certain number of copies of the file to be made before the document is destroyed or otherwise rendered inaccessible (from the client machine). The resulting copyright management infrastructure is robust, secure, scaleable and easily managed.

In one embodiment of this invention as described above, the Internet client is a data processing system or a so-called "Web appliance" such as illustrated in **Figures 6A-6D** and 7. **Figure 6A** is a pictorial representation of the data processing system as a whole. Data processing system **100** in the depicted example provides, with minimal economic costs for hardware to the user, access to the Internet. Data processing system **100** includes a data processing unit **102**. Data processing unit **102** is preferably sized to fit in typical entertainment centers and provides all required functionality, which is conventionally found in personal computers, to enable a user to "browse" the Internet. Additionally, data processing unit **102** may provide other common functions such as serving as an answering machine or receiving facsimile transmissions.

Data processing unit **102** is connected to television **104** for display of graphical information. Television **104** may be any suitable television, although color televisions with an S-Video input will provide better presentations of the graphical information. Data processing unit **102** may be connected to television **104** through a standard coaxial cable connection. A remote control unit **106** allows a user to interact with and control data processing unit **102**. Remote control unit **106** allows a user to interact with and control data processing unit **102**. Remote control unit **106** emits infrared (IR) signals, preferably modulated at a different frequency than the normal television, stereo, and VCR infrared remote

control frequencies in order to avoid interference. Remote control unit **106** provides the functionality of a pointing device (such as a mouse, glidepoint, trackball or the like) in conventional personal computers, including the ability to move a cursor on a display and select items.

Figure 6B is a pictorial representation of the front panel of data processing unit **102**. The front panel includes an infrared window **108** for receiving signals from remote control unit **106** and for transmitting infrared signals. Data processing unit **102** may transmit infrared signals to be reflected off objects or surfaces, allowing data processing unit **102** to automatically control television **104** and other infrared remote controlled devices. Volume control **110** permits adjustment of the sound level emanating from a speaker within data processing unit **102** or from television **104**. A plurality of light-emitting diode (LED) indicators **112** provide an indication to the user of when data processing unit **102** is on, whether the user has messages, whether the modem/phone line is in use, or whether data processing unit **102** requires service.

Figure 6C is a pictorial representation of the rear panel of data processing unit **102**. A three wire (ground included) insulated power cord **114** passes through the rear panel. Standard telephone jacks **116** and **118** on the rear panel provide an input to a modem from the phone line and an output to a handset (not shown). The real panel also provides a standard computer keyboard connection **120**, mouse port **122**, computer monitor port **124**, printer port **126**, and an additional serial port **128**. These connections may be employed to allow data processing unit **102** to operate in the manner of a conventional personal computer. Game port **130** on the rear panel provides a connection for a joystick or other gaming control device (glove, etc.). Infrared extension jack **132** allows a cabled infrared LED to be utilized to transmit infrared signals. Microphone jack **134** allows an external microphone to be connected to data processing unit **102**.

Video connection **136**, a standard coaxial cable connector, connects to the video-in terminal of television **104** or a video cassette recorder (not shown). Left and right audio jacks **138** connect to the corresponding audio-in connectors on television **104** or to a stereo (not shown). If the user has S-Video input, then S-Video connection **140** may be used to connect to television **104** to provide a better picture than the composite signal. If television **104** has no video inputs, an external channel 3/4 modulator (not shown) may be connected in-line with the antenna connection.

Figure 6D is a pictorial representation of remote control unit **106**. Similar to a standard telephone keypad, remote control unit **106** includes buttons **142** for Arabic numerals 0 through 9, the asterisk or "star" symbol

(*), and the pound sign (#). Remote control unit also includes "TV" button **144** for selectively viewing television broadcasts and "Web" button **146** for initiating "browsing" of the Internet. Pressing "Web" button **146** will cause data processing unit **102** to initiate modem dial-up of the user's Internet service provider and display the start-up screen for an Internet browser.

A pointing device **147**, which is preferably a trackpoint or "button" pointing device, is included on remote control unit **106** and allows a user to manipulate a cursor on the display of television **104**. "Go" and "Back" buttons **148** and **150**, respectively, allow a user to select an option or return to a previous selection. "Help" button **151** causes context-sensitive help to be displayed or otherwise provided. "Menu" button **152** causes a context-sensitive menu of options to be displayed, and "Update" button **153** will update the options displayed based on the user's input, while home button **154** allows the user to return to a default display of options. "PgUp" and "PgDn" buttons **156** and **158** allows the user to change the context of the display in display-sized blocks rather than by scrolling. The message button **160** allows the user to retrieve messages.

In addition to, or in lieu of, remote control unit **106**, an infrared keyboard (not shown) with an integral pointing device may be used to control data processing unit **102**. The integral pointing device is preferably a trackpoint or button type of pointing device. A wired keyboard (also not shown) may also be used through keyboard connection **120**, and a wired pointing device such as a mouse or trackball may be used through mouse port **122**. When a user has one or more of the remote control unit **106**, infrared keyboard, wired keyboard and/or wired pointing device operable, the active device locks out all others until a prescribed period of inactivity has passed.

Referring now to **Figure 7**, a block diagram for the major components of data processing unit **102** is portrayed. As with conventional personal computers, data processing unit **102** includes a motherboard **202** containing a processor **204** and memory **206** connected to system bus **280**. Processor **205** is preferably at least a **486** class processor operating at or above **100** MHz. Memory **206** may include cache memory and/or video RAM. Processor **205**, memory **206**, and system bus **208** operate in the same manner as corresponding components in a conventional data processing system.

Video/TV converter **210**, located on motherboard **202** and connected to system bus **208**, generates computer video signals for computer monitors, a composite television signal, and an S-Video signal. The functionality of Video/TV converter **210** may be achieved through a Trident TVG9685 video

chip in conjunction with an Analog Devices AD722 converter chip. Video/TV converter 210 may require loading of special operating system device drivers.

Keyboard/remote control interface unit 212 on motherboard 202 receives keyboard codes through controller 214, regardless of whether a wired keyboard/pointing device or an infrared keyboard/remote control is being employed. Infrared remote control unit 106 transmits signals which are ultimately sent to the serial port as control signals generated by conventional mouse or pointing device movements. Two buttons on remote control unit 106 are interpreted identically to the two buttons on a conventional mouse, while the remainder of the buttons transmit signals corresponding to keystrokes on an infrared keyboard. Thus, remote control unit 106 has a subset of the function provided by an infrared keyboard.

Connectors/indicators 216 on motherboard 202 provide some of the connections and indicators on data processing unit 102 described above. Other connections are associated with and found on other components. For example, telephone jacks 116 and 118 are located on modem 222. The power indicator within connectors/indicators 216 is controlled by controller 214.

External to motherboard 202 in the depicted example are power supply 218, hard drive 220, modem 222 and speaker 224. Power supply 218 is a conventional power supply except that it receives a control signal from controller 214 which effects shut down of all power to motherboard 202, hard drive 220 and modem 222. Power supply 218, in response to a signal from controller 214, is capable of powering down and restarting data processing unit 102.

Controller 214 is preferably one or more of the 805x family controllers. Controller 214 receives and processes input from infrared remote control 106, infrared keyboard, wired keyboard, or wired mouse. When one keyboard or pointing device is used, all others are locked out (ignored) until none have been active for a prescribed period. Then the first keyboard or pointing device to generate activity locks out all others. Controller 214 also directly controls all LED indicators except that indicating modem use. As part of the failure recovery system, controller 214 specifies the boot sector selection during any power off-on cycle.

Hard drive 220 contains operating system and applications software for data processing unit 102, which preferably includes IBM DOS 7.0, a product of International Business Machines Corporation in Armonk, New York; an operating system 221 such as Windows 3.1 (or higher), a product

of Microsoft Corporation in Redmond, Washington; and a browser 223 such as Netscape Navigator (Version 1.0 or higher), a product of Netscape Communications Corporation in Mountain View, California. Hard drive 220 may also support an SMTP mechanism to provide electronic mail, an FTP mechanism to facilitate file transfers from Internet FTP sites, and other Internet protocol mechanisms, all in a known manner. Hard drive 220 is not generally accessible to the user of the Web appliance.

Modem 222 may be any suitable modem used in conventional data processing systems, but is preferably a 33.6 kbps modem supporting the V.42bis, V.34, V.17 Fax, MNP 1-5, and AT command sets. Modem 222 is connected to a physical communication link 227, which, in turn, in connected or connectable to the Internet (not shown).

Those skilled in the art will recognize that the components depicted in Figures 6A-6D and 7 and described above may be varied for specific applications or embodiments. Such variations in which the present invention may be implemented are considered to be within the spirit and scope of the present invention.

According to the invention, the client machine (typically the hard drive 220) also includes a proxy 225. Preferably, the proxy is implemented in software and includes a cache 227 associated therewith. The cache may be integral to the proxy or logically associated therewith. The cache preferably has a size up to several hundred megabytes, which is substantially larger than the standard cache associated with a browser such as Netscape Navigator. The client machine also includes a protocol stack 229 (e.g., a TCP/IP protocol stack) and a sockets mechanism 231, which are used to support communications in a known manner. According to the invention, the proxy 225 is advantageously located on the client along with the browser. Thus, the proxy is sometimes referred to as a "client side" proxy.

Preferably, the proxy starts up when the Web appliance is booted up. Connectivity between the proxy and the browser is achieved using the sockets mechanism by configuring the browser to pass the HTTP requests to the proxy. To send an HTTP GET request, the browser creates a packet (including the URL and other information) and then opens a socket using the sockets mechanism. The packet is then sent to the IP address/port number to service the HTTP request. Thus, when the browser issues an HTTP GET request, it binds to the socket and sends the request. The request is then intercepted and processed by the proxy instead of being sent directly over the network, all in the manner previously described.

Although in the preferred embodiment the client machine is a Web "appliance", this is not a requirement of the present invention. Thus, a client machine 10 may be a personal computer such as a desktop of notebook computer, e.g., an IBM® or IBM-compatible machine running under the OS/2® operating system, an IBM ThinkPad® machine, or some other Intel x86 or Pentium®-based computer running Windows 95 (or the like) operating system.

A representative server platform comprises an IBM RISC System/6000 computer (a reduced instruction set of so-called RISC-based workstation) running the AIX (Advanced Interactive Executive Version 4.1 and above) Operating System 21 and Server program(s) 22. The platform 20 also includes a graphical user interface (GUI) 23 for management and administration. It may also include an application programming interface (API) 24. HTTP GET requests are transferred from the client machine to the server platform, typically via the dial-up computer network, to obtain documents or objects formatted according to HTML or some other markup language. While the above platform is useful, any other suitable hardware/operating system/server software may be used.

One of the preferred implementations of the client side or server side mechanisms of the invention is as a set of instructions (program code) in a code module resident in the random access memory of the computer. Until required by the computer, the set of instructions may be stored in another computer memory, for example, in a hard disk drive, or in a removable memory such as an optical disk (for eventual use in a CD ROM) or floppy disk (for eventual use in a floppy disk drive), or downloaded via the Internet or other computer network.

In addition, although the various methods described are conveniently implemented in a general purpose computer selectively activated or reconfigured by software, one of ordinary skill in the art would also recognize that such methods may be carried out in hardware, in firmware, or in more specialized apparatus constructed to perform the required method steps.

As used herein, "Web client" should be broadly construed to mean any computer or component thereof directly or indirectly connected or connectable in any known or later-developed manner to a computer network, such as the Internet. The term "Web server" should also be broadly construed to mean a computer, computer platform, an adjunct to a computer or platform, or any component thereof. Of course, a "client" should be broadly construed to mean one who requests or gets the file, and "server" is the entity which downloads the file. Moreover, although the present invention is described in the context of the Hypertext Markup Language

(HTML), those of ordinary skill in the art will appreciate that the invention is applicable to alternative markup languages including, without limitation, SGML (Standard Generalized Markup Language) and XML (Extended Markup Language).

In addition, the term "Web appliance" should be broadly construed to cover the display system illustrated in **Figures 6A-6D**, as well as any other machine in which a browser application is associated with some television class or other display monitor. Moreover, while the preferred embodiment is illustrated in the context of a dial-up network, this is not a limitation of the present invention. There may be other "bottleneck" resources in a direct connect network that could be managed indirectly by using this approach.

CLAIMS

1. A method for managing use of a digital file, comprising the steps of:

establishing a secure link between a pair of devices, each of the devices being certified to operate under a given security protocol;

establishing a usage scheme defining one or more conditions under which the digital file may be transferred between the pair of devices; and

transferring one or more copies of the digital file over the secure link between the pair of devices in accordance with the established usage scheme.

2. The method as described in Claim 1 wherein the pair of devices include a storage device and a rendering device.

3. The method as described in Claim 2 wherein the storage device and the rendering device are located in a computer.

4. The method as described in Claim 2 wherein the storage device is located in a first computer and the rendering device is located in a second computer and the secure link is established over a computer network connecting the first and second computers.

5. The method as described in Claim 4 wherein the second computer is a personal computer and the rendering device includes circuitry for establishing the secure link.

6. The method as described in Claim 4 wherein the second computer is a Web appliance and the rendering device includes software for establishing the secure link.

7. The method as described in Claim 2 wherein the rendering device is selected from a group of rendering devices consisting essentially of a printer, a display, and a sound card.

8. The method as described in Claim 1 further including the step of establishing an account representing a given monetary value.

9. The method as described in Claim 8 further including the step of allocating a given charge against the given monetary value when a copy of the digital file is transferred between the pair of devices.

10. The method as described in Claim 9 further including the step of associating the given charge with a content provider account to facilitate the payment of the given consideration to the provider of the digital file.

11. The method as described in Claim 1 wherein the usage scheme includes a given payment method.

12. A method for managing use of digital material in a computer network, comprising the steps of:

establishing an account for a given client computer including a representation of a given monetary value;

establishing an account for a given content provider including a representation of a given royalty value;

establishing a count of a number of permitted copies of a digital file;

in response to a given protocol, transferring a copy of the digital file from a source to a target associated with the given client computer;

adjusting the given monetary value in the account of the given client computer; and

adjusting the given royalty value in the account of the given content provider.

13. The method as described in Claim 12 wherein the given protocol includes the steps of:

determining whether a given client computer requesting transfer of the digital file is authorized to effect the transfer;

if the client is authorized to effect the transfer of the digital file, determining whether the count has a given value; and

if the count has the given value, transferring the digital file from the source to the target.

14. The method as described in Claim 13 wherein the given value is a non-zero value.

15. The method as described in Claim 13 wherein the given protocol further includes the step of adjusting the count after a copy of the digital file has been transferred.

16. The method as described in Claim 15 wherein the count is decremented.

17. The method as described in Claim 12 wherein the source and target are located in the given client computer connected to the computer network.

18. The method as described in Claim 17 wherein the source is a disk storage device and the target is a device selected from a group of rendering devices consisting essentially of a printer, a display, and a sound card.

19. The method as described in Claim 12 wherein the source is located on a first computer and the target is located on a second computer connected to the first computer via the computer network.

20. A method for managing use of digital material in a computer network including a Web client connectable to a Web server, comprising the steps of:

establishing a count of a number of permitted copies of a digital file located at a source device in the Web client;

in response to a given protocol, transferring one or more copies of the digital file from the source device to a set of one or more target rendering devices in the Web client; and

for each such transfer from the source device to one of the target rendering devices, logging an indication that the digital file has been transferred to facilitate payment of a given consideration to a provider of the digital file.

21. The method as described in Claim 20 wherein the Web client is a Web appliance and the source device is a secure disk storage.

22. The method as described in Claim 21 wherein each target rendering device is a device selected from a group of target rendering devices consisting essentially of a printer, a display, and a sound card.

23. The method as described in Claim 20 wherein the Web client is connected to the Web server via a non-secure connection.

24. The method as described in Claim 23 wherein the given protocol further includes the step of establishing a secure channel between the source device and a target rendering device prior to transferring the digital file.

25. The method as described in Claim 24 wherein the step of establishing a secure channel includes generating a secret key shared by the source device and the target rendering device.

26. The method as described in Claim 25 wherein the source device encrypts the digital file with the secret key as the source device transfers the digital file to the target rendering device, and wherein the target rendering device decrypts the digital file with the secret key upon receipt.

27. A computer program product in computer-readable media for use in a Web client having a source device and one or more target rendering devices, the computer program product comprising:

means for establishing a count of a number of permitted copies of a digital file located at the source device;

means, responsive to a given protocol, for transferring one or more copies of the digital file from the source device to the one or more target rendering devices;

means, responsive to each transfer, for logging an indication that the digital file has been transferred to facilitate payment of a given consideration to a provider of the digital file; and

means responsive to the logging means for adjusting the count.

28. The computer program product as described in Claim 27 further including means responsive to a given occurrence for transferring the indication to a central authority.

29. The computer program product as described in Claim 28 wherein the given occurrence is establishing a dialup connection between the Web client and an Internet Service Provider.

30. A computer system connected to a computer network and including a source device and one or more target rendering devices, comprising:

a processor;

an operating system;

an application for managing use of digital material, comprising:

means for establishing a count of a number of permitted copies of a digital file located at the source device;

means, responsive to a given protocol, for transferring one or more copies of the digital file from the source device to the one or more target rendering devices;

means, responsive to each transfer, for logging an indication that the digital file has been transferred to facilitate payment of a given consideration to a provider of the digital file; and

means responsive to the logging means for adjusting the count.

31. The computer system as described in Claim 30 wherein the application further includes means for restricting transfer of the digital file when the count reaches a given value.

32. A data processing system, comprising:

a remote control unit; and

a base unit connectable to a monitor for providing Internet access under the control of the remote control unit, the base unit comprising:

a processor having an operating system;

a browser application run by the operating system;

a secure disk storage in which a digital file is stored;

one or more target rendering devices; and

means for restricting a number of copies of the digital file that may be transferred between the secure disk storage and the one or more target rendering devices.

33. The data processing system as described in Claim 32 wherein the restricting means includes means responsive to a given occurrence for transmitting an indication of a number of copies of the digital file that were transferred between the secure disk storage and the one or more target rendering devices during a given time interval.

34. The data processing system as described in Claim 33 wherein the given occurrence is a dialup connection of the data processing system to an Internet Service Provider.

35. A management server for use in managing collection and allocation of royalties among content providers, the management server connected in a computer network to an access provider servicing a plurality of Web client appliances receiving dialup access to Web content, the management server comprising:

means for establishing an account for each of set of given content providers, each account including a representation of a given royalty value; and

means for adjusting the given royalty value in the account of the given content provider in response to receipt of an indication that a given digital file associated with the given content provider has been transferred from a source to a target rendering device in a given Web client appliance.

36. A copy management system, comprising:

a first device and a second device, each of which is certified to operate under a given security protocol;

means for establishing a secure link between the first and second devices; and

means responsive to establishment of the secure link for managing transfer of a permitted number of copies of a digital file between the first and second devices in accordance with copy control information restrictions associated with the digital file.

FIG. 1

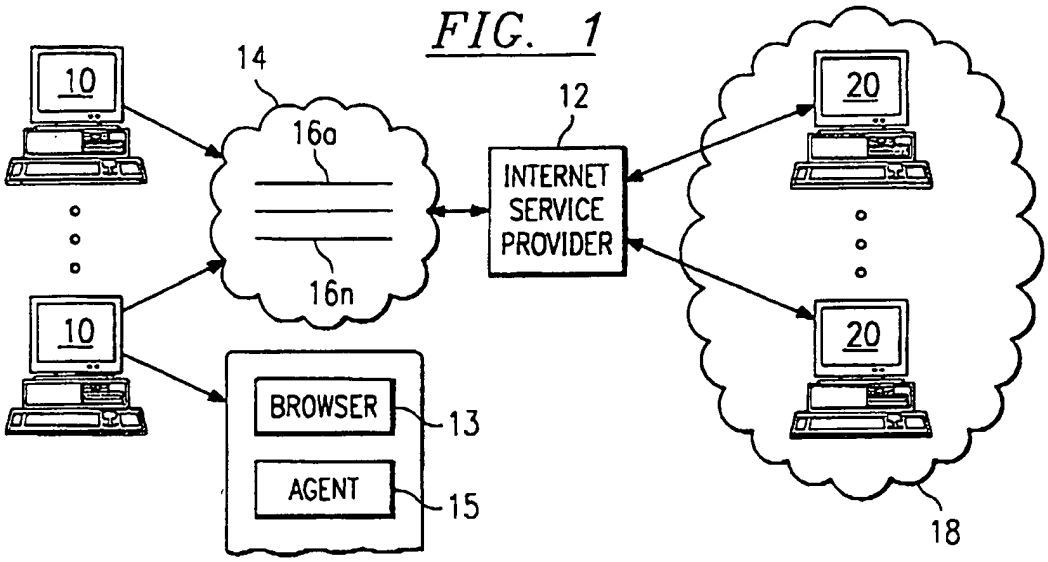


FIG. 2

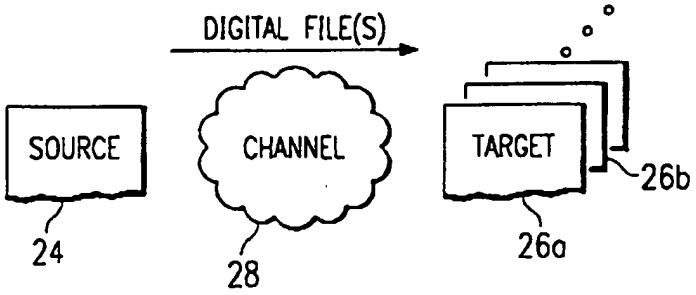


FIG. 3

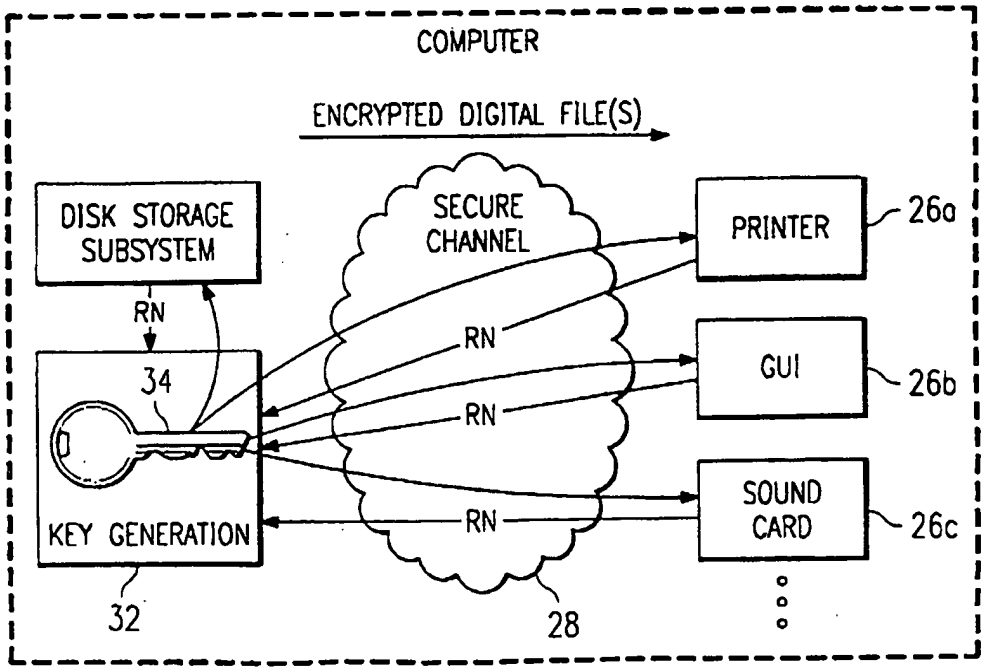
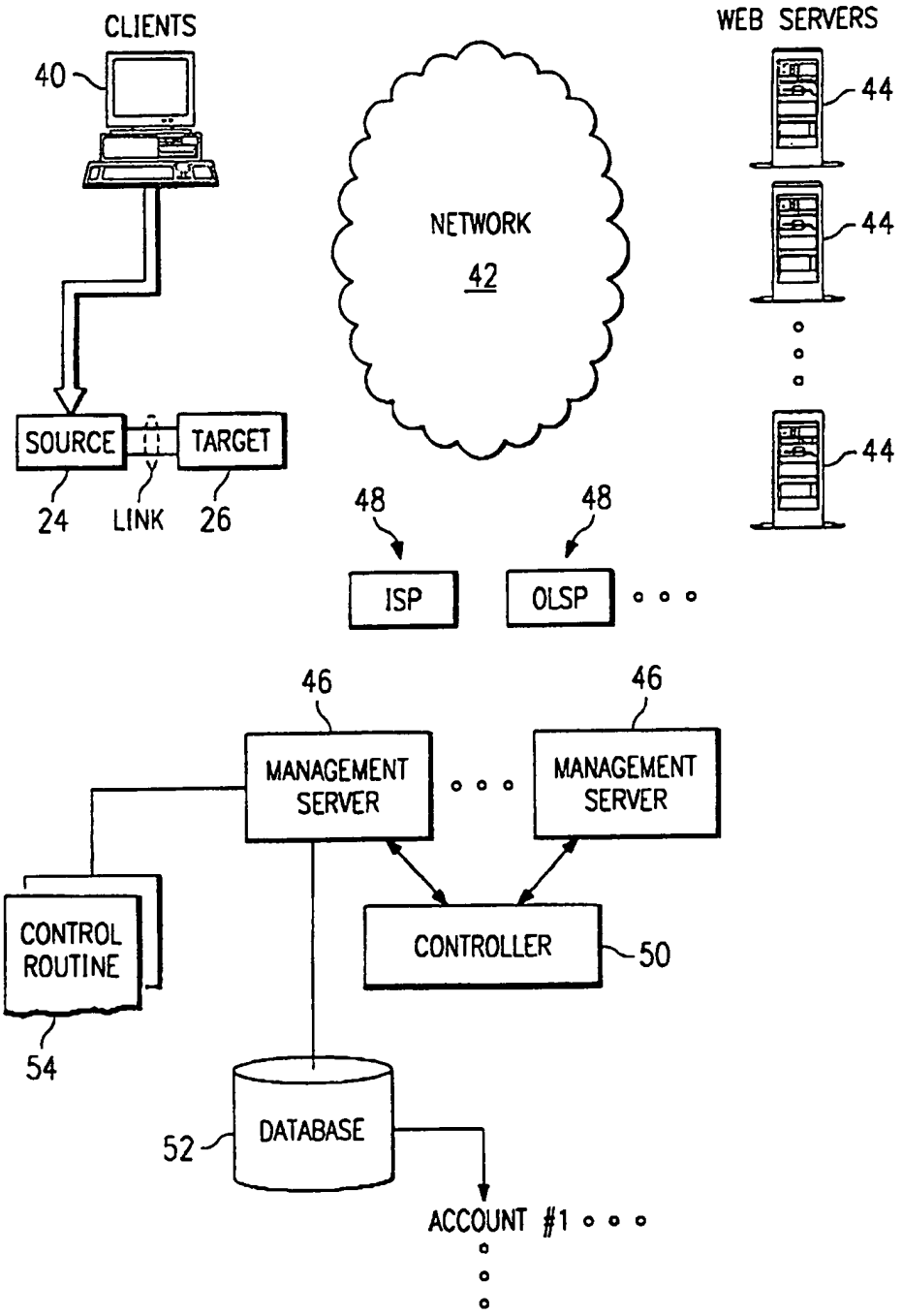


FIG. 4


CLIENTS
40
WEB SERVERS
44
44
44
NETWORK
42
SOURCE
TARGET
24
LINK
26
48
48
ISP
OLSP
46
46
MANAGEMENT SERVER
MANAGEMENT SERVER
CONTROL ROUTINE
54
CONTROLLER
50
DATABASE
52
ACCOUNT #1

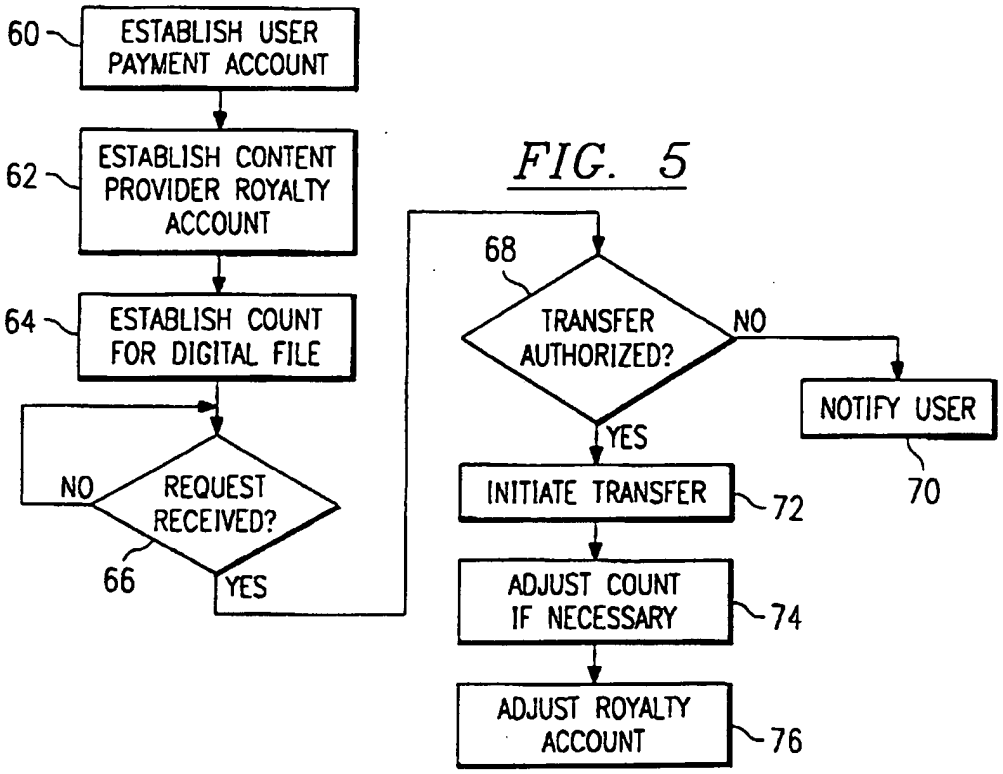

60
ESTABLISH USER PAYMENT ACCOUNT
62
ESTABLISH CONTENT PROVIDER ROYALTY ACCOUNT
64
ESTABLISH COUNT FOR DIGITAL FILE
NO
REQUEST RECEIVED?
66
YES
68
TRANSFER AUTHORIZED?
NO
NOTIFY USER
70
YES
INITIATE TRANSFER
72
ADJUST COUNT IF NECESSARY
74
ADJUST ROYALTY ACCOUNT
76

FIG. 5

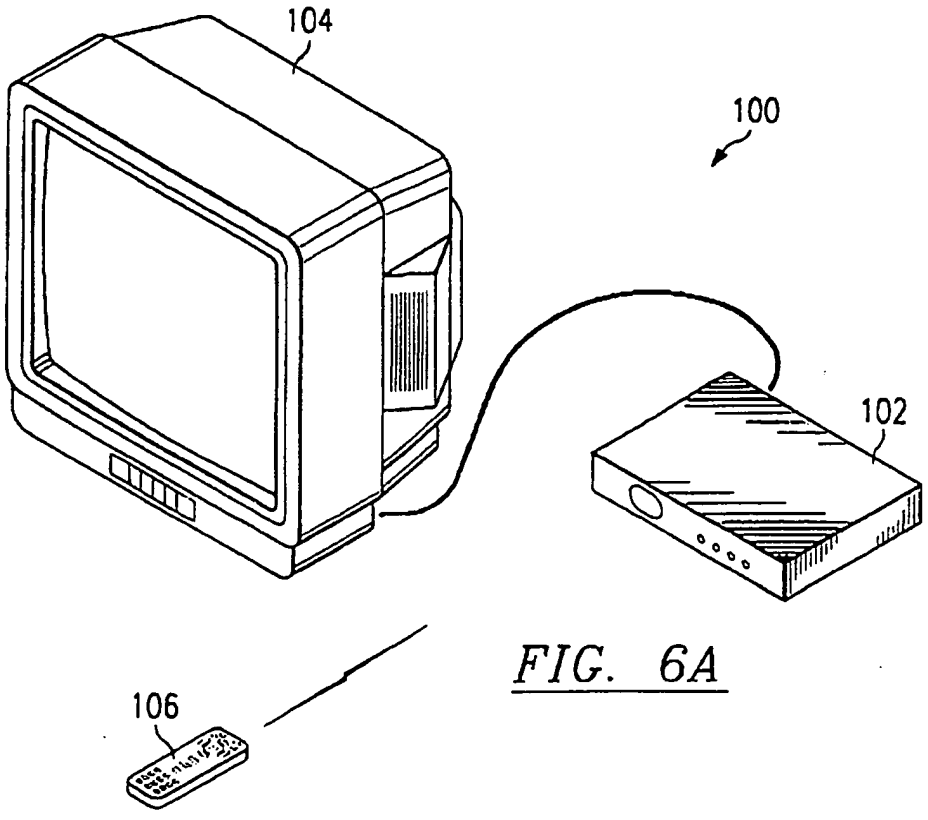

104
100
102
106

FIG. 6A

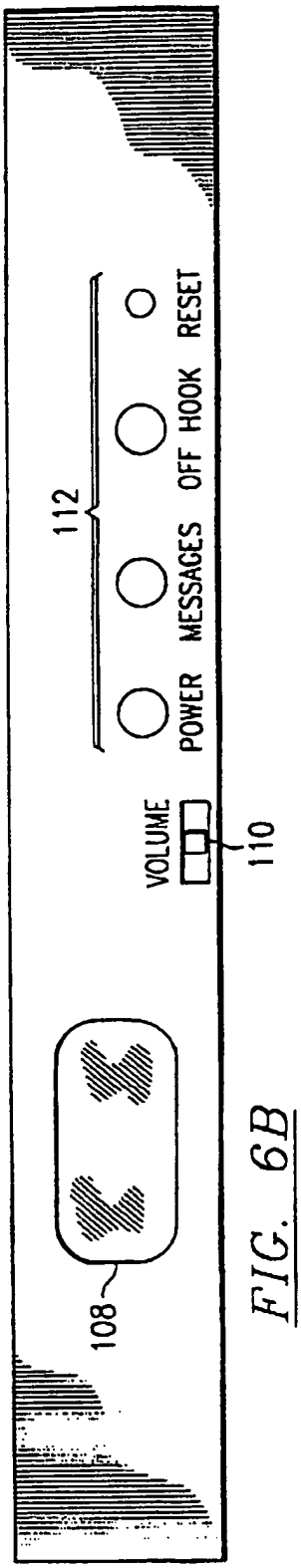

112
POWER
MESSAGES
OFF HOOK
RESET
VOLUME
110
108

FIG. 6B

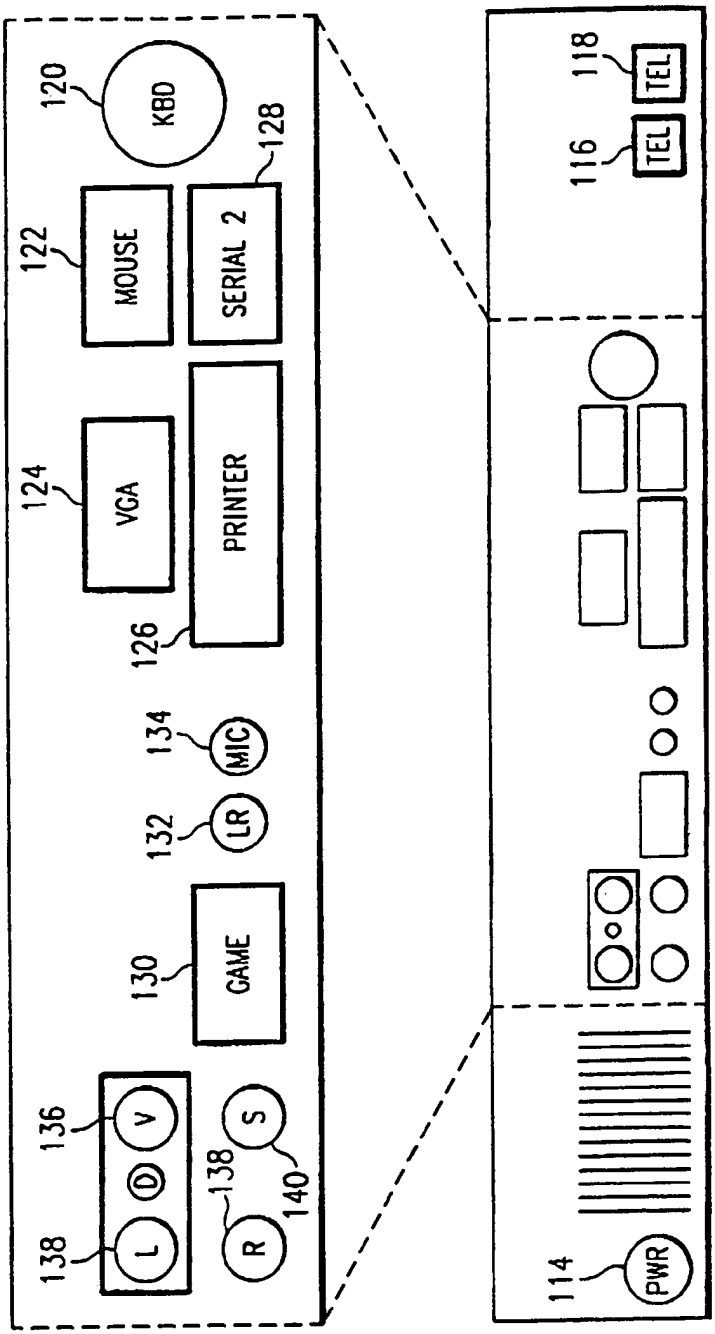

120
KBD
122
MOUSE
128
SERIAL 2
124
VGA
126
PRINTER
134
MIC
132
LR
130
GAME
136
V
138
L
138
R
140
S
118
TEL
116
TEL
114
PWR

FIG. 6C

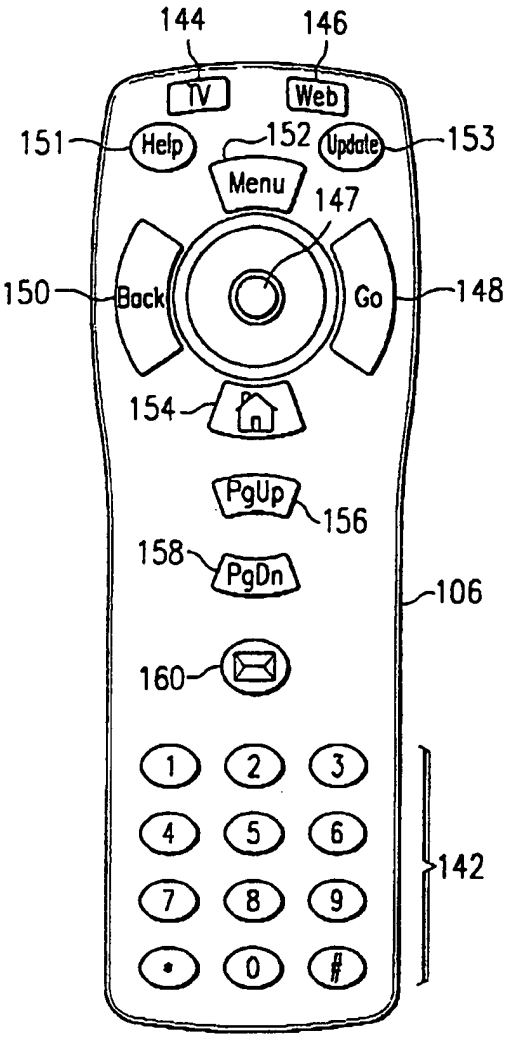
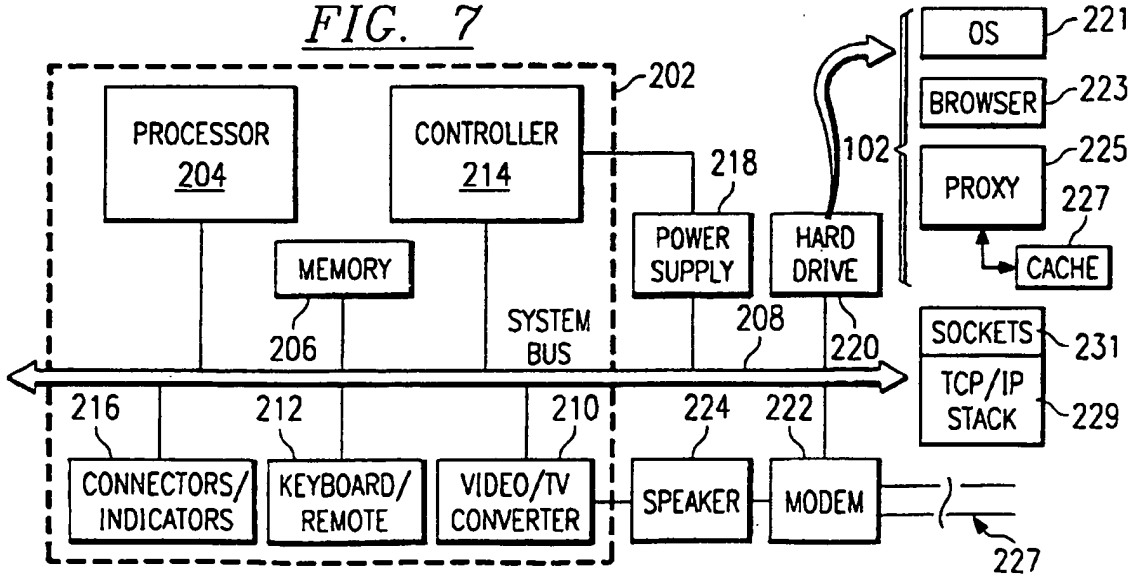

144
146
TV
Web
151
Help
152
Update
153
Menu
147
150
Back
Go
148
154
PgUp
156
158
PgDn
106
160
1
2
3
4
5
6
7
8
9
*
0
#
142

FIG. 6D

FIG. 7


202
PROCESSOR
204
CONTROLLER
214
218
102
OS
221
BROWSER
223
PROXY
225
227
CACHE
MEMORY
206
SYSTEM
BUS
POWER
SUPPLY
HARD
DRIVE
208
220
SOCKETS
231
TCP/IP
STACK
229
216
212
210
224
222
CONNECTORS/
INDICATORS
KEYBOARD/
REMOTE
VIDEO/TV
CONVERTER
SPEAKER
MODEM
227

INTERNATIONAL SEARCH REPORT

Int. ional Application No
PCT/GB 98/03828

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F1/00

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 5 532 920 A (HARTRICK THOMAS V ET AL) 2 July 1996 see figures 1,2 see column 6, line 44 - column 7, line 15 see column 14, line 51 - column 16, line 25	1-20, 23-25, 27-33, 35,36
Y	EP 0 798 906 A (SUN MICROSYSTEMS INC) 1 October 1997 see figures 1-3 see column 4, line 21 - column 5, line 33	1-20, 23-25, 27-33, 35,36

☐ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
31 March 1999	08/04/1999

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Weiss, P

1

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Information on patent family members

Int ional Application No
PCT/GB 98/03828

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5532920 A	02-07-1996	EP 0567800 A	03-11-1993
		JP 2659896 B	30-09-1997
		JP 6103286 A	15-04-1994
EP 0798906 A	01-10-1997	US 5761421 A	02-06-1998
		JP 10105529 A	24-04-1998

Form PCT/ISA/210 (patent family annex) (July 1992)

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : H04L 9/32

A2

(11) International Publication Number: **WO 99/60750**

(43) International Publication Date: 25 November 1999 (25.11.99)

(21) International Application Number: PCT/FI99/00432

(22) International Filing Date: 18 May 1999 (18.05.99)

(30) Priority Data:
981132 20 May 1998 (20.05.98) FI

(71) Applicant *(for all designated States except US)*: NOKIA NETWORKS OY [FI/FI]; Keilalahdentie 4, FIN–02150 Espoo (FI).

(72) Inventor; and
(75) Inventor/Applicant *(for US only)*: USKELA, Sami [FI/FI]; Puistokaari 8 B 12, FIN–00200 Helsinki (FI).

(74) Agent: KOLSTER OY AB; Iso Roobertinkatu 23, P.O. Box 148, FIN–00121 Helsinki (FI).

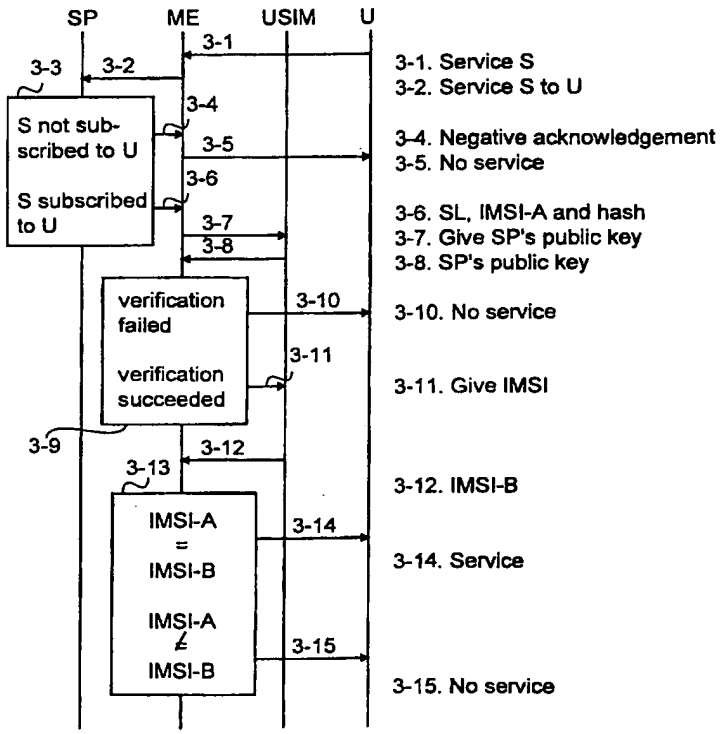
(81) Designated States: AE, AL, AM, AT, AT (Utility model), AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, CZ (Utility model), DE, DE (Utility model), DK, DK (Utility model), EE, EE (Utility model), ES, FI, FI (Utility model), GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SK (Utility model), SL, TJ, TM, TR, TT, UA, UG, US, UZ, VN, YU, ZA, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published
In English translation (filed in Finnish).
Without international search report and to be republished upon receipt of that report.

(54) Title: PREVENTING UNAUTHORIZED USE OF SERVICE

(57) **Abstract**

A method, a system, a network element and an apparatus of a telecommunication system for preventing unauthorized use of a service. The method, in which a service request is received from a user and the service is generated by means of a service logic, is characterized in that to prevent unauthorized use of the service, authentication data is appended to the service logic (3–6), the user requesting the service is authenticated by means of the authentication data (3–9), and the service logic is executed (3–14) only if the authentication succeeds.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

PREVENTING UNAUTHORIZED USE OF SERVICE

BACKGROUND OF THE INVENTION

The invention relates to preventing unauthorized use of services and especially to preventing unauthorized use of the services in a mobile communication system.

Mobile communication systems were developed, because there was a need to allow people to move away from fixed telephone terminals without affecting their reachability. The services offered through mobile stations have developed along with the mobile communication systems. At the moment, various new forms of service are being planned for the current and particularly for the future third-generation mobile communication systems, such as Universal Mobile Telecommunication System (UMTS) and International Mobile Telecommunication 2000 (IMT-2000). UMTS is being standardized by ETSI (European Telecommunications Standards Institute), whereas ITU (International Telecommunications Union) is standardizing the IMT-2000 system. These future systems are very similar in basic features. The following will describe in greater detail the IMT-2000 system whose architecture is illustrated in Figure 1.

Like all mobile communication systems, IMT-2000 produces wireless data transmission services to mobile users. The system supports roaming, in other words, IMT-2000 users can be reached and they can make calls anywhere within the IMT-2000 system coverage area. IMT-2000 is expected to fulfil the need for a wide range of future services, such as virtual home environment (VHE). With the virtual home environment, an IMT-2000 user has access to the same services everywhere within the coverage area of the system. According to present knowledge, a flexible implementation of various services and especially supporting roaming requires the loading of certain service logics into the terminal of the user and/or the serving network. A serving network is the network through which the service provider offers his service to the end-user. A service logic is a program, partial program, script or applet related to the service. The service is generated by means of the service logic by executing at least the service logic and the functions defined in it. A service can also comprise several service logics.

A problem with the arrangement described above is that it does not in any way verify that the user really has the right to use the service. It is

especially easy to copy and make unauthorized use of services in which the service logic is loaded into the terminal and/or serving network.

BRIEF DESCRIPTION OF THE INVENTION

Thus, it is an object of the invention to develop a method and an apparatus implementing the method so as to solve the above-mentioned problem. The object of the invention is achieved by a method, a system, a network element and an apparatus characterized by what is stated in the independent claims. The term apparatus refers here to a network element of the serving network, a terminal or any other corresponding service platform, into which the service logic can be loaded. The preferred embodiments of the invention are set forth in the dependent claims.

The invention is based on the idea of forming a service logic of two parts: user authentication and the actual service logic. The data required for user authentication is appended to the service logic, and the user is always authenticated before executing the actual service logic. This provides the advantage that an unauthorized use and copying of the service logic can be prevented. Only the users, to whom the service is subscribed and who thus have the right to use the service, can use it.

In a preferred embodiment of the invention, the service provider is always verified before the service is executed. This improves considerably the security of the user and a possible service platform into which the service logic is loaded. This ensures that the service logic truly originates from the service provider.

In a preferred embodiment of the invention, subscriber identification used to individualise a user is used in user authentication. This provides the advantage that subscriber authentication is simple, but reliable.

In a preferred embodiment of the invention, the service logic is saved with its user and authentication data in the memory of the service platform where it is loaded, and for a new user, only the authentication data of the new user is loaded. This provides the advantage that the service logic need not be loaded several times consecutively, which reduces the network load.

BRIEF DESCRIPTION OF THE DRAWINGS

In the following, the invention will be described in more detail in connection with preferred embodiments and with reference to the attached drawings in which

Figure 1 illustrates the IMT-2000 architecture,

Figure 2 shows a flow chart of the service platform functions in a first preferred embodiment of the invention,

Figure 3 is a signalling diagram of a second preferred embodiment of the invention, and

Figure 4 shows the operation of a network element controlling a service of a service provider in a third preferred embodiment of the invention.

DETAILED DESCRIPTION OF THE INVENTION

The present invention can be applied to any data transmission system in which the user can receive the subscribed services in any terminal supporting service provision. In the following, the invention will be described using the IMT-2000 system as an example, without limiting the invention to this particular system, however. The specifications of mobile communication systems in general and those of the IMT-2000 and UMTS system in particular evolve rapidly. This evolution may require extra changes to the invention. Therefore, all terms and expressions should be interpreted as widely as possible and they are intended to describe and not to limit the invention. It is the function that is essential for the invention and not in which network element or apparatus it is executed.

Figure 1 shows the network architecture of the IMT-2000 system on a general level, because the system specifications are currently being defined. A more detailed network structure bears no essential significance with regard to the invention. Third-generation mobile communication systems separate a service provider SP and a network operator from each other. A service provider offers services to an end-user through a network SN of one or more network operators. This type of network SN is called a serving network. A service provider can offer services through a serving network SN of one or more network operators. In addition, a service provider may switch to another serving network during the service without the user noticing it. A service provider can also be a network operator. A serving network SN comprises an actual access network AN, one or more core networks CN, and an

interworking unit adapting interfaces IWU for each different type of core network. According to present knowledge, an access network comprises base stations BS and radio network controllers RNC controlling them (not shown in the figure). A core network can be a network according to the pan-European mobile communication system GSM (Global System for Mobile Communication). Connections to other networks ON are established through a core network CN.

In the example in Figure 1, a home location register with IMT-2000 enhancement HLRi and the service control node SCN have been located in the serving network SN. The enhanced home location register HLRi contains not only the home register data of the core network but also the subscriber and service data required by the IMT-2000 system. The service provider SP maintains this IMT-2000 data for the part of the services. The subscriber makes an order agreement with the service provider which then charges the subscriber for the use of the services. The service control node SCN is a service platform to which the service logic related to the service can be loaded and in which it can be executed. The service control node SCN can also take care of loading the service elsewhere in the network and forward service requests from the user to the service provider. In addition to this, the service control node SCN makes sure that the services of the home network are also available in the visited networks.

In third-generation mobile communication networks, subscriber and user are also separated. The subscriber grants the user access to the subscribed services by giving the user an identification card (IC Card), for instance a USIM card (User and Services and Identity Module). The user accesses the services with a mobile terminal MT which is connected through base stations BS to a serving network SN over a radio path. A mobile terminal MT comprises actual mobile equipment ME and a detachably connected identification card USIM, also called a subscriber identity module. It is a smart card which can be detached from the mobile terminal and with which the subscriber can use a card-controlled mobile terminal. The user is identified by the card in the mobile terminal and not by the terminal itself. According to present knowledge, the USIM card is a multi-functional card and supports mobile communication system applications and other applications, such as Java applications, healthcare applications, etc. The subscriber can subscribe to the services of several different service providers with the same subscriber

identity module USIM. The subscriber and the user can be one and the same person. The subscriber identity module USIM also contains an international mobile subscriber identity IMSI with which the subscriber can be explicitly identified and which can also be used to identify the user. The identifier of a mobile subscriber is called subscriber identity.

The terminal selection of third-generation systems will probably be extremely versatile. The terminal can be a simplified terminal for speech only or it can be a terminal providing diverse services, which acts as a service platform and supports the loading and execution of various service logics.

A mobile communication system implementing the functionality of the present invention comprises not only means required for generating and loading services according to prior art, but also means for appending authentication data to a service logic and means for authenticating the user prior to executing the service logic. Here, appending also refers to embedding data into the service logic. In addition, the system can comprise means for verifying the service provider and means for saving the service logic with its supplementary data into the memory and means for receiving plain authentication data. The means for appending authentication data and the possible means for appending verification data are preferably located together with the means required for loading the service logic of the service provider. The other means are preferably located on the service platform, for instance in the terminal or the service control point of the network operator. The means or a part of them can also be located elsewhere, for instance in the network node of the subscriber network or in the serving support node of the core network.

Figure 2 shows a flow chart of the operation according to the first preferred embodiment of the invention on a service platform which can be actual mobile equipment ME or a service control node SCN, for instance. In the first preferred embodiment of the invention, an encryption technique, known per se, based on public keys is utilized in a novel and inventive manner. One such encryption technique is RSA (Rivest Shamir Adleman public-key cryptographic algorithm) which can be used for both encryption and digital signature. In the first preferred embodiment of the invention, at least the secret key of the subscriber and the public key of the service provider are saved in the subscriber identity module USIM. If the subscriber has several key pairs, the secret key of the pair, whose public key has been entered in the subscriber data of the user, is saved. Correspondingly, a service provider can

have several key pairs of which one, for instance, is saved in the subscriber identity module and information on the pair, whose key is saved in the identity module, is entered in the subscriber data. This ensures that the secret and public key of the same pair is used. In the first preferred embodiment of the invention, the service provider is verified by a digital signature. It is generated in the first embodiment of the invention by calculating a one-way hash (one-way hash function) from the service logic, which is then encrypted. This embodiment provides the unexpected advantage that in connection with the verification of the service provider, the fact whether the service logic has been changed, is also checked. If the service logic has been changed, the hash calculated from it also changes and the service provider verification does not succeed anymore.

With reference to Figure 2, a service request concerning a service S1 is received from a user U1 in step 200. In step 201, a check is made to see whether the service logic SL1 related to the service S1 is in the memory. If it is not, the service request is forwarded to the service provider in step 202. The service provider finds the actual service logic SL1 related to the service S1 and appends to the service logic authentication data A1 required for user authentication, which in the first preferred embodiment is the public key of the subscriber. After this, the service provider calculates the hash from the actual service logic and the authentication data and appends it as verification data V1 to the service logic and encrypts the thus created file with its own secret key. The file contains the verification data V1, the authentication file A1 and the actual service logic SL1. Alternatively, the service provider could encrypt the hash with its secret key, append the encrypted hash as the verification data V1 to the file and then encrypt the file with the public key of the user. After this, in step 203, the file, i.e. the actual service logic SL1 related to the service S1, the authentication data A1 appended to it for the user U1, and the verification data V1 calculated from them, is loaded onto the service platform. In step 204, the service logic SL1 is saved and in it, the authentication data A1 and the verification data V1 related to the user U1, and, of course, information on the user U1 to which the authentication data A1 and the verification data V1 are related. The data is stored in encrypted format in the memory. Then, in the first preferred embodiment, the public key of the service provider is requested from the subscriber identity module USIM in the terminal of the user in step 205 and received in step 206, after which the encryption of the service

logic SL1, the authentication data A1 and the verification data V1 is decrypted in step 207 using the received key. In embodiments in which the service provider only has one key pair, the information on the public key of the provider can already be on the service platform and need not be separately requested. When the encryptions have been decrypted, the service provider is verified by calculating a hash from the service logic and the authentication data in step 208 and by comparing the thus calculated hash with the verification data V1 in step 209. If they are the same, the verification of the service provider succeeds, and after this, a challenge, i.e. a character string, is selected in step 210. How the challenge is selected bears no significance with regard to the invention. A simple and safe solution is to use a random number generator, whereby the challenge is a random number. The selected challenge is encrypted in step 211 with the public key of the subscriber, i.e. the authentication data A1. After this, in step 212, the encrypted challenge is sent to the subscriber identity module USIM in the terminal of the user U1, which decrypts the encrypted challenge into plain text with the secret key of the subscriber and sends the plain text back to the service platform. In step 213, the service platform receives the plain text and in step 214, it compares the original challenge with the plain text. If the character strings are the same, user authentication succeeds and the actual service logic SL1 can be executed in step 215.

If it is detected in step 209 that the calculated hash is not the same as the verification data, the service provider verification fails or the service logic has been changed. In both cases, executing the service logic would be a security risk and, therefore, it is not executed, and in step 217, all data saved for the service S1, i.e. the service logic SL1 and all appended authentication and verification data with user data, is deleted from the memory.

If it is detected in step 214 that the challenge is not the same as the plain text, authentication does not succeed and the service logic is not executed, and in step 216, the authentication data A1, verification data V1 and information on the user U1 appended to the service logic SL1 of the service S1, is deleted from the memory. This way, the actual service logic SL1 need not be loaded next time, only the authentication data and verification data.

If it is detected in step 201 that the service logic SL1 related to the service S1 is in the memory, a check is made in step 218 to see if authentication and verification data for the user U1 is appended to it. If this

data, too, is in the memory, operation continues from step 205 where the public key of the service provider is requested from the subscriber identity module USIM. From step 205 onward, operation continues as described above. This way, network resources are saved, because the once loaded data need not be loaded again.

If it is detected in step 218 that no authentication and verification data for the user U1 is appended to the service logic SL1 in the memory, in step 219, the authentication and verification data for the service S1 is requested from the service provider for the user U1. The authentication data A1 and the verification data V1 are received in step 220, after which they and information on the user U1 are appended to the service logic SL1 in step 221. After this, operation continues from step 205 where the public key of the service provider is requested from the subscriber identity module USIM. From step 205 onward, operation continues as described above.

The service platform can be actual mobile equipment ME or a network element of the serving network, such as the service control node SCN. The memory where the data and service logics are saved can also be a cache memory. In embodiments in which the service logic is saved in the memory, the service platform can comprise means for deleting the service logic from the memory for predefined reasons, for instance after a certain time period.

In embodiment in which the service logic is not saved in the memory, steps 200, 202, 203 and 205 to 215 are executed. The data deletion described in steps 216 and 217 is not done, but the actual service logic is left unexecuted.

The steps described above in Figure 2 are not in absolute chronological order and some of the steps can be executed simultaneously or deviating from the given order. Other functions can also be executed between the steps. Some of the steps, such as the service provider verification, can also be left out. The essential thing is to authenticate the user before the actual loaded service logic is executed.

Figure 3 shows signalling according to a second preferred embodiment of the invention. In the second preferred embodiment, the subscriber identification IMSI is used as the authentication data. It is also assumed that the service logic is loaded into the actual mobile equipment ME and not saved in its memory.

With reference to Figure 3, user U sends information in message 3-1 to mobile equipment ME through the user interface requesting service S. The mobile equipment ME sends the service request through the serving network to service provider SP in message 3-2. The service request contains information on the required service S and the user U requesting the service. In step 3-3, the service provider checks if the service S is subscribed to the user U. If the service S is not subscribed to the user, the service provider sends through the serving network a negative acknowledgement to the service request in message 3-4 to the mobile equipment ME which forwards the information in message 3-5 through the user interface to the user U.

If the service S is subscribed to the user, the service provider retrieves the subscriber identification IMSI-A of the user U and the service logic SL related to the service S and calculates a hash from them. After this, the service provider encrypts the service logic and the related data (IMSI-A, hash) with the secret key of the service provider. In message 3-6, the service provider sends the service logic SL, the identification IMSI-A and the hash to the mobile equipment ME. In the second preferred embodiment, after receiving the message 3-6, the mobile equipment ME requests the public key of the service provider from the subscriber identity module USIM in message 3-7. The subscriber identity module USIM sends it to the mobile equipment in message 3-8, after which the mobile equipment ME verifies the service provider in step 3-9. The mobile equipment decrypts the encryption of the service logic SL, the subscriber identification A1 and the hash with the received public key and calculates a hash from the combination of the service logic and the subscriber identification IMSI-A. If the calculated hash is not the same as that received in message 3-6, the verification fails. In such a case, the service logic is not executed and the mobile equipment ME sends information on the verification failure through the user interface to the user U in message 3-10, saying, for instance, that the service is not available.

If the hash calculated in step 3-9 and the received hash are the same, the verification succeeds and the mobile equipment ME requests the subscriber identification IMSI of the user U from the subscriber identity module USIM in message 3-11. The subscriber identity module USIM retrieves the subscriber identification IMSI-B from its memory and sends it to the mobile equipment ME in message 3-12. In step 3-13, the mobile equipment authenticates the user by checking if the IMSI-A received from the service

provider is the same as the IMSI-B received from the identity module. If the user passes the authentication in step 3-9 (i.e. IMSI-A is the same as IMSI-B), the mobile equipment ME executes the actual service logic SL and provides the service through the user interface to the user U in messages 3-14. If the values of the subscriber identifications IMSI differ from each other, authentication fails. In such a case, the mobile equipment does not execute the actual service logic, but informs the user U through the user interface in message 3-10 that the authentication failed, saying, for instance, that the service is not available.

The signalling messages described above in connection with Figure 3 are for reference only and can contain several separate messages to forward the same information. In addition, the messages can also contain other information. The messages can also be freely combined. In embodiment in which the service provider is not verified, the messages 3-7, 3-8 and 3-10 related to verification and step 3-9 are left out. Depending on the service providers, core network and mobile equipment, other network elements, to which various functionalities have been distributed, can take part in the data transmission and signalling.

Figure 4 shows a flow chart of a network element controlling a service of a service provider in a third preferred embodiment of the invention. In the third preferred embodiment, authentication and verification are only performed when a service logic is loaded into a visited (visiting) network or mobile equipment. The visited network is a network whose network element, into which the service is loaded, is a network element belonging to a provider other than the service provider. The third preferred embodiment utilizes both public key encryption and symmetrical encryption, such as DES (Data Encryption Standard). The latter encryption technique is used when the service logic is loaded into the mobile equipment. A common key is saved for it in both the subscriber identity module and the subscriber data of the user. In addition, the public key of the service provider is saved in the subscriber identity module for the service logics to be loaded into visited networks. Only encryption of the service logic with the secret key of the service provider prior to sending the service logic to the serving network or encryption with the common key prior to loading it in the mobile equipment is used as signature.

With reference to Figure 4, in step 400, a service request concerning a service S2 is received from a user U2. In step 401, a check is

made to see if the user U2 subscribes to the service S2. If the user subscribes to the service S2, a check is made in step 402 to see if the service logic SL2 related to the service U2 requires loading into the mobile equipment ME of the user. If the service logic SL2 is loaded into the mobile equipment of the user, in step 403, a common key is retrieved from the subscriber data of the user U2 for encrypting the service logic SL2 in step 404. This common key is used both as the authentication data of the user and the verification data of the service provider. Nobody else should have any information on the common key in this case. The authentication and verification are performed in connection with the decryption of the service logic. The encrypted service logic SL2 is loaded into the mobile equipment ME in step 405. The user is authenticated and the service provider is verified in the mobile equipment, for instance by sending the encrypted service logic to the subscriber identity module USIM in the mobile equipment, which decrypts the service logic using the common key in its memory and sends the plain-text service logic to the mobile equipment. When the service logic has been executed, information concerning this is received in step 406, and the subscriber is charged for the use of the service in step 407.

If it is detected in step 402 that the service logic SL2 will not be loaded into the mobile equipment, a check is made in step 408 to see if the user U2 is in the home network area. If yes, the service logic SL2 is executed in step 409, after which operation continues from step 407 in which the user is charged for the use of the service.

If it is detected in step 408 that the user is not in the home network area, in the third preferred embodiment of the invention, the service logic SL2 must be loaded into the visited network. To do this, in step 410, the public key of the user U2 is retrieved from the subscriber data for appending it as authentication data to the service logic. In step 411, the public key of the user is appended to the service logic SL2, and they are encrypted using the secret key of the service provider in step 412. The encryption also acts as the verification data. If the service provider has several key pairs of public and secret keys, the secret key of the pair whose public key has been saved in the identity module of the user is used. The encrypted service logic, to which the authentication data is appended, is loaded into the visiting network in step 413. The network element of the visiting network verifies the service provider by decrypting the service logic using the public key of the service provider and

authenticates the user, for instance in the manner described in connection with Figure 2, after which the service logic is executed. When the service logic has been executed, information concerning this is received in step 406, and the subscriber is charged for the use of the service in step 407.

If it is detected in step 411 that the requested service is not subscribed to the user, information is transmitted in step 414 that the service is not available to the user.

Above, in connection with Figure 4, it was assumed that the authentication and verification succeeded. If this is not the case, the service logic is not executed and the subscriber not invoiced. The steps described above in connection with Figure 4 are not in absolute chronological order and some of the steps can be executed simultaneously or deviating from the given order. Other functions can also be executed between the steps. Some of the steps can also be left out. The essential thing is that the authentication data is in some way appended to the service logic being loaded.

In the above embodiments, the actual service logic has been changed to ensure that the authentication and verification are done. This has been done by adding to the service logic a part taking care of the authentication and verification, which is always executed before the service logic. In some embodiments, the service logic can only be changed to ensure the authentication. In some embodiments, there is no need to change the service logic, and the authentication data and the possible verification data are appended to the service logic as separate data, and the service platform makes sure that the authentication and the possible verification are done. In these embodiments, pre-encrypted service logics can be used, which reduces the load of the network element, because encryption is done only once.

It has been assumed above in connection with Figures 2, 3 and 4 that the service provider appends the authentication data to the service logic before the encryption. The authentication data can also be appended to a pre-encrypted service logic. In such a case, the serving network or mobile equipment can also be adapted to append the authentication data to the service logic, for instance by means of the user data provided in the service request. It has been presented above that the user is authenticated only after the verification. However, the order bears no significance with regard to the invention. The user can be authenticated before the service provider is verified in embodiment in which the service provider is also verified. The data and/or

service logic also need not be encrypted unless the encryption is used for authentication and/or verification. Other alternatives for authentication, verification and possible encryption than those described above in connection with the preferred embodiments can also be used. The preferred embodiments can also be combined. The essential thing is that the user is authenticated before executing the service logic at least when the service logic is loaded into the mobile equipment or visiting network. In embodiment in which the service logic is loaded into the mobile equipment, the encryption of the service logic with the public key of the subscriber can also be used as the authentication data. The subscriber is authenticated when the identity module USIM decrypts the encryption with the secret key of the subscriber. For security's sake, it is advantageous that USIM never sends even to the mobile equipment the secret key saved in it, and the decryption with the secret key is always performed in USIM. Other data for authentication and possible verification than used in the above examples can also be used. The requirements for the authentication data and possible verification data are adequate individualization, reliability and non-repudiation. Adequate individualization means that the data specifies the user at least by subscriber.

No hardware changes are required in the structure of the serving network. It comprises processors and memory that can be utilized in functions of the invention. All changes required for implementing the invention can instead be made as additional or updated software routines in the network elements into which the service logic is loaded. An example of such a network element is the service control node. Extra memory is also needed in the network element saving the loaded service logic with its supplementary data.

The structure of the service provider also requires no hardware changes. The service provider comprises processors and memory that can be utilized in functions of the invention. All changes required for implementing the invention can be made as additional or updated software routines to achieve the functionality of the invention. Extra memory may be needed depending on the embodiment of the invention. It is, however, limited to a small amount sufficient for saving the extra authentication data and the possible verification data.

The structure of the mobile equipment requires no hardware changes. It comprises processors and memory that can be utilized in functions of the invention. All changes required for implementing the invention can

instead be made as additional or updated software routines in the mobile equipment which is adapted to function as a service platform. If the service logic is saved in the mobile equipment, extra memory is also needed.

In the subscriber identity module USIM, the extra memory possibly needed for implementing the invention is limited to a small amount sufficient for saving the extra authentication data, the possible verification data and the decryption algorithms possibly needed.

It will be understood that the above description and the figures related to it are only presented for the purpose of illustrating the present invention. The various modifications and variations of the invention will be obvious to those skilled in the art without departing from the scope or spirit of the invention disclosed in the attached claims.

CLAIMS

1. A method for preventing unauthorized use of a service in a mobile communication system, in which method

a service request is received from a user of the service, and

the service is generated by means of a service logic,

characterized in that in the method

authentication data is appended to the service logic (3-6),

the user requesting the service is authenticated by means of the authentication data (3-9), and

the service logic is executed only if the authentication succeeds (3-14).

2. A method as claimed in claim 1, **characterized** in that

verification data of the service provider is also appended to the service logic,

the service provider is verified in connection with user authentication (3-13), and

the service logic is executed only if the verification also succeeds.

3. A method as claimed in claim 2, **characterized** in that

a first hash calculated from the service logic is used as verification data of the service logic,

the service logic is loaded onto a service platform where it is executed to generate the service,

the service provider is verified on the service platform by calculating a second hash from the service logic, and

if the first and the second hash are the same, the verification succeeds,

if the first and the second hash differ, the verification fails.

4. A method as claimed in claim 2, **characterized** in that

the signature of the service provider is used as the verification data,

the service logic is signed by encrypting it with the secret key of the service provider, and

the service provider is verified by decrypting the encryption of the service logic with the public key of the service provider.

5. A method as claimed in any one of the above claims, **characterized** in that

the secret key of the subscriber is saved in the subscriber identity module (USIM) of the user of the service,

the public key of the subscriber is used as the authentication data,

a challenge encrypted with the public key of the subscriber is sent to the subscriber identity module located in the mobile equipment of the user requesting the service (207),

the challenge is decrypted into plain text with the secret key of the subscriber in the identity module,

the plain text is received from the identity module (208),

a check is made to see if the unencrypted challenge and the plain text correspond to each other (209), and

if they correspond, the authentication succeeds, and

if they do not correspond, the authentication fails.

6. A method as claimed in claims 1, 2, 3 or 4, **characterized** in that

individual identity of the subscriber is used as the authentication data,

a service request is received from the user (3-1),

the individual subscriber identity related to the user is requested (3-7),

the requested identity is received (3-8),

a check is made to see if the authentication data and the requested identity correspond to each other (3-9), and

if they correspond, the authentication succeeds, and

if they do not correspond, the authentication fails.

7. A method as claimed in any one of the above claims, **characterized** in that

the service logic is loaded onto the service platform where it is executed to generate the service, and

the authentication data is appended to the service logic in connection with the loading.

8. A method as claimed in claim 7, **characterized** in that

the service logic, the authentication data appended to it, and the data indicating the user are saved on the service platform in connection with the loading (204),

a service request is received from the user,

a check is made to see if the service logic related to the requested service is saved on the service platform (201), and

if not, the service logic is loaded (203),

if yes,

- a check is made to see if authentication data has been saved for the user requesting the service (217), and

- if yes, the user is authenticated,

- if not,

-- authentication data is requested for the user (218),

-- the authentication data and the data indicating the user are saved in the service logic (220), and

-- the user is authenticated.

9. A telecommunication system comprising

a first part (SP) to produce the service for the user by means of a service logic, and

a second part to provide the service (SN, MT) to the user of the service,

in which system the first part (SP) is adapted to identify the user requesting the service and to check, if the service is subscribed to the user, and if the service is subscribed to the user, to generate the service by loading the service logic into the second part (SN, MT) which is adapted to provide the service by executing the loaded service logic,

characterized in that

the first part (SP) is adapted to append authentication data into the service logic being loaded for user authentication, and

the second part (SN, MT) is adapted to authenticate the user and to execute the service logic only in response to a successful authentication.

10. A system as claimed in claim 9, **characterized** in that

the first part (SP) is adapted to sign the service logic by encrypting it with an encryption key agreed with the second part, and

the second part (SN, MT) is adapted to verify the first part by decrypting the encryption of the service logic with a key corresponding to the agreed key and to execute the service logic only if the verification also succeeds.

11. A system as claimed in claim 9 or 10, **characterized** in that

the telecommunication system is a mobile communication system (IMT-2000) comprising at least one service provider and serving network,

the first part is the service provider (SP), and

the second part is the serving network (SN) comprising at least one network element (SCN).

12. A system as claimed in claim 9 or 10, **characterized** in that

the telecommunication system is a mobile communication system (IMT-2000) comprising at least one service provider (SP) and mobile terminal (MT) which is connected to the service provider through a serving network (SN) and which mobile terminal (MT) comprises in addition to actual mobile equipment (ME) a subscriber identity module (USIM) which is detachably connected to the mobile equipment,

the first part is the service provider (SP), and

the second part is the actual mobile equipment (ME).

13. A network element (SP) generating a telecommunication system service for a user, which produces the service by means of a service logic and which comprises means for identifying the user requesting the service and for checking if the service is subscribed to the user and for loading the service logic into the telecommunication system if the service is subscribed to the user,

characterized in that the network element (SP) comprises means for appending the authentication data to the service logic being loaded so that the user of the service is authenticated before the service logic is executed.

14. A network element (SP) as claimed in claim 13, **characterized** in that it comprises means for signing the service logic before it is loaded into the network.

15. A network element (SP) as claimed in claim 13 or 14, **characterized** in that it comprises a processor arranged to execute software routines, and said means have been implemented as software routines.

16. An apparatus of a telecommunication system, which apparatus comprises service logic executing means for providing a service from a service provider of a telecommunication system to a user of the service,

characterized in that the apparatus (SCN, ME) comprises

separation means for separating the authentication data of a user from a loaded service logic,

authentication means responsive to the separation means for user authentication, and

service logic execution means are adapted to be responsive to the authentication means.

17. An apparatus (SCN, ME) as claimed in claim 16, **characterized** in that

it comprises verification means for service provider verification by means of verification data in the loaded service logic, and

the service logic verification means are adapted to be responsive to the authentication means.

18. An apparatus (SCN, ME) as claimed in claim 16 or 17, **characterized** in that it comprises a processor arranged to execute software routines, and said means are implemented as software routines.

19. An apparatus as claimed in claim 16, 17 or 18, **characterized** in that it is a network element (SCN) of a mobile communication system, which is adapted to function as a service platform.

20. An apparatus as claimed in claim 16, 17 or 18, **characterized** in that it is the mobile equipment (ME) in a mobile communication system.

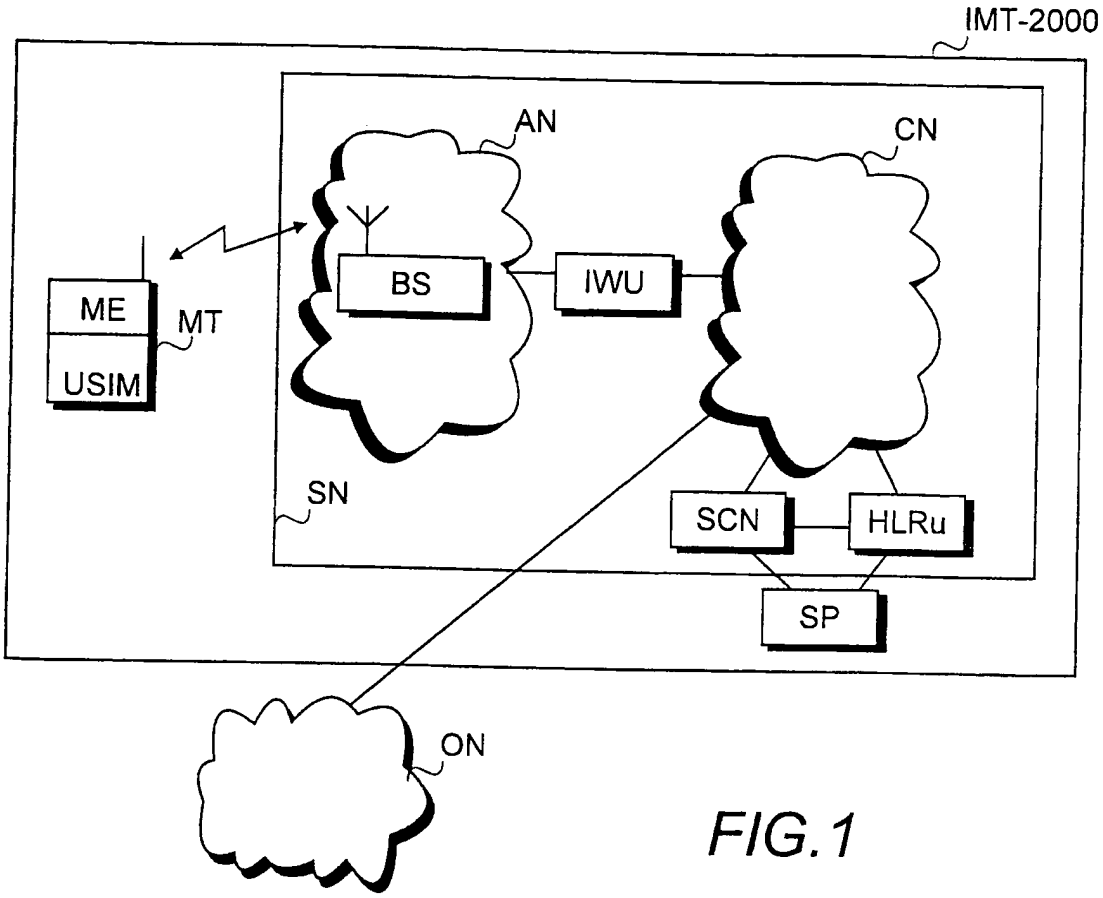


FIG.1

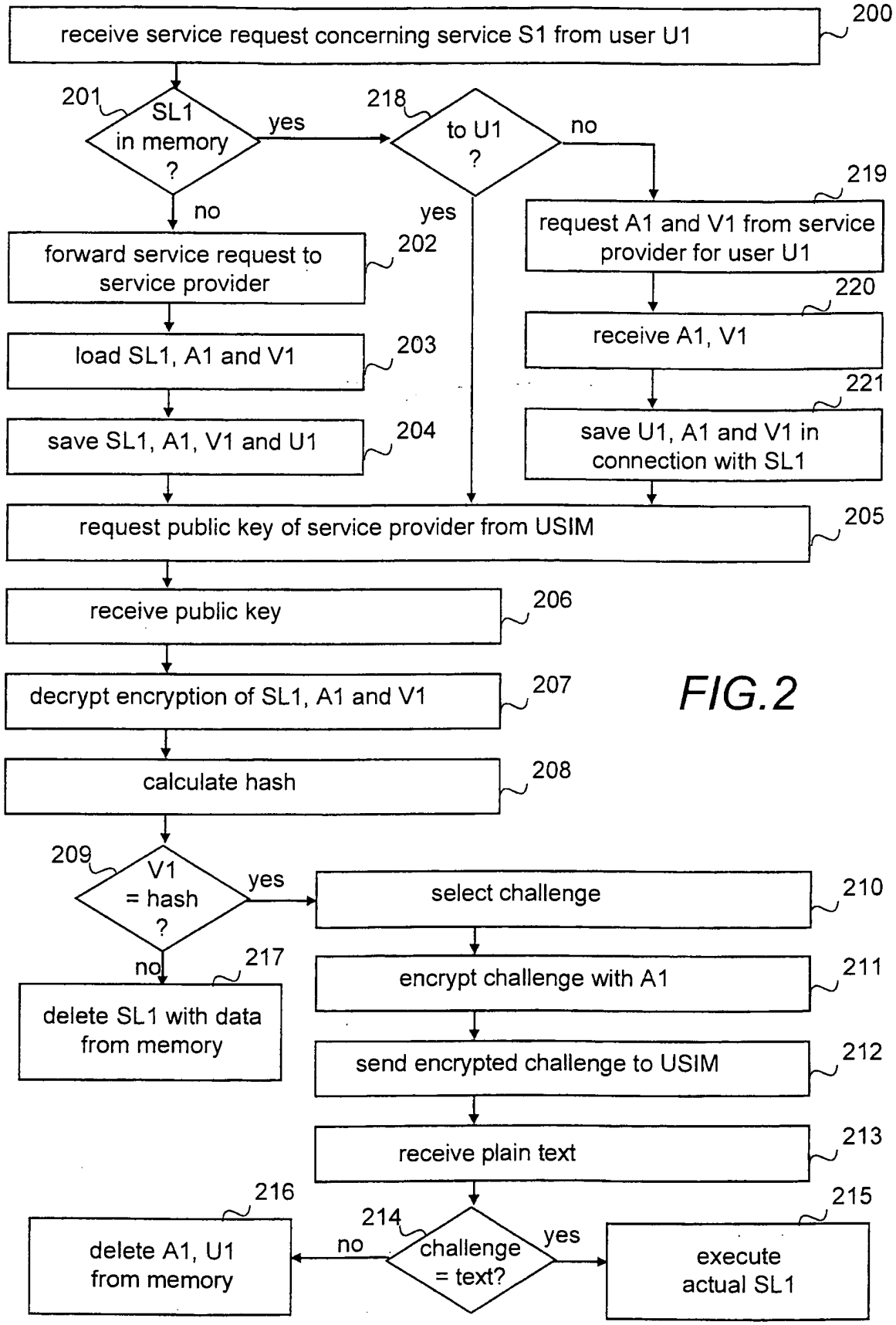

receive service request concerning service S1 from user U1
200
201
SL1 in memory ?
yes
218
to U1 ?
no
219
request A1 and V1 from service provider for user U1
no
yes
forward service request to service provider
202
220
receive A1, V1
load SL1, A1 and V1
203
221
save U1, A1 and V1 in connection with SL1
save SL1, A1, V1 and U1
204
request public key of service provider from USIM
205
receive public key
206
decrypt encryption of SL1, A1 and V1
207
calculate hash
208
209
V1 = hash ?
yes
select challenge
210
no
217
encrypt challenge with A1
211
delete SL1 with data from memory
send encrypted challenge to USIM
212
receive plain text
213
216
214
215
delete A1, U1 from memory
no
challenge = text?
yes
execute actual SL1

FIG.2

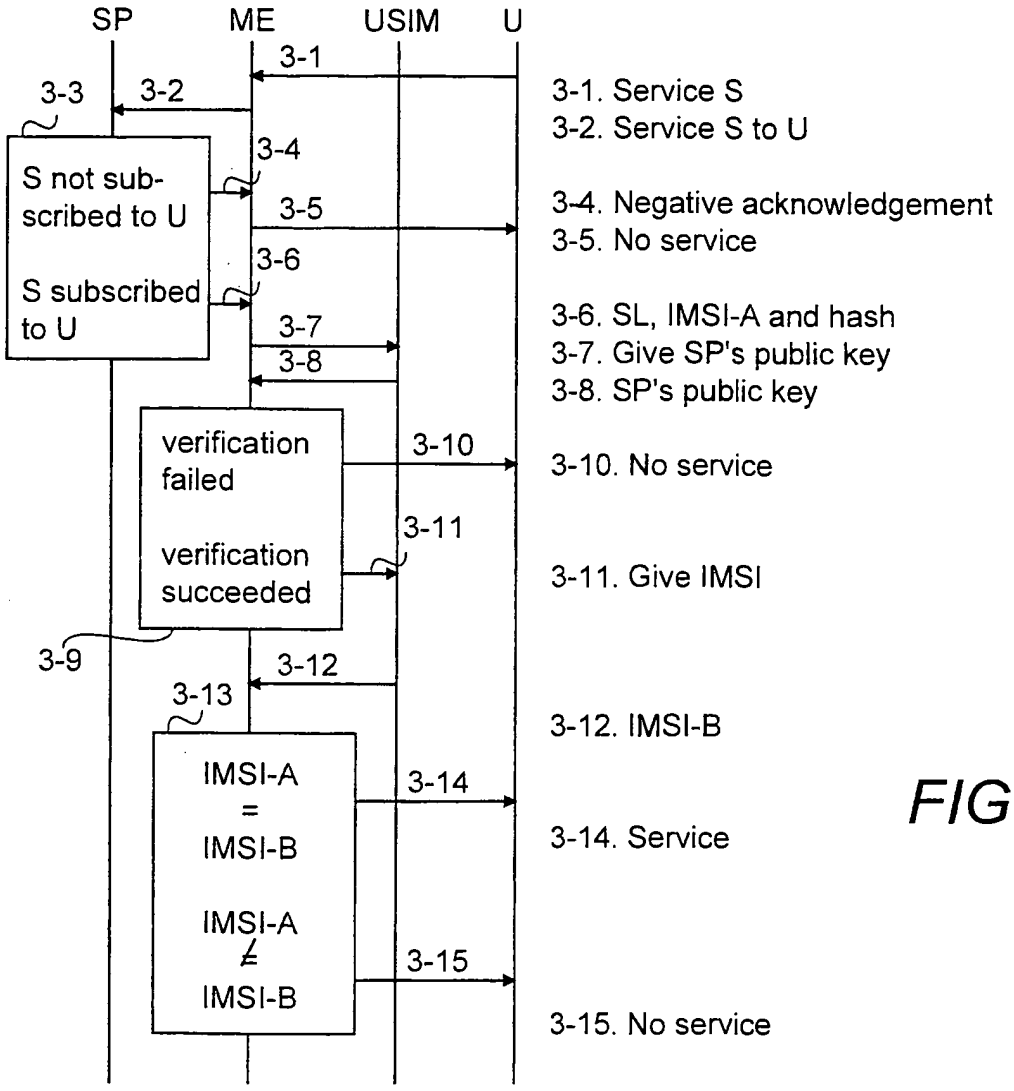

SP
ME
USIM
U
3-1
3-2
3-3
3-4
S not sub-scribed to U
3-5
3-6
S subscribed to U
3-7
3-8
verification failed
3-10
verification succeeded
3-11
3-9
3-12
3-13
IMSI-A = IMSI-B
3-14
IMSI-A ≠ IMSI-B
3-15
3-1. Service S
3-2. Service S to U
3-4. Negative acknowledgement
3-5. No service
3-6. SL, IMSI-A and hash
3-7. Give SP's public key
3-8. SP's public key
3-10. No service
3-11. Give IMSI
3-12. IMSI-B
3-14. Service
3-15. No service

FIG.3

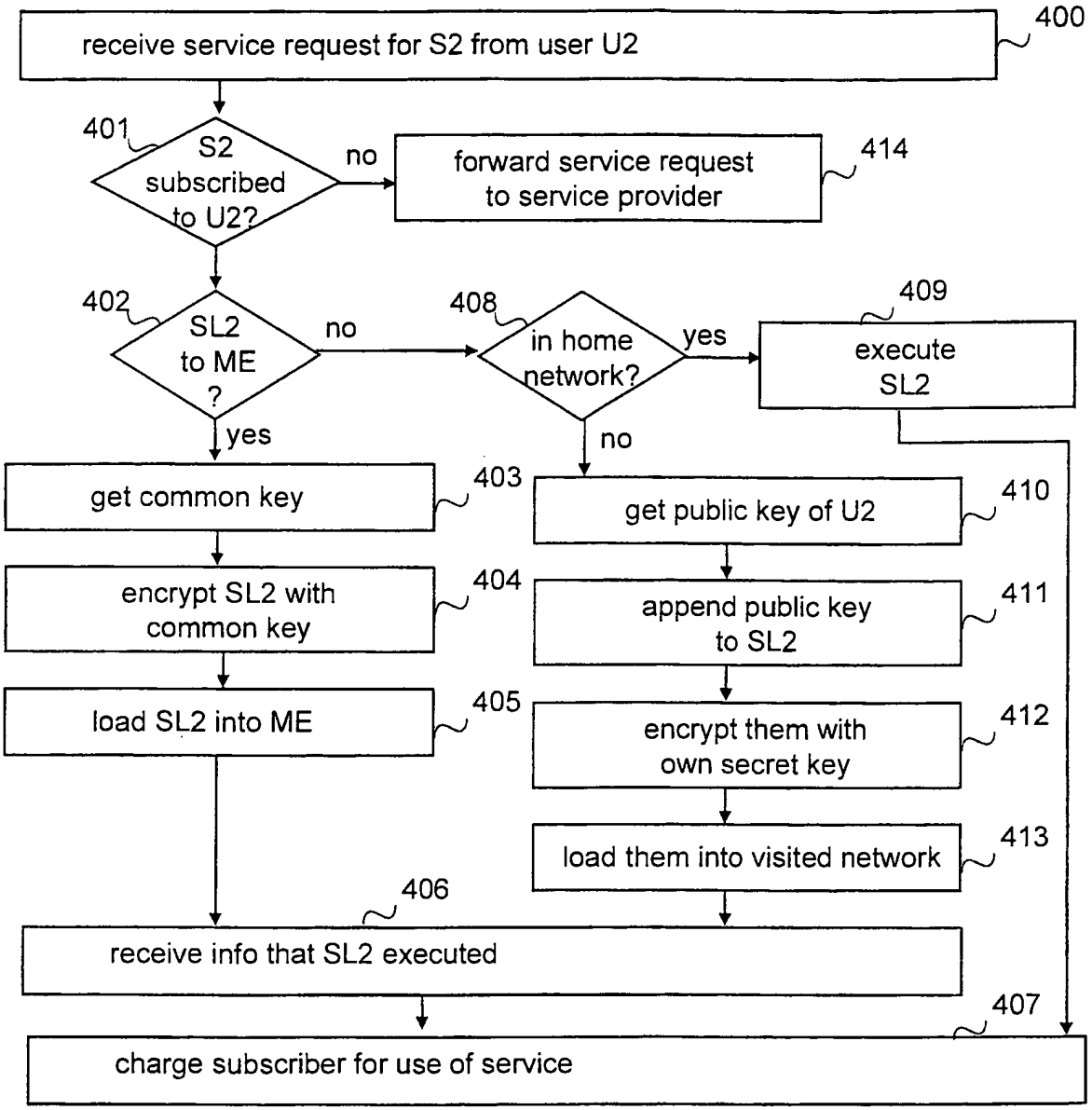

receive service request for S2 from user U2
400
401
S2 subscribed to U2?
no
forward service request to service provider
414
402
SL2 to ME ?
no
408
in home network?
yes
409
execute SL2
yes
no
get common key
403
get public key of U2
410
encrypt SL2 with common key
404
append public key to SL2
411
load SL2 into ME
405
encrypt them with own secret key
412
load them into visited network
413
406
receive info that SL2 executed
407
charge subscriber for use of service

FIG.4

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 7 : H04N 7 /24 — A2

(11) International Publication Number: **WO 00/04727**

(43) International Publication Date: 27 January 2000 (27.01.00)

(21) International Application Number: PCT/EP99/04704

(22) International Filing Date: 2 July 1999 (02.07.99)

(30) Priority Data:
60/092,726 14 July 1998 (14.07.98) US
09/276,437 25 March 1999 (25.03.99) US

(71) Applicant: KONINKLIJKE PHILIPS ELECTRONICS N.V. [NL/NL]; Groenewoudseweg 1, NL-5621 BA Eindhoven (NL).

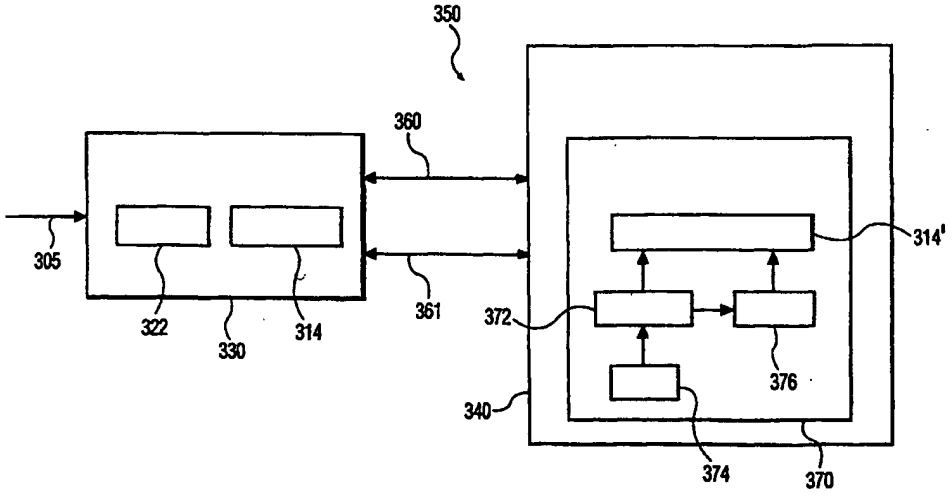
(72) Inventor: EPSTEIN, Michael, A.; Prof. Holstlaan 6, NL-5656 AA Eindhoven (NL).

(74) Agent: GROENENDAAL, Antonius, W., M.; Internationaal Octrooibureau B.V., Prof. Holstlaan 6, NL-5656 AA Eindhoven (NL).

(81) Designated States: BR, CN, JP, KR, MX, European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
Without international search report and to be republished upon receipt of that report.

(54) Title: USE OF A WATERMARK FOR THE PURPOSE OF COPY PROTECTION



(57) Abstract

A copyright protection system for protecting content wherein a time dependent ticket is calculated (314) at a source device (330) by combining a checkpoint with a ticket. The checkpoint is transmitted (361) from a display device (340) to the source device prior to the source device transmitting (360) watermarked content to the display device. The checkpoint is also stored (376) at the display device. Thereafter, the source device transmits, to the display device, watermarked content, the ticket, and the time dependent ticket. At the display device, the stored checkpoint is compared (314) to a current count of a local clock (374) that was utilized for producing the checkpoint. If the stored checkpoint is within a window of time of the local clock, then the stored checkpoint is combined (314') with the ticket in the same way that the checkpoint is combined with the ticket at the source device. A result of the combination is compared to the time dependent ticket and if the result equals the time dependent ticket, then the watermark and ticket may be compared in the usual way to determine the copy protection status of the copy protected content (314').

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

Use of a Watermark for the Purpose of Copy Protection.

Field of the Invention

This invention generally relates to a system for protecting copyrighted content. Specifically, the present invention pertains to utilizing a ticket and a watermark to protect content.

Background of the Invention

The ability to transmit digital information securely is increasingly important. Owners of content want to be able to provide the content to authorized users without having the content utilized by unauthorized users. However, one problem with digital content is that an exact copy can be made without any degradation in the quality of the copy. Therefore, the copying of digital content is very attractive to pirating operations or attackers.

Both small-scale and commercial pirates are interested in defeating copy-protected content in order to produce and sell illegal copies of the content. By avoiding payments to the rightful owner of the copy-protected content, the pirates may reap large profits. Typically, the pirate may take advantage of the difference in release windows in order access high value content and distribute it.

For instance, in the movie industry, release windows are utilized to maximize profit from content. The essence of these release windows is to first release the content to a premium service such as a pay-per-view service or a video on demand service. Thereafter, the content may be released on a lower price service such as a home-box-office service. At this time, the content may also be available to a consumer through a purchased storage medium such as a Digital Video Disc (DVD).

Pirates however, frustrate the use of these release windows by pirating the content that is available through the premium service and then releasing pirated versions of the content to the public. This may cause substantial financial losses to the rightful owners of the content. Accordingly, a successful copy protection scheme should at least frustrate a pirates attempt for a sufficient period of time till the legitimate owner of the content may reap their rightful profits.

Beyond some level of attacker, the expense of defeating the attacker exceeds a reasonable limit whereby the device must be priced beyond what consumer is willing to pay. Thus, a copy protection solution must be cost effective but secure against a large number of attackers.

A cost-effective method of copy protection is discussed in detail by Jean-Paul Linnartz et al., in Philips Electronics Response to Call for Proposals Issued by the Data Hiding Subgroup Copy Protection Technical Working Group, July 1997 ("Linnartz"). Within a digital transmission, such as an MPEG transport stream, additional data may be embedded within the transport stream to set the copy protection status of content contained within the digital transmission. For instance, the desired copy protection status may be "copy-once", "no-more-copy", "copy-never", and "copy-freely". Content that has a status of copy-once may be played and copied. During copying, the copy-once content is altered such that the content is in the no-more-copy state. Copy-never content is content that may only be played and may not be copied. Copy-freely content may be played and copied without restriction.

The additional data may take the form of a digital watermark. The watermark may be embedded directly into the content so that removal of the watermark will degrade the quality of the content. The watermark may be utilized as part of the copy protection scheme. As an example, the copy-freely state may be designated by the lack of a watermark within the content.

In operation, a transmission, such as a digital transmission, is sent from a source device and received by a receiving device. A source device is a device that is writing content onto a data bus, initiating a broadcast transmission, initiating a terrestrial transmission, etc. A sink device is a device that reads content from the data bus, etc.

Fig. 1 shows a typical system for the transmission of content. In Fig. 1, the source device is a broadcast initiator 101 that utilizes a transmitting antenna 102 to transmit content. The sink device is a broadcast receiver, such as a set-top-box (STB) 104 that utilizes a receiving antenna 103 for receiving the transmitted content. The STB 104 is shown connected to a display device 105, a player 106, and a player/recorder 107, through a bus 108. The term bus is utilized herein to refer to any system for connecting one device to another device. The bus may be a hard wired system such as a coaxial wire, an IEEE 1553 bus, etc., or the bus may be a wireless system such as an infra-red (IR) or radio frequency (RF) broadcast system. Several of the devices shown in Fig. 1 may at one time act as a source device and at another time act as a sink device. The STB 104 may be a sink for the broadcast transmission and be a

source for a transmission on the bus 108. The player/recorder 107 may be a source/sink of a transmission to/from, respectively, the bus 108.

In the copy protection scheme discussed by Linnartz, a watermark (W) is embedded within transmitted content. A ticket is transmitted along with the transmitted content. The embedded watermark and the ticket together are utilized to determine the copy protection status of the transmitted content. The watermark may be embedded into the content by at least two known methods. One method embeds the watermark (W) in the MPEG coding of the content. Another method embeds the watermark (W) in the pixel data of the content. The ticket (T) is mathematically related to the watermark (W) as discussed in more detail below.

Performing one or more one-way functions on the ticket (T) derives the watermark (W). By use of the term one-way function, what is meant is that it is computationally unfeasible to compute the inverse of the function. An example of a publicly known mathematical one-way function is a hashing function, such as secure hash algorithm one (SHA-1) or RACE Integrity Primitives Evaluation Message Digest (RIPEMD). Computing an inverse means finding which particular x_0 leads to a given y_0 with $y_0=F(x_0)$. The term unfeasible is intended to mean that the best method will take too long to be useful for a pirate. For instance, the time that is required for a pirate to compute the inverse of a hashing function is too long for the pirate to frustrate the intended release window for protected content. The most efficient method known to find such an x_0 may be to exhaustively search all possible bit combinations of x_0 and to compute and verify $F(x_0)$ for each attempt. In other cases, there may be a more efficient method than an exhaustive search to compute an inverse of a one-way function, yet these methods are still too time consuming to be feasible for the pirate.

The bit content of the ticket (T) is generated from a seed (U). The content owner provides the seed (U). From the seed (U), a physical mark (P) is created. The physical mark (P) may be embedded on a storage medium such as a Read-Only Memory (ROM) disk. Performing one or more one-way functions on the physical mark (P), produces the ticket (T). The number of functions performed on the physical mark (P) to create the ticket (T) depends on the copy protection intended for the content.

In accordance with the system, the ticket (T) changes state during every passage of a playback device (e.g., a source device) and a recording device (e.g., a sink device). As discussed above, the state modifications are mathematically irreversible and reduce the remaining copy and play rights of the content that are granted by the ticket (T). In this way,

the ticket (T) indicates the number of sequential playback and recordings that may still be performed and acts as a cryptographic counter that can be decremented but not incremented.

It should be noted that the copy protection scheme only protects content on compliant systems. A compliant system is any system that obeys the copy protection rules described above and hereinafter. A non-compliant system may be able to play and copy material irrespective of the copy protection rules. However, a compliant system should refuse to play copies of content illegally made on a non-compliant system.

In accordance with the copy protection scheme, a physical mark (P) (e.g., data) is embedded on a storage medium and is not accessible by other user equipment. The physical mark (P) data is generated at the time of manufacturing of the storage medium as described above and is attached to the storage medium in a way in which it is difficult to remove the physical mark (P) data without destroying the storage medium. The application of a one-way mathematical function, such as a hashing function, to the physical mark (P) data four times results in a watermark. Much like watermarks embedded in paper, the watermark is embedded in the medium (e.g., containing video, audio, or data) in such a way that it is infeasible to remove the watermark without destroying the material. At the same time the watermark should be imperceptible when the medium is used in the usual manner, such as when content from the medium is displayed.

A watermark by itself may indicate whether or not content stored on the storage medium is copy-once or copy-never. For instance, the absence of a watermark may indicate that the content may be copied freely. The presence of the watermark without a ticket on a storage medium may indicate copy-never content.

When the content is transmitted over a bus or other transmission medium, the physical mark (P) data is hashed twice to generate a ticket. When a compliant player receives the content, the ticket is hashed twice and matched to the watermark. In the case where the twice-hashed ticket and the watermark match, the content is played. In this way, a party may not substitute a false ticket along with the content to frustrate the copy protection scheme. In the case were there is a watermark but no ticket in the content, a compliant system will refuse to record the content.

When a compliant recorder reads the content, the watermark is checked to see if the material is copy-freely, copy-once, or copy-never. When there is no watermark, the content is copy-freely and may be copied freely as discussed above. When the content contains a watermark but no ticket, the content is copy-never and a compliant recorder will refuse to copy the content. However, a compliant player will play the content as long as the ticket

hashed two times matches the watermark. When the content is copy-once, the content contains both a watermark and a ticket, a compliant recorder will hash the ticket twice and compare the twice-hashed ticket to the watermark. In the case where the watermark matches the twice-hashed ticket, the content may be recorded along with a once-hashed ticket and the watermark, thereby creating copy-no-more content (e.g., content with a once-hashed ticket and a watermark). The physical mark will be different on a writable disc and thus, even if an illegal copy is made of copy-never content via a non-compliant recording device, a compliant player will refuse to play the content recorded on the writable disc.

It should be noted that in a broadcast system, such as a pay-per-view system, a copy-never state may be indicated by the presence of a once-hashed ticket and a watermark. Both copy-no-more stored content and copy-never broadcast content are treated by a compliant system similarly. The content containing the once-hashed ticket may be played but may not be recorded in a compliant system. In the event that a party tries to record the content with the once-hashed ticket, a compliant recorder will first twice-hash the once-hashed ticket and compare the result (e.g., a thrice-hashed ticket) with the watermark. Since the thrice-hashed ticket will not match the watermark, the compliant recorder will refuse to record the content.

A compliant player that receives the once-hashed ticket will hash the once-hashed ticket and compare the result (e.g., a twice-hashed ticket) to the watermark. Since the twice-hashed ticket matches the watermark, the compliant player will play the content.

However, a problem exists wherein a non-compliant recorder receives content containing a ticket (a twice-hashed physical mark) and a watermark. In the event that a non-compliant recorder does not alter the ticket upon receipt or recording (e.g., the non-compliant recorder makes a bit-for-bit copy), the non-compliant recorder may make multiple copies of the ticket and the watermark that will play on a compliant player and that may be recorded on a compliant recorder. The same problem can exist where a non-compliant recorder receives content containing a once-hashed ticket (a thrice-hashed physical mark) and a watermark indicating copy-no-more content. In this case, the non-compliant recorder may make multiple copies of the once-hashed ticket and the watermark that will play on the compliant player.

In a case wherein the player receives the content directly from a read only medium, such as a Compact Disc ROM (CD-ROM), a physical mark can be embedded in the physical medium of the CD-ROM that is produced by an authorized manufacturer. The player may then check the physical mark to ensure that the content is being received from an authorized medium. In this way, if a pirate makes an unauthorized copy, the physical mark

will not be present on the unauthorized copy and a compliant player will refuse to play the content. However, in the case of broadcast data for instance, wherein a player does not read content directly from the read-only medium, this method of copy protection is unavailable. Thus, for instance, a non-compliant player may deceive a compliant display device.

Accordingly, it is an object of the present invention to overcome the disadvantages of the prior art.

Summary of the Invention

This object of the present invention is achieved by a copy protection system for protecting content, such as content containing a watermark embedded therein (e.g., watermarked content). To this end, the invention provides a content protecting method, a copy protection system, a source device, and a display device as defined in the independent claims. The dependent claims define advantageous embodiments. In accordance with the present invention, a relative time dependent ticket is created at a source device preferably utilizing a display device dependent time reference (a checkpoint). In accordance with one embodiment of the present invention, the checkpoint is combined with a ticket utilizing a concatenation function and a one-way function (e.g., a hashing function). The checkpoint is transmitted from the display device to the source device prior to the source device transmitting watermarked content to the display device. The checkpoint is also stored at the display device. Thereafter, the source device transmits to the display device watermarked content, the ticket, and the relative time dependent ticket.

At the display device, the stored checkpoint is compared to a current relative time reference. If the difference between the stored checkpoint and the current relative time reference is acceptable, then further steps, as discussed below, may proceed. What is an acceptable difference between the stored checkpoint and the current relative time reference will depend on the nature of the desired content protection. For example, in one embodiment or for one particular type of content, the difference may be short to ensure that the content is being transmitted and received in real time. In another embodiment or for another type of content, the difference may be longer to allow for storage of the content for later playback.

When the difference between the stored checkpoint and the current relative time reference is acceptable, the ticket is next hashed twice and compared to the watermark in the usual way. In the event that the ticket compares to the watermark ($W = H(H(T))$), the stored checkpoint is combined with the ticket in the same way that the checkpoint was combined with the ticket at the source device. A result of the combination is compared to the relative

time dependent ticket. If the result equals the relative time dependent ticket, then the display device is provided with access (e.g., enabled to display) to the watermarked content.

Preferably, the checkpoint is derived from a counter that purposely is inaccurate such that the count can be said to be unique as compared to the count from other display devices. The counter is constructed with a sufficient number of bits such that the counter will not roll over to zero in the lifetime of the display device. The counter is constructed to only count up, such that the count may not be reversed and thereby, allow expired content to be displayed.

In yet another embodiment, a certificate containing the public key of the source device is sent to the display device prior to the above described process. A public key known to the display device may be used to verify the certificate. Preferably, the public key used to verify the certificate is built into the display device by the manufacturer of the display device. In this embodiment, the relative time dependent ticket (the checkpoint concatenated with the ticket) may be encrypted utilizing a private key of the source device. The encrypted relative time dependent ticket is then transmitted from the source device to the display device along with the watermarked content and the ticket. Thereafter, prior to the display device verifying the checkpoint, the display device decrypts the relative time dependent ticket utilizing a public key of the source device. In still yet another embodiment, the relative time dependent ticket may be signed (as is know in the art, by hashing the relative time dependent ticket and encrypting that hashed result) utilizing a private key of the source device. The resulting signature is sent along with the watermarked content, the relative time dependent ticket, and ticket to the display device. Thereafter, prior to the display device verifying the checkpoint, the display device verifies the signature on the relative time dependent ticket utilizing a public key of the source device.

Brief Description of the Drawings

The following are descriptions of embodiments of the present invention that when taken in conjunction with the following drawings will demonstrate the above noted features and advantages, as well as further ones. It should be expressly understood that the drawings are included for illustrative purposes and do not represent the scope of a present invention. The invention is best understood in conjunction with the accompanying drawings in which:

Fig. 1 shows a conventional system for the transmission of content;

Fig. 2 shows an illustrative communication network in accordance with an embodiment of the present invention; and

Fig. 3 shows details of an illustrative communication network in accordance with embodiment of the present invention wherein a source device provides content to a sink device.

Detailed Description of the Invention

Fig. 2 depicts an illustrative communication network 250 in accordance with an embodiment of the present invention. A source device 230, such as Set Top Box (STB), a Digital Video Disc (DVD), a Digital Video Cassette Recorder (DVCR), or another source of content, utilizes a transmission channel 260 to transmit content to a sink device 240. The transmission channel 260 may be a telephone network, a cable television network, a computer data network, a terrestrial broadcast system, a direct broadcast satellite network, some combination thereof, or some other suitable transmission system that is know in the art. As such, the transmission channel 260 may include RF transmitters, satellite transponders, optical fibers, coaxial cables, unshielded twisted pairs of wire, switches, in-line amplifiers, etc. The transmission channel 260 may also operate as a bi-directional transmission channel wherein signals may be transmitted from/to the source device 230, respectively, to/from the sink device 240. An additional transmission channel 261 may also be utilized between the source device 230 and the sink device 240. Typically, the transmission channel 260 is a wide-bandwidth channel that in addition to transmitting copy protection content (e.g., copy protection related messages), transmits copy protected content. The transmission channel 261 typically is a low-bandwidth channel that is utilized to transmit copy protection content.

The sink device 240 contains a memory 276 that is utilized for storing a checkpoint. The sink device 240 also contains a counter, such as a counter 272, that is utilized for generating the checkpoint. Preferably, the counter 272 should increment on a microsecond or better resolution as suitable for the application. The counter 272 should be free running. For instance, the counter 272 should count at all times that the sink device 240 is on. The bits of the counter 272 should employ non-volatile memory such as an electrically erasable programmable read-only memory (EEPROM) for the storage of the count. The counter 272 preferably is constructed to only count in one direction (e.g., up) and not in another direction (e.g., down). In a preferred embodiment, the counter 272 is driven by an inaccurate time source (e.g., inaccurate in terms of keeping time over hours, not necessarily over seconds), such as clock 274. The clock 274 is preferably unreliable so that drift with respect to time and

temperature is also non-negligible. Over time, this has the effect of randomizing the count of a counter for each sink device of a population of sink devices. In addition, the counter 272 may be driven fast for a random period of time to initialize the counter 272 to a random number at the time of manufacture. All of the above, has an effect of further randomizing the counter 272. The counter 272 is also configured such that it is inaccessible to a user. Accordingly, the user may not reset the counter 272.

The checkpoint, in accordance with the present invention, is transmitted to the source device 230 utilizing at least one of the transmission channels 260, 261. The source device 230 utilizes the checkpoint to change the ticket such that the watermarked content may only be utilized (e.g., played) by a corresponding sink device as described in more detail below. In the event that the corresponding sink device, such as the sink device 240, receives the watermarked content, then the content may be provided to a device, such as a display device 265, for display thereon. Preferably, the display device 265 is integral to the sink device 240 such that the display device 265 is the final arbiter in determining whether the copy protected content may be utilized. It should be obvious that although the device is illustratively shown as the display device 265, in fact the device may be any known device that may be suitably utilized for the copy protected content. For instance, in a case wherein the copy protected content is audio content, the device may be the device that outputs the audio signal.

In one embodiment of the present invention, the content may be provided from the source device 230 in the form of a Moving Picture Experts Group (MPEG) compliant transport stream, such as an MPEG-2 compliant transport stream. However, the present invention is not limited to the protection of an MPEP-2 compliant transport stream. As a person skilled in the art would readily appreciate, the present invention may be suitably employed with any other data stream that is known in the art for transmitting content.

In another embodiment, the source device 230 may be a conditional access (CA) device. In this embodiment, the transmission channel 260 is a conditional access module bus.

Fig. 3 depicts details of an illustrative communication network 350 in accordance with an embodiment of the present invention. In the communication network 350, a source device 330 provides content including copy protected content to a sink device 340 over a transmission channel 360. As discussed above with regard to the transmission channel 260, the transmission channel 360 may be a wide bandwidth transmission channel that may also have a bi-directional capability, such as a CA module bus.

The sink device 340 contains a copy protection status determination circuit 370 for creating/storing a checkpoint (C) and for determining the copy protection status of received content. The copy protection status determination circuit 370 contains a counter 372 and a clock 374 for creating the checkpoint (C). The counter 372 preferably contains a large number of bits (e.g., 64 bits for a clock 374 that increments on a millisecond basis). Preferably, the counter 372 should have a total count cycle time (the time required for the counter 372 to reach a top count from a bottom count) longer than a useful life of the sink device 340 (e.g., ten years). The clock 374 is preferably randomized (e.g., unreliable such that drift with respect to time and temperature is non-negligible) as discussed above with regard to the clock 274 shown in Fig. 2. The counter 372 is configured such that it is inaccessible and has no reset function even in the event of a removal of power. As such, the counter 372 may contain non-volatile storage, such as programmable read-only memory (PROM), electrically erasable PROM (EEPROM), static random access memory (static-RAM), etc. Further, the copy protection status determination circuit 370 contains a memory device 376 for storing the checkpoint (C).

In operation, the source device 330 may request the checkpoint (C) from the sink device 340 prior to transmitting copy protected content. In alternate embodiments, the sink device 340 may transmit the checkpoint (C) to the source device 330 as a portion of a request for the source device 330 to begin transmission of copy protected content to the sink device 340. The sink device 340 may utilize either of the transmission channels 360, 361 for transmission of the request for copy protected content and/or for transmission of the checkpoint (C). However, in some embodiments of the present invention, the transmission channel 360 may be unidirectional and may only be utilized for the transmission of content to the sink device 340 from the source device 330. In these embodiments, the transmission channel 361 is utilized for the transmission of the checkpoint (C) from the sink device 340 to the source device 330. The transmission channel 361 may also be utilized for transmitting a request for copy protected content from the sink device 340 to the source device 330.

In an alternate embodiment, the transmission channel 360 has bi-directional capability and may be utilized for transmissions both to and from the source device 330, and to and from the sink device 340. In this embodiment, the transmission channel 361 may not be present or it may be utilized solely for the transmission of content requiring low bandwidth. For instance, the source device 330 may utilize the transmission channel 361 to transmit to the sink device 340 a request for the transmission of the checkpoint (C).

In one particular embodiment, the source device 330 is a conditional access (CA) device 330, the transmission channel 360 is a CA module bus 360, and the sink device 340 is a display device 340. Prior to the transmission of copy protected content, the CA device 330 transmits a request for a checkpoint (C) (e.g., the current count from the free running counter 372) from the display device 340. In response to the request, the display device 340 transmits the checkpoint (C) to the CA device 330 over the CA module bus 360. In addition to sending the checkpoint (C) to the CA device 330, the display device 340 saves the checkpoint (C) in the memory 376.

The CA device 330 contains a processor 314. The processor 314 utilizes a ticket and the checkpoint (C), received from the display device 340, to create a relative time dependent ticket (TDT) as discussed in more detail below. In one embodiment, the processor 314 may simply be a fixed hardware device that is configured for performing functions, such as mathematical functions, including a concatenation function, a one-way function, such as a hashing function, etc. In alternate embodiments, the processor 314 may be a microprocessor or a reconfigurable hardware device. What is intended by the term "relative time dependent ticket (TDT)" is that due to the randomization of the counter 372 as discussed above, the checkpoint (C) is not directly related to an absolute time amongst all sink devices. The checkpoint (C) is only related to a relative time of a given sink device such as the display device 340.

In one embodiment, the copy protected content is received via an input 305 as an audio/video (A/V) signal. Preferably, in this embodiment, the A/V signal contains a watermark (W) and a ticket (T). The watermark (W) and the ticket (T) are related as discussed with regard to the prior art (e.g., $W = H(H(T))$). Preferably, the watermark (W) is embedded into the copy protected content. In this way, removal of the watermark (W) from the copy protected content will result in the copy protected content becoming largely degraded. The ticket accompanies the content and is not embedded in it.

In an alternate embodiment, the copy protected content is read from a physical medium, such as a digital video disc (DVD). In this embodiment, the DVD may contain a physical mark (P) as described above. Further, content contained on the DVD (e.g., A/V content) has a watermark (W) embedded therein (e.g., watermarked content) such that removal of the watermark (W) from the A/V content results in the A/V content becoming largely degraded. In this embodiment, the physical mark (P), the ticket (T), and the watermark (W) are related as follows:

$$T = H(H(P)) \qquad (1)$$

$$W = H(H(T)) \quad (2)$$

In any event, at the CA device 330, the checkpoint (C) is combined with the ticket (T), utilizing for instance concatenation and hashing functions. Thereby, a time dependent ticket (TDT) is created as follows:

$$TDT = H(T.C). \quad (3)$$

The watermarked content, containing a watermark (W) embedded therein, the time dependent ticket (TDT), and the ticket (T), are then transmitted via the CA module bus 360 to the display device 340.

At the receiver 340, the copy protection status determination circuit 370 extracts the watermark (W) from the watermarked content. The copy protection status determination circuit 370 compares the watermark (W) and the ticket (T) in the usual way, as is known in the art (e.g., $W = H(H(T))$?).

In the event that the comparison does not pass (e.g., $W \neq H(H(T))$), then the content is discarded and any selected operation at the display device 340 (e.g., play, record, etc.) regarding the content is disabled. However, if the comparison does pass (e.g., $W = H(H(T))$), then the copy protection determination circuit 370 retrieves the stored checkpoint (C) from the memory 376 and combines the ticket (T) with the stored checkpoint (C), utilizing the same operation that was utilized at the source device 330 for creating the time dependent ticket (TDT). To this end, the receiver 340 comprises a processor 314' that is comparable to the processor 314 in the source device 330. For instance, concatenation and hashing functions may be utilized at the display device 340 for combining the ticket (T) with the stored checkpoint (C). A result of the combination is then compared to the time dependent ticket (TDT):

$$TDT = H(T.C)? \quad (4)$$

In the event that the result does not equal the time dependent ticket (TDT), then the content is discarded and any selected operation at the display device (e.g., play, record, etc.) regarding the content is disabled. This may happen, for instance, in a case wherein an improper display device (e.g., a display device other than the display device that requested the content) has received the content. If the result does equal the time dependent ticket (TDT), then access to the content is enabled in accordance with the access granted by the ticket.

In a preferred embodiment, a further step is performed prior to the display device 340 having access to the copy protected content. Specifically, the checkpoint (C) stored in the memory 376 is compared to a current count of the (running) counter 372. In the event that the stored checkpoint (C) is within an allowable window of the current count from the counter 372 (e.g., within 24 hours of the count for some applications), then the display device 340 is provided with access to the copy protected content. What is an allowable window between the stored checkpoint (C) and the current count will depend on the nature of the desired content protection. For example, in one embodiment or for one particular type of content, the allowed window (the difference between the stored checkpoint (C) and the current count) may be short to ensure that the content is being transmitted and received in real time. In another embodiment or for another type of content, the allowed window may be longer (e.g., months or years) to allow for storage of the content for later playback.

If the checkpoint (C) has expired (e.g., not within the allowed window), then the checkpoint (C) is erased and the display device 340 is not provided with access to the copy protected content. As is readily ascertained by a person of ordinary skill in the art, the comparison of the checkpoint (C) to the current count may be performed any time prior to the display device having access to the copy protected content. In a preferred embodiment, the checkpoint (C) is compared to the current count prior to the comparison of the watermark (W) to the ticket (T).

It should be clear that a trusted source should be utilized to create the recorded content or the real time transmitted content (e.g., received over the input 305). A CA device, such as the CA device 330, which is inherently designed to be tamper resistant is an example of a trusted real time source. In this case, it may be assumed that the CA device 330 decrypts the watermarked content so that prior to the watermarked contents arrival at the CA device 330, the watermarked content cannot be recorded.

In a case wherein the ticket (T) does not properly compare to the watermark (W), or some other portion of the copy protection status determination process fails, the copy protected content is discarded. In addition, when the copy protection status determination process fails, no operation regarding the copy protected content is enabled at the display device 340.

In accordance with the present invention, a checkpoint (C) from a counter of a given display device is in effect unique. Accordingly, the copy protected content transmitted by the CA device 330 may not be distributed to a display device other than the display device that sent the checkpoint (C). In addition, by comparing the checkpoint (C) to the count of the

counter 372, the copy protected content may be restricted to being played within a time, as determined by the window of time as discussed above.

In yet another embodiment, a private/public key system, as is known by a person of ordinary skill in the art, is utilized to further secure the copy protected content in accordance with the present invention. In accordance with this embodiment, the display device 340 has a public key that is trusted e.g., secure for example by being installed in part of the display device hardware, such as stored in the memory 376. The public key corresponds to a private key of the manufacturer of the display device 340 and is stored, for instance, in a memory 322 at the CA device 330. The private key is utilized to sign certificates of each CA device manufacturer, as is known in the art.

In operation, when the CA device 330 is connected to the display device 340 via the CA module bus 360, a certificate containing the CA device 330 public key is sent to the display device 340. Once the certificate containing the public key of the CA device 330 is verified by the display device 340, as is known in the art, the public key of this CA device 330 is stored at the display device 340. Thereafter, the CA device 330 may digitally sign the time dependent ticket (TDT). For instance, the time dependent ticket (TDT) may be hashed and the result may be encrypted by the private key of the CA device 330 to form a signature. The signature is sent from the CA device 330 to the display device 340 together with the watermarked content, the ticket, and the time dependent ticket (TDT). At the display device 340, the signature is verified utilizing the public key of the CA device 330 and thereafter, the time dependent ticket (TDT) and checkpoint (C) are utilized as described above.

In yet another embodiment, the time dependent ticket (TDT) may be encrypted utilizing the private key of the CA device 330. The encrypted time dependent ticket (TDT) is then transmitted from the CA device 330 to the display device 340 along with the watermarked content and the ticket (T). Thereafter, prior to the display device 340 verifying the checkpoint (C), the display device 340 decrypts the time dependent ticket (TDT) utilizing the public key of the CA device 330. Thereafter, the time dependent ticket (TDT) may be utilized as discussed above.

An illustrative protocol for use of a checkpoint and a private/public key system in accordance with an embodiment of the present invention is described below. In accordance with the present invention, after a CA device is connected to a display device, the CA device sends a certificate containing the CA device public key to the display device. The display device verifies the certificate utilizing the embedded public key of the manufacturer and stores the verified public key of the CA device. In response to a request for copy protected content

from the display device, the CA device requests a checkpoint (C) from the display device. The display device sends the checkpoint (C) to the CA device and also stores a copy of the checkpoint (C) locally (e.g., at the display device). The CA device combines the checkpoint (C) with the ticket (T) utilizing concatenation and hashing functions to produce a time dependent ticket (TDT). The CA device encrypts the time dependent ticket (TDT) utilizing the CA device private key. The encrypted time dependent ticket (TDT) is then sent to the display device along with the watermarked content and the ticket (T). The display device compares the stored checkpoint (C) with the current state of a counter to determine if the checkpoint (C) is within an allowable window of time of the current state of the counter. If the stored checkpoint (C) is not within the allowable window of time of the current state of the counter, then access to the content is disabled. If the stored checkpoint (C) is within the allowable window, then the display device utilizes the public key of the CA device to decrypt the time dependent ticket (TDT). The display device combines the ticket (T) with the stored checkpoint (C) utilizing concatenation and hashing functions and compares a result to the time dependent ticket (TDT). If the result is not equal to the time dependent ticket (TDT), then access to the content is disabled. If the result is equal to the time dependent ticket (TDT), the ticket and watermark are compared in the usual way. If step 480 fails (e.g., $W \neq H(H(T))$), then in step 485, access to the content is disabled. If the ticket and the watermark do no correspond, (e.g., $W = H(H(T))$), access to the content is enabled (e.g., the content may be displayed).

The following embodiments of the invention overcome the disadvantages of the prior art. A display device is provided that is the final arbiter in deciding whether to display the protected content. In this way, the display device is the gatekeeper that disallows recordings that are made and played back on non-compliant players/recorders. A further embodiment provides a method of transmitting copy protected copy-never content that will prevent a pirate from making copies that will display on a compliant display device. A ticket is created that is unique to a particular display device so that copy protected content will only play on the particular display device. A still further embodiment creates a ticket that is inspected by the display device to decide whether the content is being transmitted in real time. A time dependent ticket is created that is checked by a display device to determine if content has expired or aged beyond an allowable window of time from a checkpoint. Another embodiment of the invention uses a relative time reference configured such that each display device has a different relative time reference.

It should be noted that the above-mentioned embodiments illustrate rather than limit the invention, and that those skilled in the art will be able to design many alternative

embodiments without departing from the scope of the appended claims. In the claims, any reference signs placed between parentheses shall not be construed as limiting the claim. The word "comprising" does not exclude the presence of other elements or steps than those listed in a claim. Another embodiment of the invention can be implemented by means of hardware comprising several distinct elements, and by means of a suitably programmed computer. In a device claim enumerating several means, several of these means can be embodied by one and the same item of hardware.

CLAIMS:

1. A method of protecting content transmitted as a stream of data, the method comprising the steps of:

determining a checkpoint at a receiving device (240);

calculating, at a source device (230), a time dependent ticket utilizing the checkpoint, wherein a watermark, a ticket, and the checkpoint together indicate a copy protection status of the content;

transmitting said stream of data, said watermark, said ticket, and said time dependent ticket to said receiving device (240); and

comparing said time dependent ticket to a stored checkpoint at said receiving device (240).

2. The method of claim 1, wherein said step of calculating said time dependent identifier comprises the steps of:

combining said checkpoint with said ticket, and

calculating a one-way operation on said combined checkpoint and ticket.

3. The method of claim 2, further comprising the step of selecting said one-way function to be a hashing function.

4. The method of claim 1, further comprising the step of comparing, at said receiving device (240), said ticket and said watermark to determine the copy protection status of the content if said time dependent ticket compares to said stored checkpoint.

5. The method of claim 1, wherein said checkpoint is a checkpoint from a receiver counter (272).

6. The method of claim 5, wherein said receiver counter (272) is randomized.

7. The method of claim 5, wherein the step of comparing said time dependent ticket further comprises the step of comparing said stored checkpoint to a current count from said receiver counter (272).

8. The method of claim 1, wherein said step of calculating said time dependent ticket further comprises the step of signing said time dependent ticket with a private key of said source device (230), and wherein said step of comparing said time dependent ticket further comprises the step of verifying the signature using a public key of said source device (230).

9. A copy protection system for protecting content wherein a ticket and a watermark indicates a copy protection status of said content, the system comprising:

a source device (330) configured to calculate a time dependent ticket using a checkpoint and a one-way function, and to provide a data stream containing said content, said ticket, a watermark, and said time dependent ticket; and

a display device (340) configured to produce said checkpoint, configured to receive said data stream, and configured to compare said time dependent ticket to said checkpoint using said ticket and said one-way function.

10. The system of claim 9, wherein said display device (340) is further configured to compare said ticket to said watermark and to display said content if said time dependent ticket compares to said checkpoint.

11. The system of claim 9, wherein said display device (340) comprises a counter (372) and wherein said checkpoint is a checkpoint from said counter (372).

12. The system of claim 11, wherein said display device (340) is further configured to randomize said counter (372).

13. The system of claim 11, wherein said display device (340) is further configured to compare said checkpoint to a current count from said counter (372) prior to displaying said content.

14. A source device (330) for protecting content wherein a ticket and a watermark indicate a copy protection status of the content, said source device (330) comprising:

a reader device configured to read watermarked content from a physical medium and configured to read a physical mark from said physical medium; and

a processor (314) configured to receive a checkpoint, configured to calculate said ticket using said physical mark and a one-way function, configured to calculate a time dependent ticket using said ticket, said checkpoint, and said one-way function, and configured to provide to a receiver (340) a data stream containing said watermarked content, said ticket, and said time dependent ticket.

15. A display device (340) for receiving data containing watermarked content and a ticket, wherein said ticket and watermark together indicate a copy protection status of the content, said display device comprising:

a counter (372) configured to provide a checkpoint and a current time reference; and

a processor (314'), wherein if said checkpoint is contained within a time window determined by said current time reference, said (314') processor is configured to:

receive a time dependent ticket and said data,

combine said ticket with said checkpoint to produce a first result,

perform a one-way function on said first result to produce a second result, and compare said second result to said time dependent ticket, wherein said display device (340) is further configured to display said data if said second results compares to said time dependent ticket.

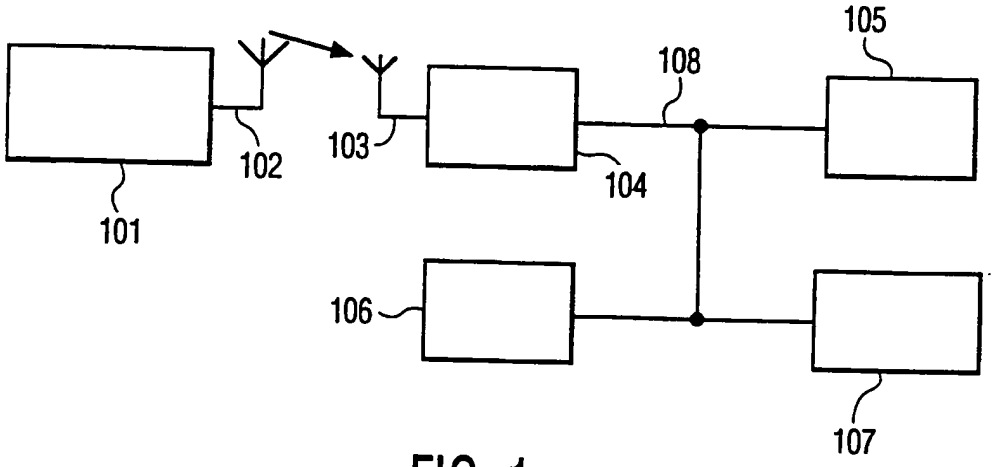

105
108
103
102
104
101
106
107

FIG. 1

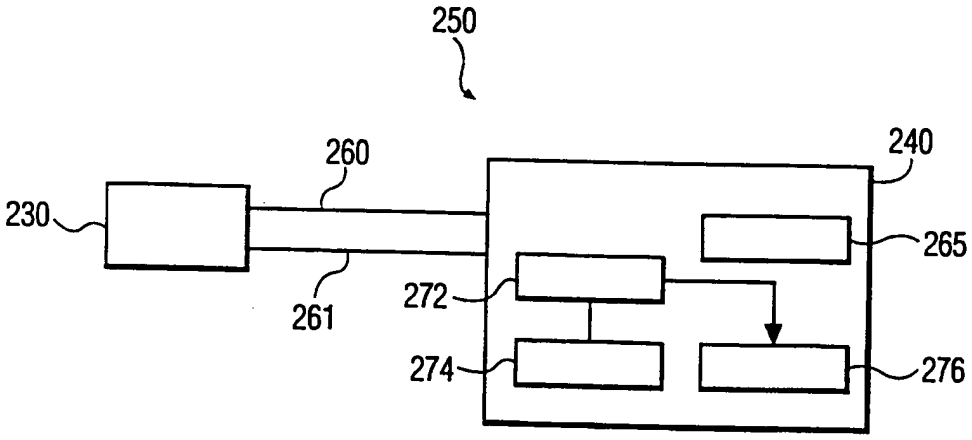

250
260
240
230
265
272
261
274
276

FIG. 2

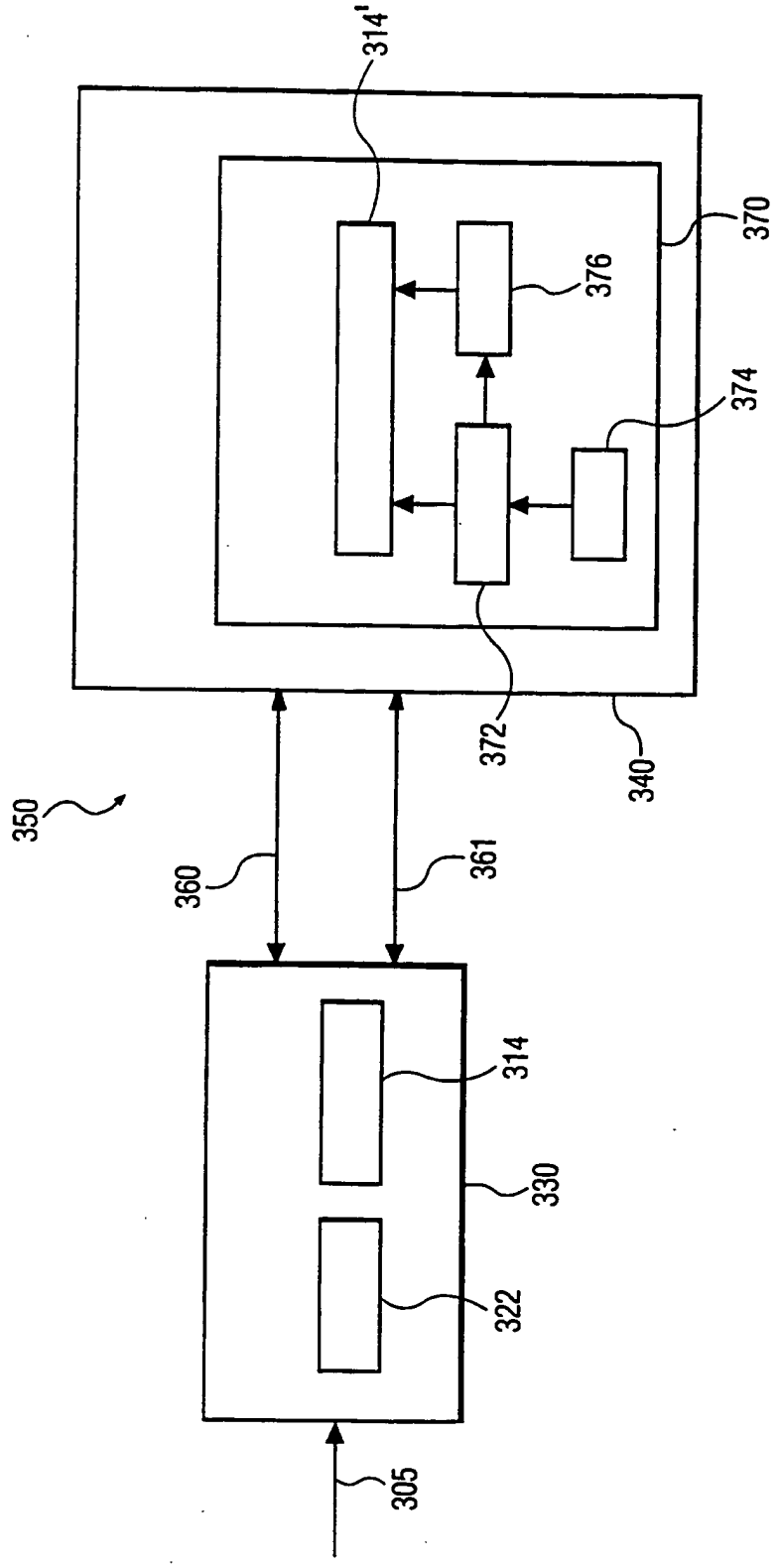


FIG. 3

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 7 : H04N 7/30 — A2

(11) International Publication Number: **WO 00/05898**

(43) International Publication Date: 3 February 2000 (03.02.00)

(21) International Application Number: PCT/US99/16638

(22) International Filing Date: 21 July 1999 (21.07.99)

(30) Priority Data:

60/093,860	23 July 1998 (23.07.98)	US
09/169,829	11 October 1998 (11.10.98)	US

(63) Related by Continuation (CON) or Continuation-in-Part (CIP) to Earlier Applications

US	60/093,860 (CIP)
Filed on	23 July 1998 (23.07.98)
US	09/169,829 (CIP)
Filed on	11 October 1998 (11.10.98)

(71) Applicant *(for all designated States except US)*: OPTIVISION, INC. [US/US]; 3450 Hillview Avenue, Palo Alto, CA 94304 (US).

(72) Inventor; and

(75) Inventor/Applicant *(for US only)*: LI, Weiping [US/US]; 159 California Avenue, J103, Palo Alto, CA 94306 (US).

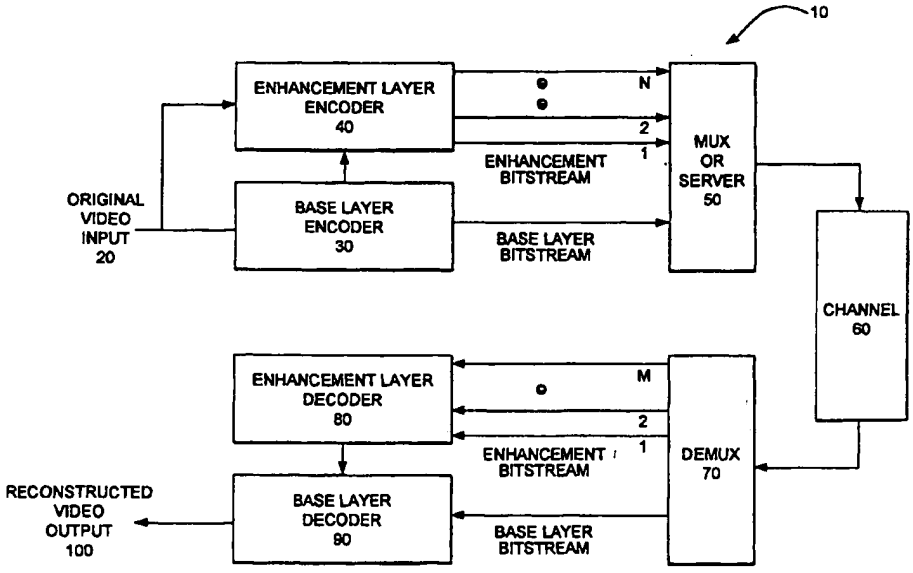
(74) Agent: DAVIS, Paul; Wilson Sonsini Goodrich & Rosati, 650 Page Mill Road, Palo Alto, CA 94304-1050 (US).

(81) Designated States: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, US, UZ, VN, YU, ZA, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published

Without international search report and to be republished upon receipt of that report.

(54) Title: SCALABLE VIDEO CODING AND DECODING



(57) Abstract

A video encoding method and apparatus for adapting a video input to a bandwidth of a transmission channel of a network that includes determining the number N enhancement layer bitstreams capable of being adapted to the bandwidth of the transmission channel of a network. A base layer bitstream is encoded from the video input wherein a plurality of enhancement layer bitstreams are encoded from the video input. The enhancement layer bitstreams are based on the base layer bitstream, wherein the plurality of enhancement layer bitstreams complements the base layer bitstream and the base layer bitstream and N enhancement layer bitstreams are transmitted to the network.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

SCALABLE VIDEO CODING AND DECODING

BACKGROUND OF THE INVENTION

Field of the Invention

The present invention relates to a method and apparatus for the scaling of data signals the bandwidth of the transmission channel; and more particularly to a scalable video method and apparatus for coding video such that the received video is adapted to the bandwidth of the transmission channel.

Description of Related Art

Signal compression in the video arena has long been employed to increase the bandwidth of either the generating, transmitting, or receiving device. MPEG - an acronym for Moving Picture Experts Group - refers to the family of digital video compression standards and file formats developed by the group. For instance, the MPEG-1 video sequence is an ordered stream of bits, with special bit patterns marking the beginning and ending of a logical section.

MPEG achieves high compression rate by storing only the changes from one frame to another, instead of each entire frame. The video information is then encoded using a technique called DCT (Discrete Cosine Transform) which is a technique for representing a waveform data as a weighted sum of cosines. MPEG use a type of lossy compression wherein some data is removed. But the diminishment of data is generally imperceptible to the human eye. It should be noted that the DCT itself does not lose data; rather, data compression technologies that rely on DCT approximate some of the coefficients to reduce the amount of data.

The basic idea behind MPEG video compression is to remove spatial redundancy within a video frame and temporal redundancy between video frames. The DCT-based (Discrete Cosine Transform) compression is used to reduce spatial redundancy and motion compensation is used to exploit temporal redundancy. The images in a video stream usually do not change much within small time intervals.

Thus, the idea of motion-compensation is to encode a video frame based on other video frames temporally close to it.

A video stream is a sequence of video frames, each frame being a still image. A video player displays one frame after another, usually at a rate close to 30 frames per second. Macroblocks are formed, each macroblock consists of four 8 x 8 luminance blocks and two 8 x 8 chrominance blocks. Macroblocks are the units for motion-compensated compression, wherein blocks are basic unit used for DCT compression. Frames can be encoded in three types: intra-frames (I-frames), forward predicted frames (P-frames), and bi-directional predicted frames (B-frames).

An I-frame is encoded as a single image, with no reference to any past or future frames. Each 8 x 8 block is encoded independently, except that the coefficient in the upper left corner of the block, called the DC coefficient, is encoded relative to the DC coefficient of the previous block. The block is first transformed from the spatial domain into a frequency domain using the DCT (Discrete Cosine Transform), which separates the signal into independent frequency bands. Most frequency information is in the upper left corner of the resulting 8 x 8 block. After the DCT coefficients are produced the data is quantized, i.e. divided or separated. Quantization can be thought of as ignoring lower-order bits and is the only lossy part of the whole compression process other than sub-sampling.

The resulting data is then run-length encoded in a zig-zag ordering to optimize compression. The zig-zag ordering produces longer runs of 0's by taking advantage of the fact that there should be little high-frequency information (more 0's as one zig-zags from the upper left corner towards the lower right corner of the 8 x 8 block).

A P-frame is encoded relative to the past reference frame. A reference frame is a P- or I-frame. The past reference frame is the closest preceding reference frame. A P-macroblock is encoded as a 16 x 16 area of the past reference frame, plus an error term.

To specify the 16 x 16 area of the reference frame, a motion vector is included. A motion vector (0, 0) means that the 16 x 16 area is in the same position as the macroblock we are encoding. Other motion vectors are generated are relative to that position. Motion vectors may include half-pixel values, in which case pixels are averaged. The error term is encoded using the DCT, quantization, and run-length

encoding. A macroblock may also be skipped which is equivalent to a (0, 0) vector and an all-zero error term.

A B-frame is encoded relative to the past reference frame, the future reference frame, or both frames.

A pictorial view of the above processes and techniques in application are depicted in prior art Fig. 15, which illustrates the decoding process for a SNR scalability. Scalable video coding means coding video in such a way that the quality of a received video is adapted to the bandwidth of the transmission channel. Such a coding technique is very desirable for transmitting video over a network with a time-varying bandwidth.

SNR scalability defines a mechanism to refine the DCT coefficients encoded in another (lower) layer of a scalable hierarchy. As illustrated in prior art Fig. 15, data from two bitstreams is combined after the inverse quantization processes by adding the DCT coefficients, Until the dat is combined, the decoding processes of the two layers are independent of each other.

The lower layer (base layer) is derived from the first bitstream and can itself be either non-scalable, or require the spatial or temporal scalability decoding process, and hence the decoding of additional bitstream, to be applied. The enhancement layer, derived from the second bitstream, contains mainly coded DCT coefficients and a small overhead.

In the current MPEG-2 video coding standard, there is an SNR scalability extension that allows two levels of scalability. MPEG achieves high compression rate by storing only the changes from one frame to another, instead of each entire frame. There are at least two disadvantages of employing the MPEG-2 standard for encoding video data. One disadvantage is that the scalability granularity is not fine enough, because the MPEG-2 process is an all or none method. Either the receiving device can receive all of the data from the base layer and the enhancement layer or only the data from the base layer bitstream. Therefore, the granularity is not scalable. In a network environment, more than two levels of scalability are usually needed.

Another disadvantage is that the enhancement layer coding in MPEG-2 is not efficient. Too many bits are needed in the enhancement layer in order to have a noticeable increase in video quality.

The present invention overcomes these disadvantages and others by providing, among other advantages, an efficient scalable video coding method with increased granularity.

SUMMARY OF THE INVENTION

The present invention can be characterized as a scalable video coding means and a system for encoding video data, such that quality of the final image is gradually improved as more bits are received. The improved quality and scalability are achieved by a method wherein an enhancement layer is subdivided into layers or levels of bitstream layers. Each bitstream layer is capable of carrying information complementary to the base layer information, in that as each of the enhancement layer bitstreams are added to the corresponding base layer bitstreams the quality of the resulting images are improved.

The number N of enhancement layers is determined or limited by the network that provides the transmission channel to the destination point. While the base layer bitstream is always transmitted to the destination point, the same is not necessarily true for the enhancement layers. Each layer is given a priority coding and transmission is effectuated according to the priority coding. In the event that all of the enhancement layers cannot be transmitted the lower priority coded layers will be omitted. The omission of one or more enhancement layers may be due to a multitude of reasons.

For instance, the server which provides the transmission channel to the destination point may be experiencing large demand on its resources from other users, in order to try and accommodate all of its users the server will prioritize the data and only transmit the higher priority coded packets of information. The transmission channel may be the limiting factor because of the bandwidth of the channel, i.e. Internet access port, Ethernet protocol, LAN, WAN, twisted pair cable, co-axial cable, etc. or the destination device itself, i.e. modem, absence of an enhanced video card, etc. may not be able to receive the additional bandwidth made available to it. In these instances only M number (M is an integer number = 0, 1, 2, . . .) of enhancement layers may be received, wherein N number (N is an integer number = 0, 1, 2, . . .) of enhancement layers were generated at the encoding stage, $M \leq N$.

To achieve these and other advantages and in accordance with the purpose of the present invention, as embodied and broadly described, the scalable video method and apparatus according to one aspect of the invention includes a video encoding method for adapting a video input to a bandwidth of a transmission channel of a network, the method includes determining the number N of enhancement layer bitstreams capable of being adapted to the bandwidth of the transmission channel of the network. Encoding a base layer bitstream from the video input is then performed and encoding N number of enhancement layer bitstreams from the video input based on the base layer bitstream, wherein the plurality of enhancement layer bitstreams complements the base layer bitstream. The base layer bitstream and the N enhancement layer bitstreams are then provided to the network.

According to another aspect of the present invention, a video decoding method for adapting a video input to a bandwidth of a transmission channel of a network includes, determining number M of enhancement layer bitstreams of said video input capable of being received from said transmission channel of said network. Decoding a base layer bitstream from received video input and decoding M number of enhancement layer bitstreams from the received video input based on the base layer bitstream, wherein the M received enhancement layer bitstreams complements the base layer bitstream. Then reconstructing the base layer bitstream and N enhancement layer bitstreams.

According to still another aspect of the present invention, a video decoding method for adapting a video input to a bandwidth of a receiving apparatus, the method includes demultiplexing a base layer bitstream and at least one of a plurality of enhancement layer bitstreams received from a network, decoding the base layer bitstream, decoding at least one of the plurality of enhancement layer bitstreams based on generated base layer bitstream, wherein the at least one of the plurality of enhancement layer bitstreams enhances the base layer bitstream. Then reconstructing a video output.

According to a further aspect of the present invention, a video encoding method for encoding enhancement layers based on a base layer bitstream encoded from a video input, the video encoding method includes, taking a difference between an

original DCT coefficient and a reference point and dividing the difference between the original DCT coefficient and the reference point into N bit-planes.

According to a still further aspect of the present invention, a method of coding motion vectors of a plurality of macroblocks, includes determining an average motion vector from N motion vectors for N macroblocks, utilizing the determined average motion vector as the motion vector for the N macroblocks, and encoding 1/N motion vectors in a base layer bitstream.

Additional features and advantages of the invention will be set forth in the description which follows, and in part will be apparent from the description, or may be learned by practice of the invention. The aspects and other advantages of the invention will be realized and attained by the structure particularly pointed out in the written description and claims hereof as well as the appended drawings.

It is to be understood that both the foregoing general description and the following detailed description are exemplary and explanatory and are intended to provide further explanation of the invention as claimed.

BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings, which are included to provide a further understanding of the invention and are incorporated in and constitute a part of this specification, illustrate embodiments of the invention and together with the description serve to explain the principles of the invention. In the drawings:

Fig. 1 illustrates a flow diagram of the scalable video encoding method of the present invention;

Fig. 2A illustrates conventional probability distribution of DCT coefficient values;

Fig. 2B illustrates conventional probability distribution of DCT coefficient residues;

Fig. 3A illustrates the probability distribution of DCT coefficient values of the present invention;

Fig. 3B illustrates the probability distribution of DCT coefficient residues of the present invention;

Figs. 3C and 3D illustrates a method for taking a difference of a DCT coefficient of the present invention;

Fig. 5 illustrates a flow diagram for finding the maximum number of bit-planes in the DCT differences of a frame of the present invention;

Fig. 6 illustrates a flow diagram for generating (RUN, EOP) Symbols of the present invention;

Fig. 7 Illustrates a flow diagram for encoding enhancement layers of the present invention;

Fig. 8 illustrates a flow diagram for encoding (RUN, EOP) symbols and sign_enh values of one DCT block of one bit-plane;

Fig. 9 illustrates a flow diagram for encoding a sign_enh value of the present invention;

Fig. 10 illustrates a flow diagram for adding enhancement difference to a DCT coefficient of the present invention;

Fig. 11 illustrates a flow diagram for converting enhancement difference to a DCT coefficient of the present invention;

Fig. 12 illustrates a flow diagram for decoding enhancement layers of the present invention;

Fig. 13 illustrates a flow diagram for decoding (RUN, EOP) symbols and sign_enh values of one DCT block of one bit-plane;

Fig. 14 illustrates a flow diagram for decoding a sign_enh value; and

Fig. 15 illustrates a prior a conventional SNR scalability flow diagram.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Reference will now be made in detail to the preferred embodiments of the present invention, examples of which are illustrated in the accompanying drawings.

Fig. 1 illustrates the scalable video diagram 10 of an embodiment of the present invention. The original video input 20 is encoded by the base layer encoder 30 in accordance with the method of represent by flow diagram 400 of Fig. 4. A DCT coefficient OC and its corresponding base layer quantized DCT coefficient QC are generated and a difference determined pursuant to steps 420 and 430 of Fig. 4. The

difference information from the base layer encoder 30 is passed to the enhancement layer encoder 40 that encodes the enhancement information.

The encoding of the enhancement layer encoder is performed pursuant to methods 500 - 900 as depicted in Figs. 5 - 10, respectively and will be briefly described. The bitstream from the base layer encoder 30 and the N bitstreams from the enhancement layer encoder 40 are capable of being sent to the transmission channel 60 by at least two methods.

In the first method all bitstreams are multiplexed together by multiplexor 50 with different priority identifiers, e.g., the base layer bitstream is guaranteed, enhancement bitstream layer 1 provided by enhancement layer encoder 40 is given a higher priority than enhancement bitstream layer 2. The prioritization is continued until all N (wherein N is an integer from 0, 1, 2, . . .) of the bitstreams layers are prioritized. Logic in the encoding layers 30 or 40 in negotiation with the network and intermediated devices determine the number N of bitstream layers to be generated.

The number of bitstream layers generated is a function of the total possible bandwidth of the transmission channel 60, i.e. Ethernet, LAN, or WAN connections (this list is not intended to exhaustive but only representation of potential limiting devices and/or equipment), and the network and other intermediate devices. The number of bitstream layers M (wherein M is an integer and $M \leq N$) reaching the destination point 100 can be further limited by not just the physical constraints of the intermediate devices but the congestion on the network, thereby necessitating the dropping of bitstream layers according to their priority.

In a second method the server 50 knows the transmission channel 60 condition, i.e. congestion and other physical constraints, and selectively sends the bitstreams to the channel according to the priority identifiers. In either case, the destination point 100 receives the bitstream for the base layer and M bitstreams for the enhancement layer, where $M \leq N$.

The bitstreams M are sent to the base layer 90 and enhancement layer 80 decoders after being demultiplexed by demultiplexor 70. The decoded enhancement information from the enhancement layer decoder is passed to the base layer decoder to composite the reconstructed video output 100. The decoding of the multiplexed

bitstreams are accomplished pursuant to the methods and algorithms depicted in flow diagrams 1100 - 1400 of Figs. 11 - 14, respectively.

The base layer encoder and decoder are capable of performing logic pursuant to the MPEG-1, MPEG-2, or MPEG-4 (Version-1) standards that are hereby incorporated by reference into this disclosure.

Taking Residue with Probability Distribution Preserved

A detailed description of the probability distribution residue will now be made with reference to Figs 2A - 3B

In the current MPEG-2 signal-to-noise ratio (SNR) scalability extension, a residue or difference is taken between the original DCT coefficient and the quantized DCT coefficient. Fig. 2A illustrates the distribution of a residual signal as a DCT coefficient. In taking the residue small values have higher probabilities and large values have smaller probabilities. The intervals along the horizontal axis represent quantization bins. The dot in the center of each interval represents the quantized DCT coefficient. Taking the residue between the original and the quantized DCT coefficient is equivalent to moving the origin to the quantization point.

Therefore, the probability distribution of the residue becomes that as shown in Figure 2B. The residue from the positive side of Fig. 2A has a higher probability of being negative than positive and the residue taken from the negative side of the Fig. 2A has a higher probability of being positive than negative. The result is that the probability distribution of the residue becomes almost uniform. Thus making coding the residue more difficult.

A vastly superior method is to generate a difference between the original and the lower boundary points of the quantized interval as shown in Fig. 3A and Fig. 3B. In this method, the residue is taken from the positive side of Fig. 2A remains positive and the residue from the negative side of Fig. 2A remains negative. Taking the residue is equivalent to moving the origin to the reference point as illustrated in Fig. 3A. Thus, the probability of the residue becomes as shown in Fig. 3B. This method preserves the shape of the original non-uniform distribution. Although the dynamic range of the residue taken in such a manner seems to be twice of that depicted in Fig. 2B, their is no longer a need to code the sign, i.e. - or +, of the residue. The sign of the residue is

encoded in the base layer bitstream corresponding the enhancement layer, therefore this redundancy is eliminated and bits representing the sign are thus saved. Therefore, there is only a need to code the magnitude that still has a nonuniform distribution.

Bit plane coding of residual DCT coefficients

After taking residues of all the DCT coefficients in an 8 x 8 block, bit plane coding is used to code the residue. In bit-plane coding method the bit-plane coding method considers each residual DCT coefficient as a binary number of several bits instead of as a decimal integer of a certain value as in the run-level coding method. The bit-plane coding method in the present invention only replaces runlevel coding part. Therefore, all the other syntax elements remain the same.

An example of and description of the bit-plane coding method will now be made, wherein 64 residual DCT coefficients for an Inter-block and 63 residual DCT coefficients for an Intra-block (excluding the Intra-DC component that is coded using a separate method) are utilized for the example. The 64 (or 63) residual DCT coefficients are ordered into a one-dimensional array and at least one of the residual coefficients is non-zero. The bit-plane coding method then performs the following steps.

The maximum value of all the residual DCT coefficients in a frame is determined and the minimum number of bits, N, needed to represent the maximum value in the binary format is also determined. N is the number of bitplanes layers for this frame and is coded in the frame header.

Within each 8 x 8 block is represent every one of the 64 (or 63) residual DCT coefficients with N bits in the binary format and there is formed N bit-planes or layers or levels. A bit-plane is defined as an array of 64 (or 63) bits, taken one from each residual DCT coefficient at the same significant position.

The most significant bit-plane is determined with at least one non-zero bit and then the number of all-zero bit-planes between the most significant bit-plane determined and the Nth one is coded. Then starting from the most significant bit plane (MSB plane), 2-D symbols are formed of two components: (a) number of consecutive O's before a I (RUN), (b) whether there are any I's left on this bit plane, i.e. End-Of-Plane (EOP). If a bit-plane after the MSB plane contains all O's, a special symbol

ALL-ZERO is formed to represent an all-zero bit-plane. Note that the MSB plane does not have the all-zero case because any all-zero bit-planes before the MSB plane have been coded in the previous steps.

Four 2-D VLC tables are used, wherein the table VT-C-Table-0 corresponds to the MSB plane; table VLC-Table- I corresponds to the second MSB plane; table VLC-Table-2 corresponds to the third MSB plane; and table VLC-Table-3 corresponds to the fourth MSB and all the lower bit planes. For the ESCAPE cases, RUN is coded with 6 bits, EOP is coded with 1 bit. Escape coding is a method to code very small probability events which are not in the coding tables individually.

An example of the above process will now follow. For illustration purposes, we will assume that the residual values after the zigzag ordering are given as follows and N = 6: The following representation is thereby produced.

10, 0, 6, 0, 0, 3, 0, 2, 2, 0, 0, 2, 0, 0, 1, 0, ... 0, 0

The maximum value in this block is found to be 10 and the minimum number of bits to represent 10 in the binary format (1010) is 4. Therefore, two all-zero bit-planes before the MSB plane are coded with a code for the value 2 and the remaining 4 bit-planes are coded using the (RUN, EOP) codes. Writing every value in the binary format using 4 bits, the 4 bit-planes are formed as follows:

1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,0, 0 (MSB-plane)
0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,0, 0 (Second MSB-plane)
1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0,0, 0 (Third MSB-plane)
0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,0, 0 (Fourth MSB-plane or LSB-plane)

Converting the bits of each bit-plane into (RUN, EOP) symbols results in the following:

(0, 1)	(MSB-plane)
(2, 1)	(Second MSB-plane)
(0, 0), (1,O), (2,O), (1,O), (0, 0), (2, 1)	(Third MSB-plane)

(5, 0), (8, 1) (Fourth MSB-plane or LSB-plane)

Therefore, there are 10 symbols to be coded using the (RUN, EOP) VLC tables. Based on their locations in the bit-planes, different VLC tables are used for the coding. The enhancement bitstream using all four bitplanes looks as follows:

code leading-all-zero(2)

code msb(O, 1)

code msb-1(2,1)

code-msb-2(0,0), code_msb-2(1,0), code-msb-2(2,O), code-msb-2(1,0), code-msb-2(0,0), code-msb-2(2, 1) code_msb-3(5,0), code_msb-3(8, 1).

In an alternative embodiment, several enhancement bitstreams may be formed from the four bit-planes, in this example from the respective sets comprising one or more of the four bit-planes.

Motion Vector Sharing

In this alternative embodiment of the present invention motion vector sharing is capable of being utilized when the base layer bitstream exceeds a predetermined size or more levels of scalability are needed for the enhancement layer. By lowering the number of bits required for coding the motion vectors in the base layer the bandwidth requirements of the base layer bitstream is reduced. In base layer coding, a macroblock (16 x 16 pixels for the luminance component and W pixels for each chron-luminance components) of the current frame is compared with the previous frame within a search range. The closest match in the previous frame is used as a prediction of the current macroblock. The relative displacement of the prediction to the current macroblock, in the horizontal and vertical directions, is called a motion vector.

The difference between the current macroblock and it's prediction is coded using the DCT coding. In order for the decoder to reconstruct the current macroblock, the motion vector has to be coded in the bitstream. Since there is a fixed number of bits for coding a frame, the more bits spent on coding the motion vectors results in fewer bits for coding the motion compensated differences. Therefore, it is desirable to lower the number of bits for coding the motion vectors and leave more bits for coding the differences between the current macroblock and its prediction.

For each set of 2 x 2 motion vectors, the average motion vector can be determined and used for the four macroblocks. In order to not change the syntax of the base layer coding, four macroblocks are forced to have the identical motion vectors. Since only one out four motion vectors is coded in the bitstream, the amount of bits spent on motion vector coding is reduced, therefore, there are more bits available for coding the differences. The cost for pursuing such a method is that the four macroblocks, which share the same motion vector may, not get the best matched prediction individually and the motion compensated difference may have a larger dynamic range, thus necessitating more bits to code the motion vector.

For a given fixed bitrate, the savings from coding one out of four motion vectors may not compensate the increased number of bits required to code the difference with a larger dynamic range. However, for a time varying bitrate, a wider dynamic range for the enhancement layer provides more flexibility to achieve the best possible usage of the available bandwidth.

Coding Sign Bits

In an alternative embodiment of the present invention, if the base layer quantized DCT coefficient is non-zero, the corresponding enhancement layer difference will have the same sign as the base layer quantized DCT. Therefore, there is no need to code the sign bit in the enhancement layer.

Conversely, if the base layer quantized DCT coefficient is zero and corresponding enhancement layer difference is non-zero, a sign bit is placed into enhancement layer bitstream immediately after the MSB of the difference.
An example of the above method will now follow.

Difference of a DCT block after ordering

- 10, 0, 6, 0, 0, 3, 0, 2, 2, 0, 0, 2, 0, 0, 1, 0, ...0, 0

Sign indications of the DCT block after ordering

- 3, 3, 3, 3, 2, 0, 3, 3, 1, 2, 2, 0, 3, 3, 1, 2, ... 2, 3

- 0: base layer quantized DCT coefficient = 0 and difference >0
- 1: base layer quantized DCT coefficient = 0 and difference <0
- 2: base layer quantized DCT coefficient = 0 and difference =0

- 3: base layer quantized DCT coefficient = 0.

In this example, the sign bits associated with values 10, 6, 2 don't need to be coded and the sign bits associated with 3, 2, 2, 1 are coded in the following way:

Code(All Zero)

code (All Zero)

code(0,1)

code(2,1)

code(0,0),code(1,0),code(2,0),0,code(1,0),code(0,0),1,code(2,1),0

code(5,0),code(8,1),1

For every DCT difference, there is a sign indication associated with it. There are four possible cases. In the above coding 0, 1, 2, and 3 are used to denote the four cases. If the sign indication is 2 or 3, the sign bit does not have to be coded because it is either associated with a zero difference or available from the corresponding base layer data. If the sign indication is 0 or 1 a sign bit code is required once per difference value, i.e. not every bit-plane of the difference value. Therefore, a sign bit is put immediately after the most significant bit of the difference.

Optimal Reconstruction of the DCT Coefficients

In an alternative embodiment of the present invention, even though N enhancement bitstream layers or planes may have been generated, only M, wherein M $\leq$ N enhancement layer bits are available for reconstruction of the DCT coefficients due to the channel capacity, and other constraints such as congestion among others, the decoder 80 of Fig. 1 may receive no enhancement difference or only a partial enhancement difference. In such a case, the optimal reconstruction of the DCT coefficients is capable of proceeding along the following method:

If decoded difference = 0, the reconstruction point is the same as that in base layer, otherwise, the reconstructed difference = decoded difference + ¼ *(1<<decoded_bit_plane) and the reconstruction point = reference point + reconstructed difference * Q_enh +Q_enh/2.

In the present embodiment, referring to Figs. 3C and 3D, the optimal reconstruction point is not the lower boundary of a quantization bin. The above method specifies how to obtain the optimal reconstruction point in cases where the

difference is quantized and received partially, i.e. not all of the enhancement layers generated are either transmitted or received as shown in Fig. 1. wherein $M \leq N$.

What is claimed is:

1. A video encoding method for adapting a video input to a bandwidth of a transmission channel of a network, the method comprising the steps of:

determining number N of enhancement layer bitstreams capable of being adapted to said bandwidth of said transmission channel of said network;

encoding a base layer bitstream from said video input;

encoding N number of enhancement layer bitstreams from said video input based on the base layer bitstream, wherein the N enhancement layer bitstreams complements the base layer bitstream; and

providing the base layer bitstream and N enhancement layer bitstreams to said network.

2. The video encoding method according to claim 1, wherein the determining step includes negotiating with intermediate devices on said network.

3. The video encoding method according to claim 2, wherein negotiating includes determining destination resources.

4. The video encoding method according to claim 1, wherein the step of encoding the base layer bitstreams is performed by a MPEG-1 encoding method.

5. The video encoding method according to claim 1, wherein the step of encoding the base layer bitstreams is performed by a MPEG-2 encoding method.

6. The video encoding method according to claim 1, wherein the step of encoding the base layer bitstreams is performed by a MPEG-4 encoding method.

7. The video encoding method according to claim 1, wherein the step of encoding the base layer bitstreams is performed by a Discrete Cosine Transform (DCT) method.

8. The video encoding method according to claim 7, wherein after encoding the base layer bitstreams by a Discrete Cosine Transform (DCT) method a DCT coefficient is quantized.

9. The video encoding method according to claim 1, wherein the enhancement layer bitstreams are based on the difference of an original base layer DCT coefficient and a corresponding base layer quantized DCT coefficient.

10. The video encoding method according to claim 1, wherein the base layer bitstream and the N enhancement layer provide to the network are multiplexed.

11. A video decoding method for adapting a video input to a bandwidth of a transmission channel of a network, the method comprising the steps of:

determining number M of enhancement layer bitstreams of said video input capable of being received from said transmission channel of said network;

decoding a base layer bitstream from received video input;

decoding M number of enhancement layer bitstreams from the received video input based on the base layer bitstream, wherein the M received enhancement layer bitstreams complements the base layer bitstream; and

reconstructing the base layer bitstream and N enhancement layer bitstreams.

12. The video decoding method according to claim 11, wherein the determining step includes negotiating with intermediate devices on said network.

13. The video decoding method according to claim 12, wherein negotiating includes determining destination resources.

14. The video decoding method according to claim 11, wherein the step of decoding the base layer bitstreams is performed by a MPEG-1 decoding method.

15. The video decoding method according to claim 11, wherein the step of decoding the base layer bitstreams is performed by a MPEG-2 decoding method.

16. The video decoding method according to claim 11, wherein the step of decoding the base layer bitstreams is performed by a MPEG-4 decoding method.

17. The video decoding method according to claim 11, wherein the step of decoding the base layer bitstreams is performed by a Discrete Cosine Transform (DCT) method.

18. The video decoding method according to claim 17, wherein after decoding the base layer bitstreams by a Discrete Cosine Transform (DCT) method a DCT coefficient is unquantized.

19. The video decoding method according to claim 11, wherein coding of the enhancement layer bitstreams are based on the difference of an original base layer DCT coefficient and a corresponding base layer quantized DCT coefficient.

20. The video decoding method according to claim 11, wherein the base layer bitstream and the M enhancement layers to be reconstructed are de-multiplexed.

21. A video decoding method for adapting a video input to a bandwidth of a receiving apparatus, the method comprising the steps of:
demultiplexing a base layer bitstream and at least one of a plurality of enhancement layer bitstreams received from a network;
decoding the base layer bitstream;
decoding at least one of the plurality of enhancement layer bitstreams based on generated base layer bitstream, wherein the at least one of the plurality of enhancement layer bitstreams enhances the base layer bitstream; and
reconstructing a video output.

22. A video encoding method for encoding enhancement layers based on a base layer bitstream encoded from a video input, the video encoding method comprising the steps of:
taking a difference between an original DCT coefficient and a reference point; and
dividing the difference between the original DCT coefficient and the reference point into N bit-planes.

23. The video encoding method according to claim 22, wherein RUN and EOP symbols represents the N bit-planes of a DCT block.

24. The video encoding method according to claim 23, wherein the RUN and EOP symbols are encoded.

25. The video encoding method according to claim 24, wherein a sign bit is encoded if the DCT difference is equal to zero or the sign of the DCT difference is the same as the corresponding base layer bitstream data.

26. A video decoding method for reconstructing DCT coefficients M enhancement layers of N enhancement layers have been received, wherein $M \leq N$, comprising:

means for taking a reconstruction difference as a decoded difference and a portion of a decoded bit-plane;

means for taking a reconstruction point as a reference point and a reconstructed difference; and

determining an optimal reconstruction point.

27. A method of coding motion vectors of a plurality of macroblocks, the method comprising the steps of:

determining an average motion vector from N motion vectors for N macroblocks;

utilizing the determined average motion vector as the motion vector for the N macroblocks; and

encoding 1/N motion vectors in a base layer bitstream.

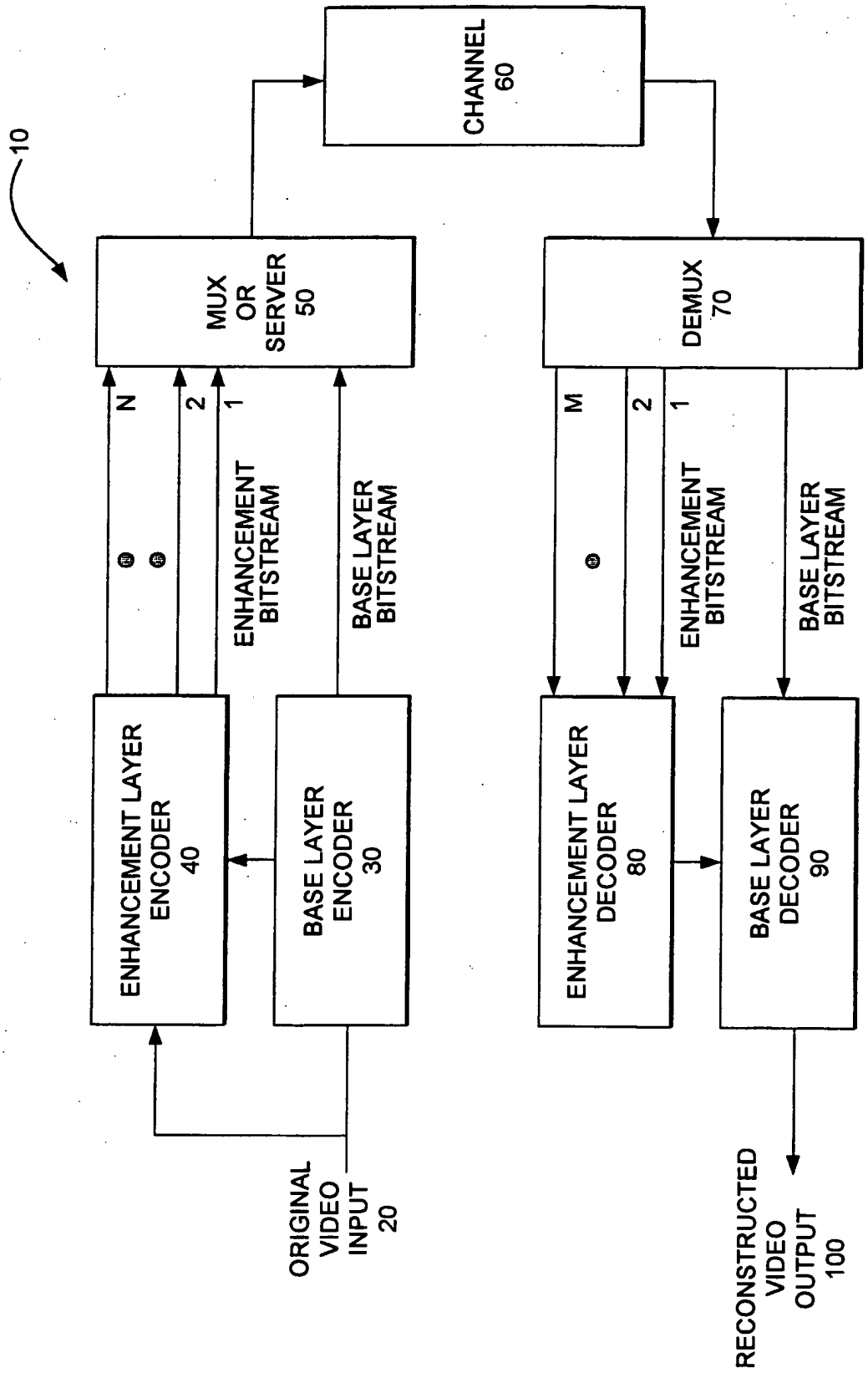


FIG. 1

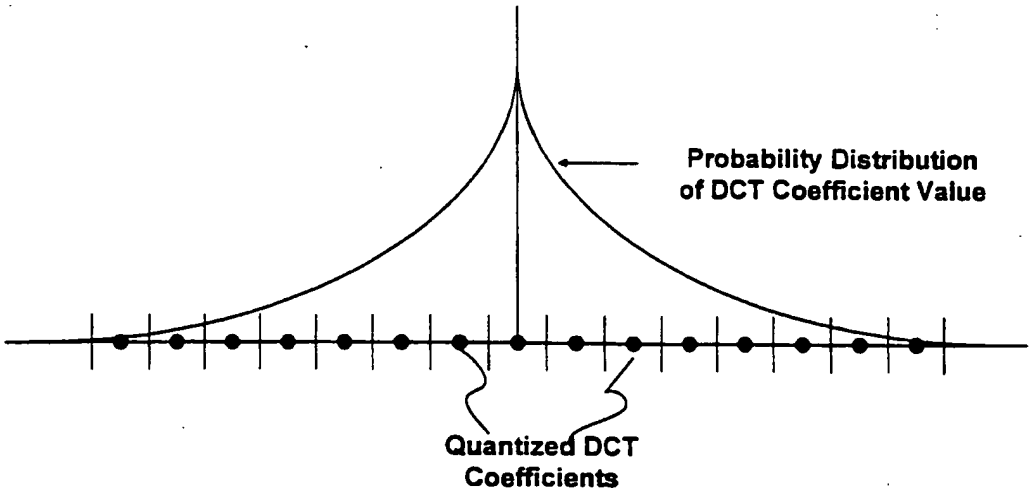


FIG. 2A

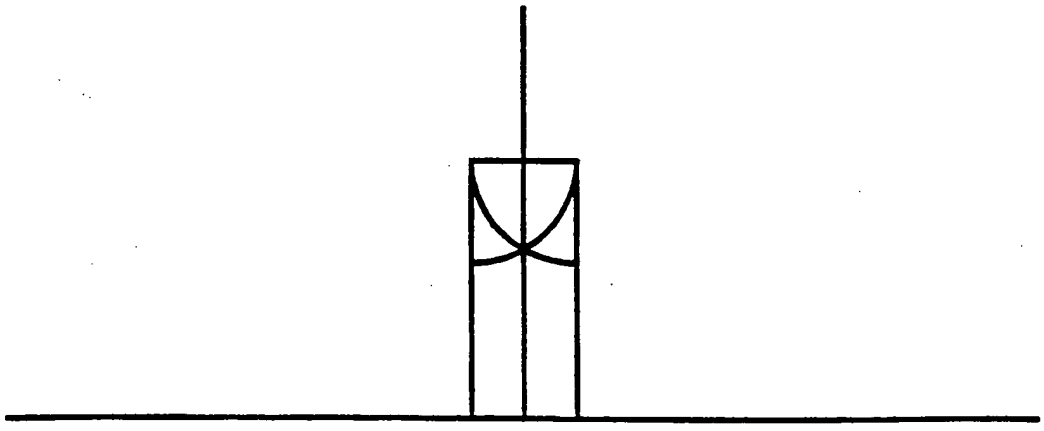


FIG. 2B

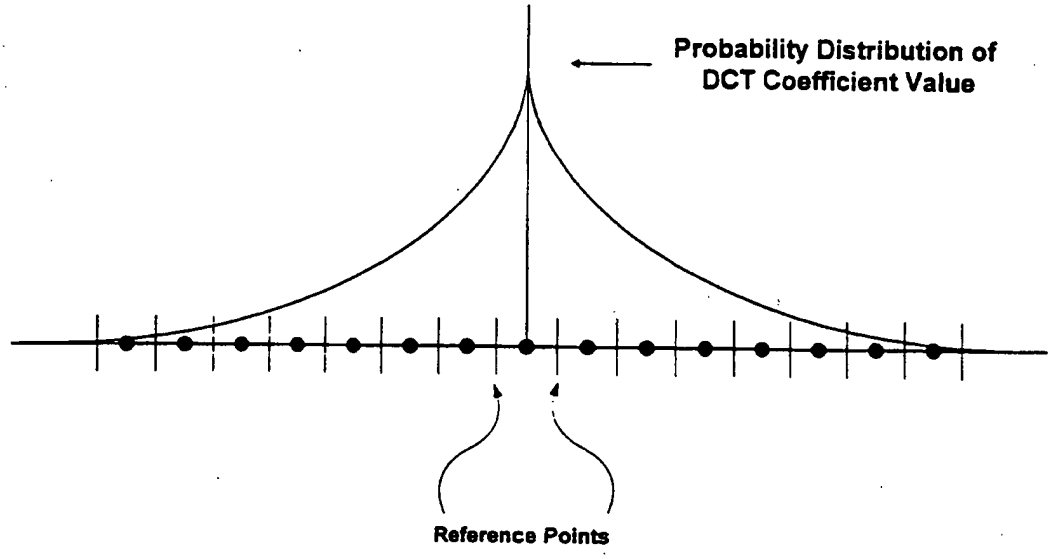


FIG. 3A

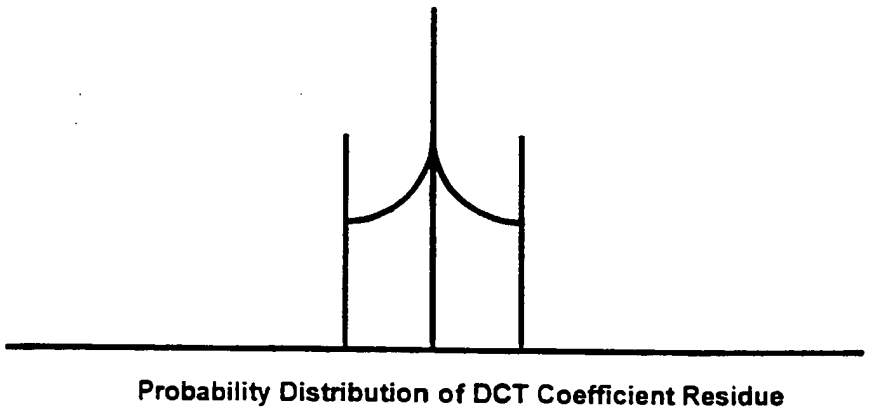


FIG. 3B

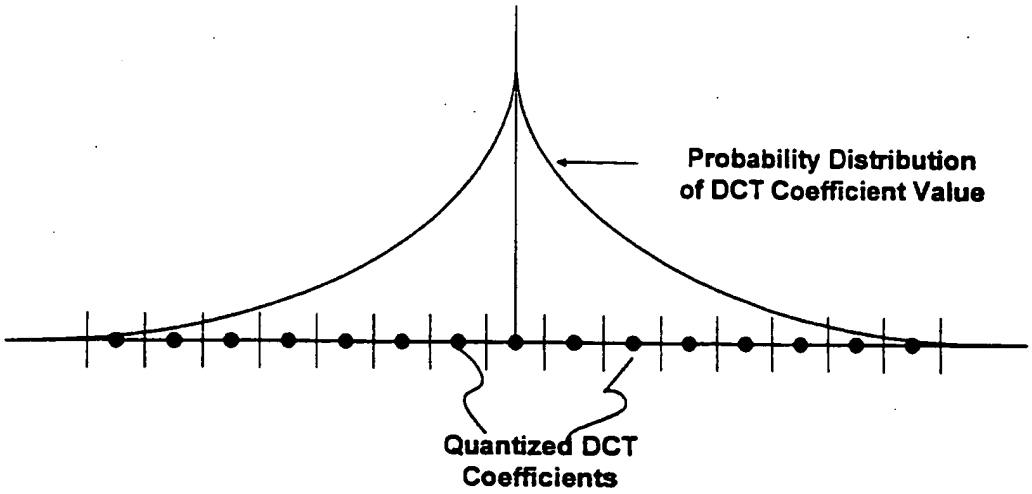


FIG. 3C

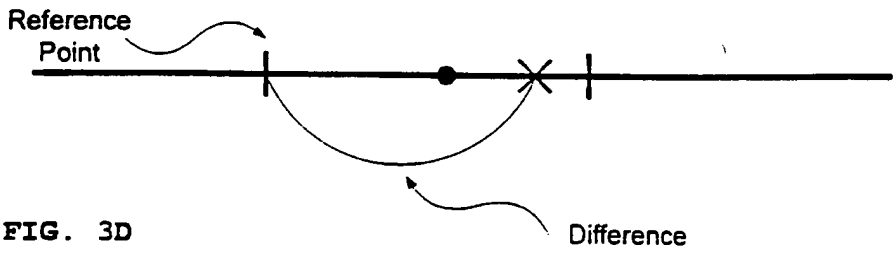


FIG. 3D

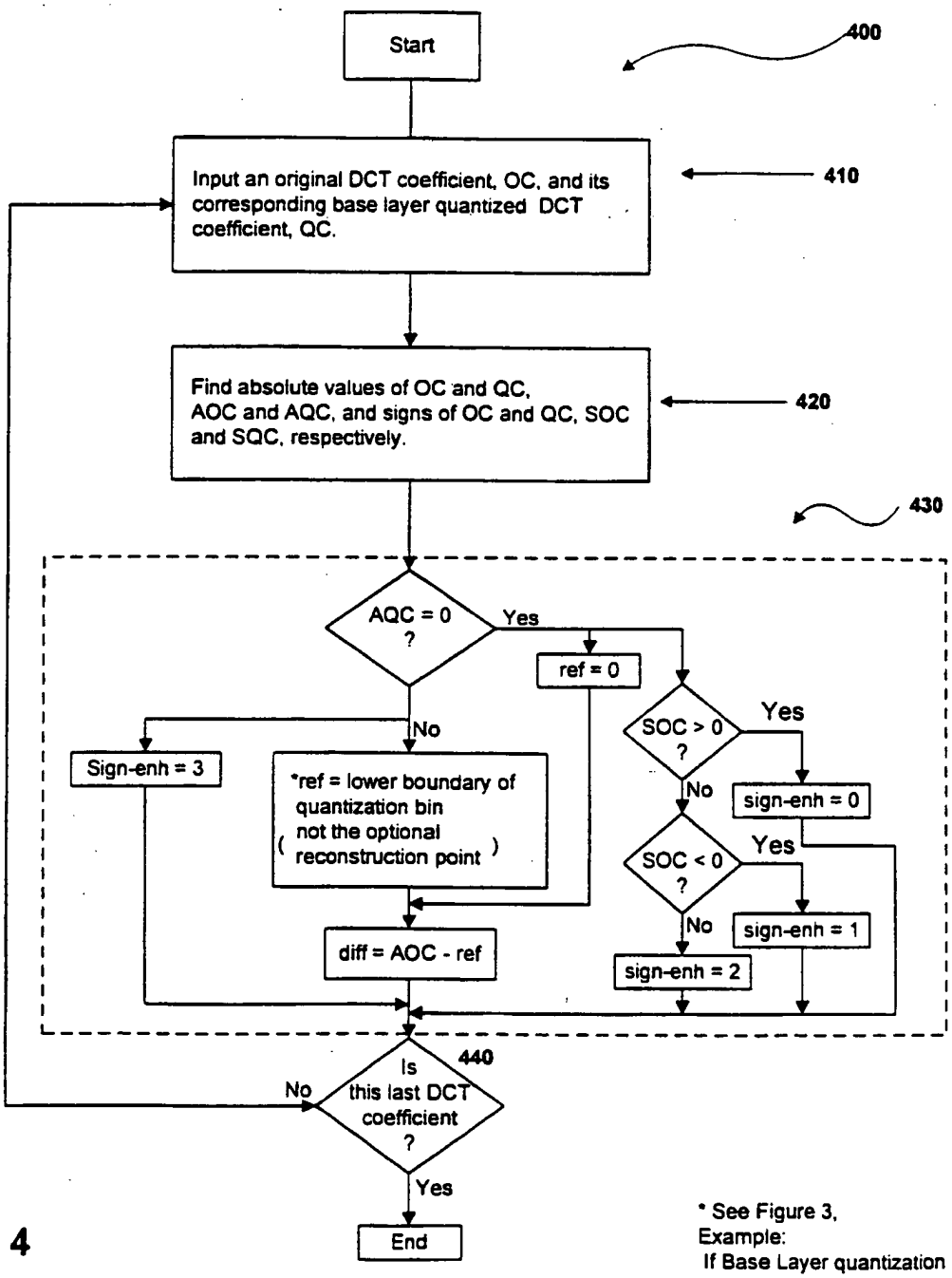


FIG. 4

* See Figure 3,
Example:
If Base Layer quantization is
AQC = AOC / (2*Q)
lower boundary is AQC * (2*Q)
optimal point is AQC*(2*Q) + Q

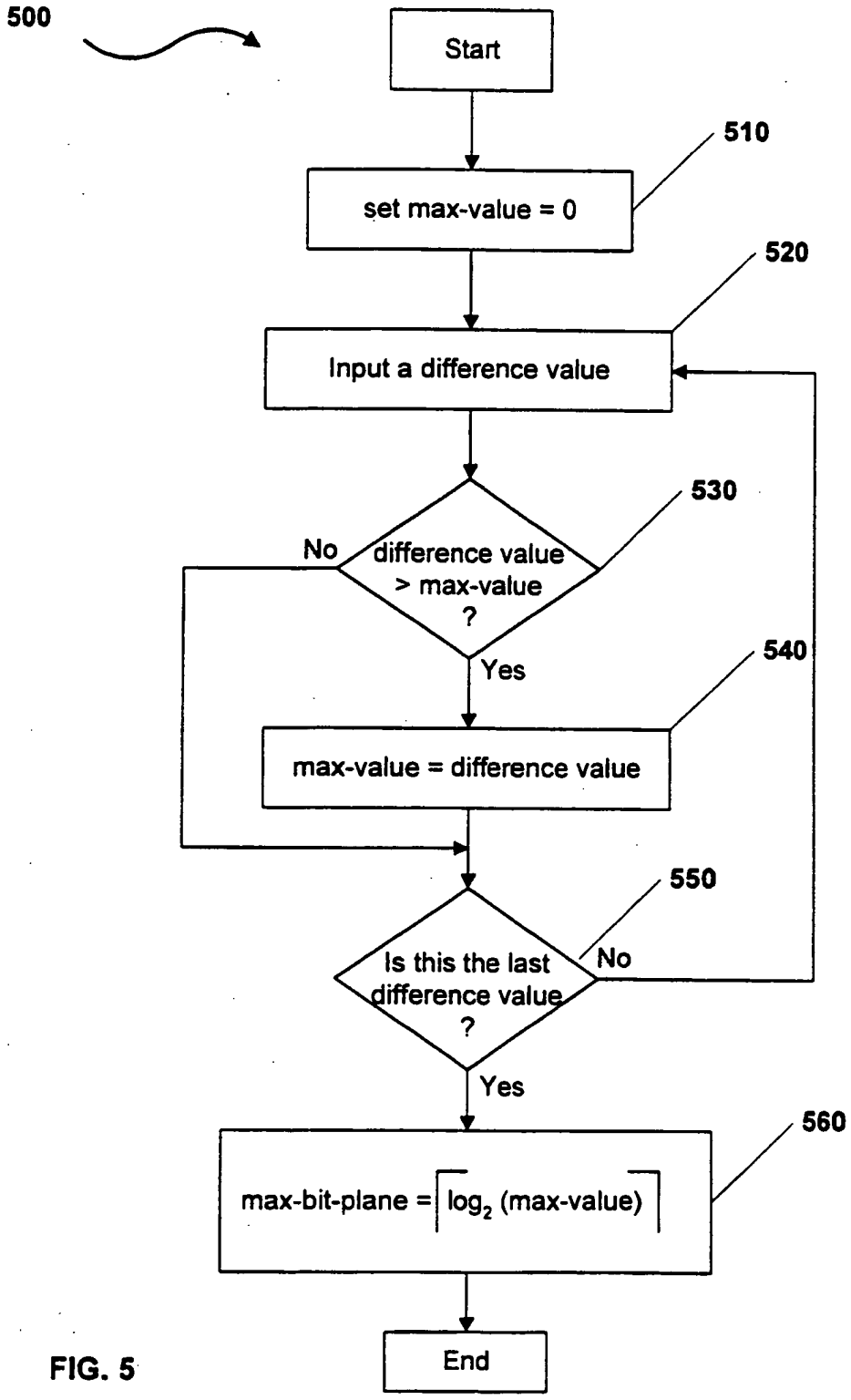

500
Start
510
set max-value = 0
520
Input a difference value
530
No
difference value > max-value ?
Yes
540
max-value = difference value
550
Is this the last difference value ?
No
Yes
560
max-bit-plane = $\lceil \log_2 (\text{max-value}) \rceil$
End

FIG. 5

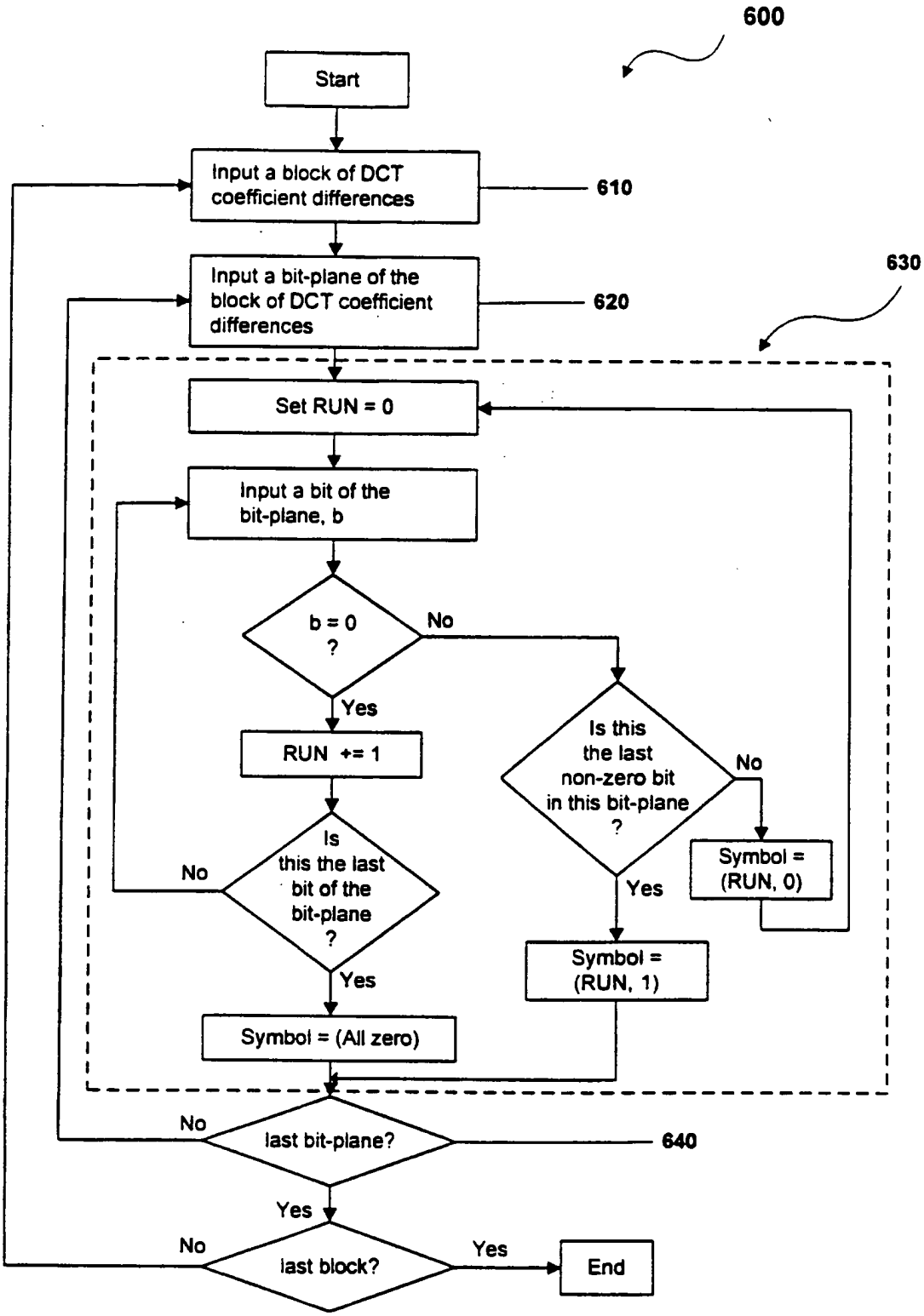

600
Start
Input a block of DCT coefficient differences
610
Input a bit-plane of the block of DCT coefficient differences
620
630
Set RUN = 0
Input a bit of the bit-plane, b
b = 0 ?
No
Yes
RUN += 1
Is this the last non-zero bit in this bit-plane ?
No
Yes
Symbol = (RUN, 0)
Is this the last bit of the bit-plane ?
No
Yes
Symbol = (RUN, 1)
Symbol = (All zero)
No
last bit-plane?
640
Yes
No
last block?
Yes
End

FIG. 6

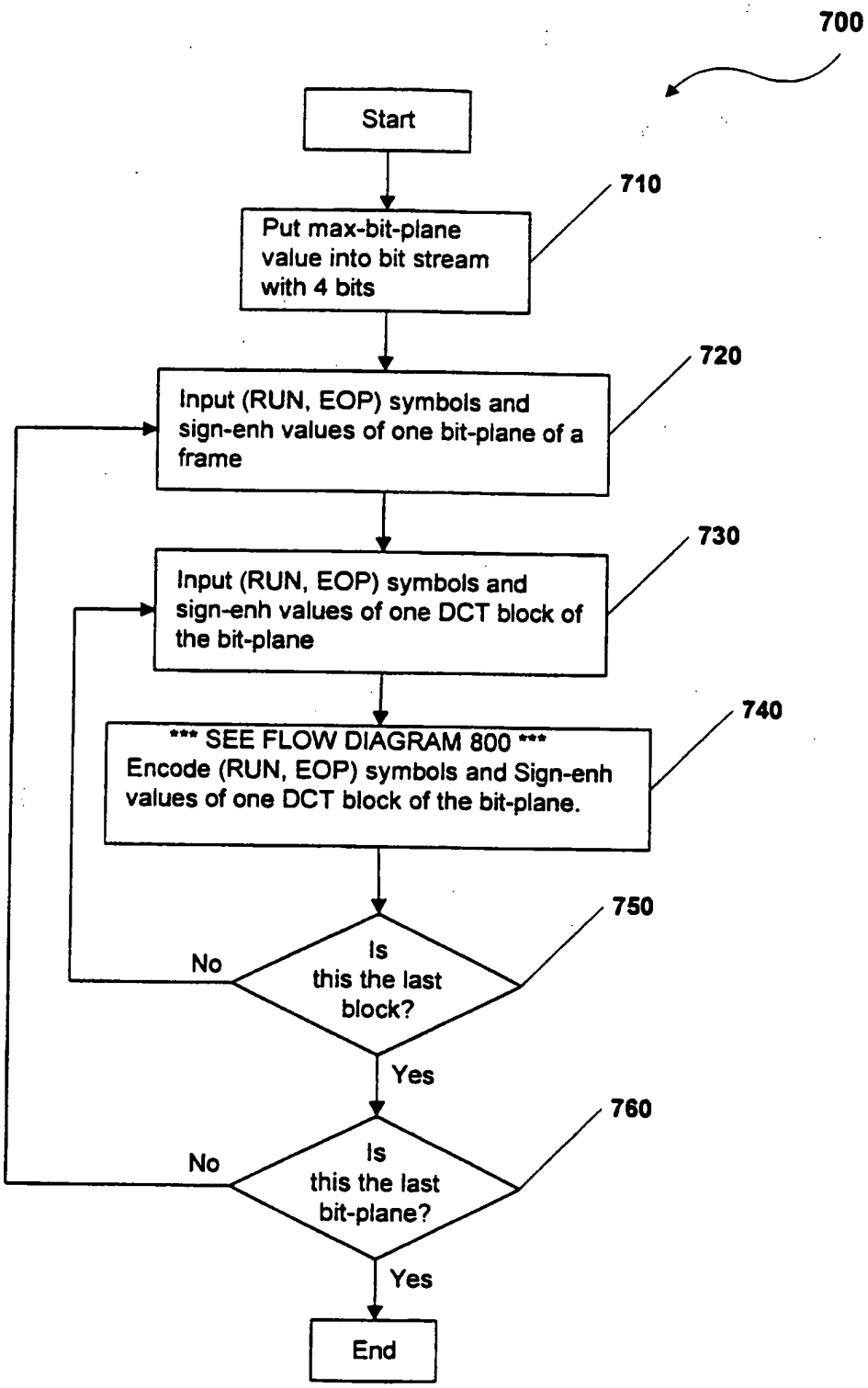

700
Start
710
Put max-bit-plane value into bit stream with 4 bits
720
Input (RUN, EOP) symbols and sign-enh values of one bit-plane of a frame
730
Input (RUN, EOP) symbols and sign-enh values of one DCT block of the bit-plane
740
*** SEE FLOW DIAGRAM 800 ***
Encode (RUN, EOP) symbols and Sign-enh values of one DCT block of the bit-plane.
750
Is this the last block?
No
Yes
760
Is this the last bit-plane?
No
Yes
End

FIG. 7

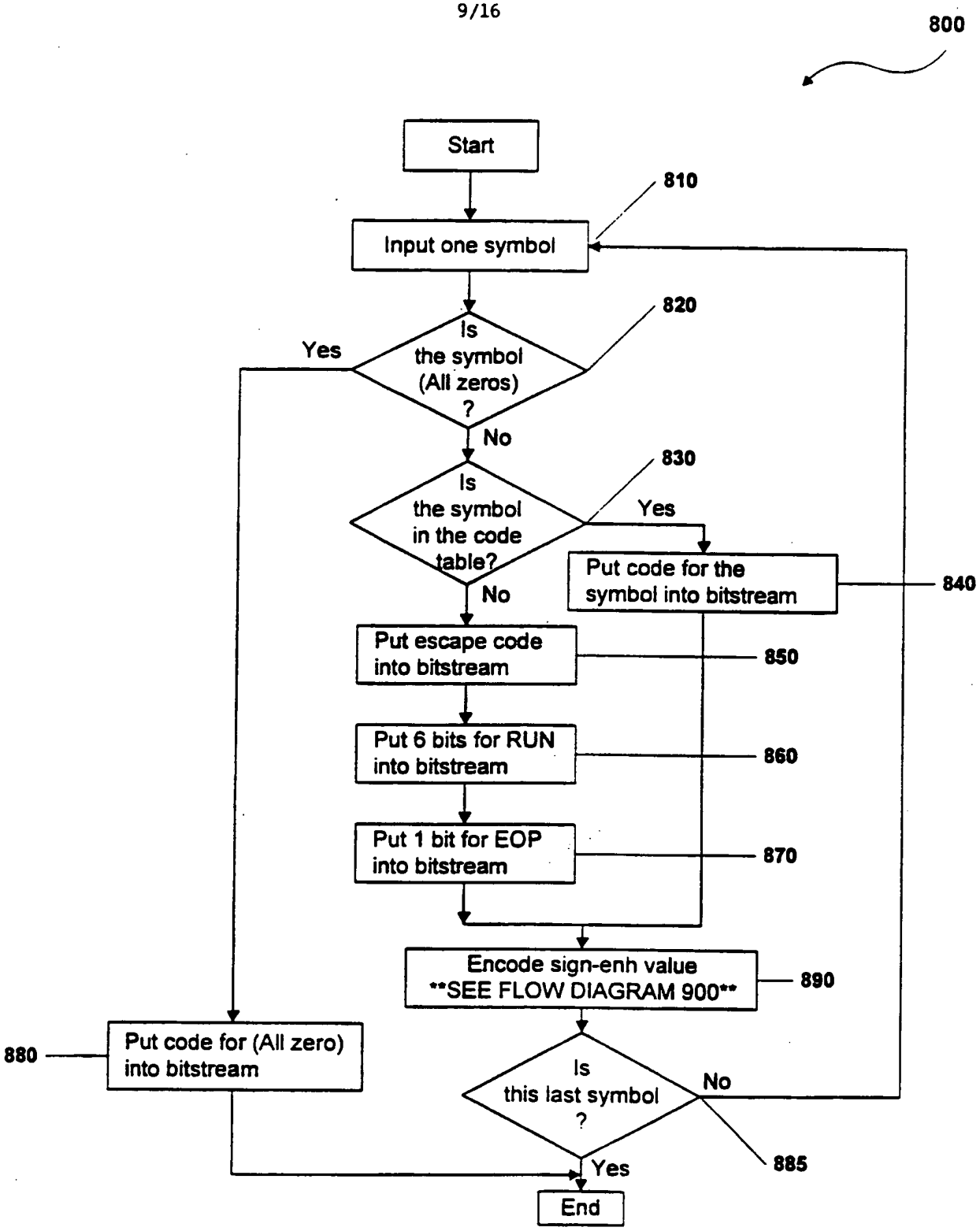

800
Start
810
Input one symbol
820
Is the symbol (All zeros) ?
Yes
No
830
Is the symbol in the code table?
Yes
No
Put code for the symbol into bitstream
840
Put escape code into bitstream
850
Put 6 bits for RUN into bitstream
860
Put 1 bit for EOP into bitstream
870
Encode sign-enh value **SEE FLOW DIAGRAM 900**
890
Put code for (All zero) into bitstream
880
Is this last symbol ?
No
885
Yes
End

FIG. 8

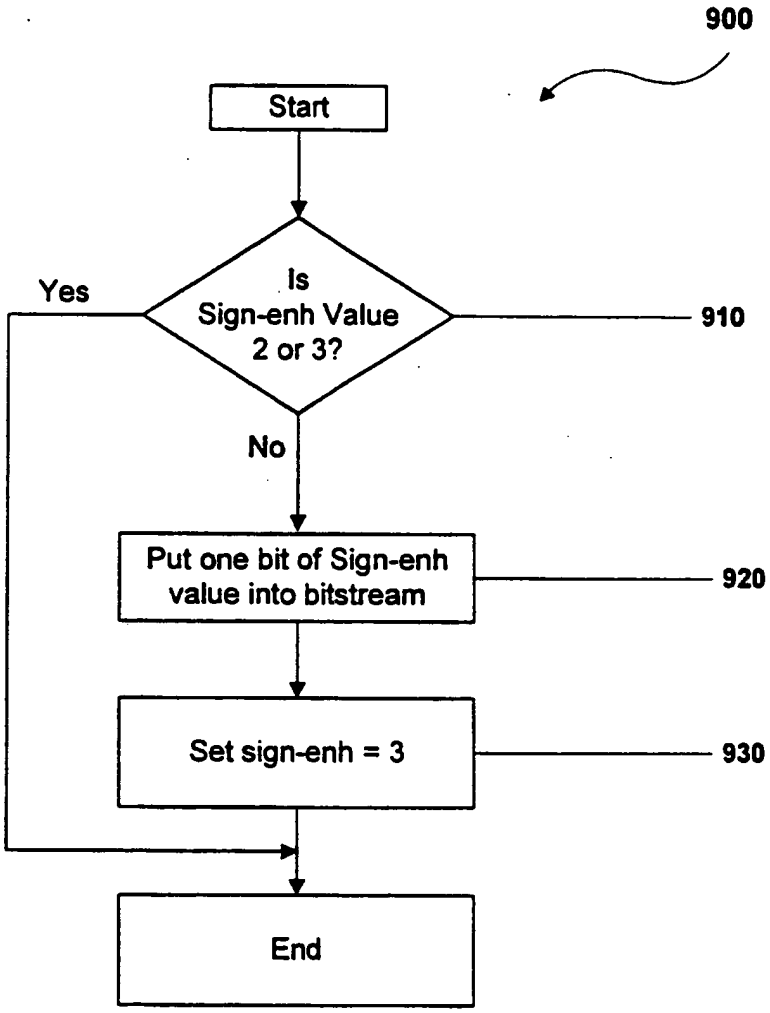

900
Start
Is
Sign-enh Value
2 or 3?
Yes
910
No
Put one bit of Sign-enh
value into bitstream
920
Set sign-enh = 3
930
End

FIG. 9

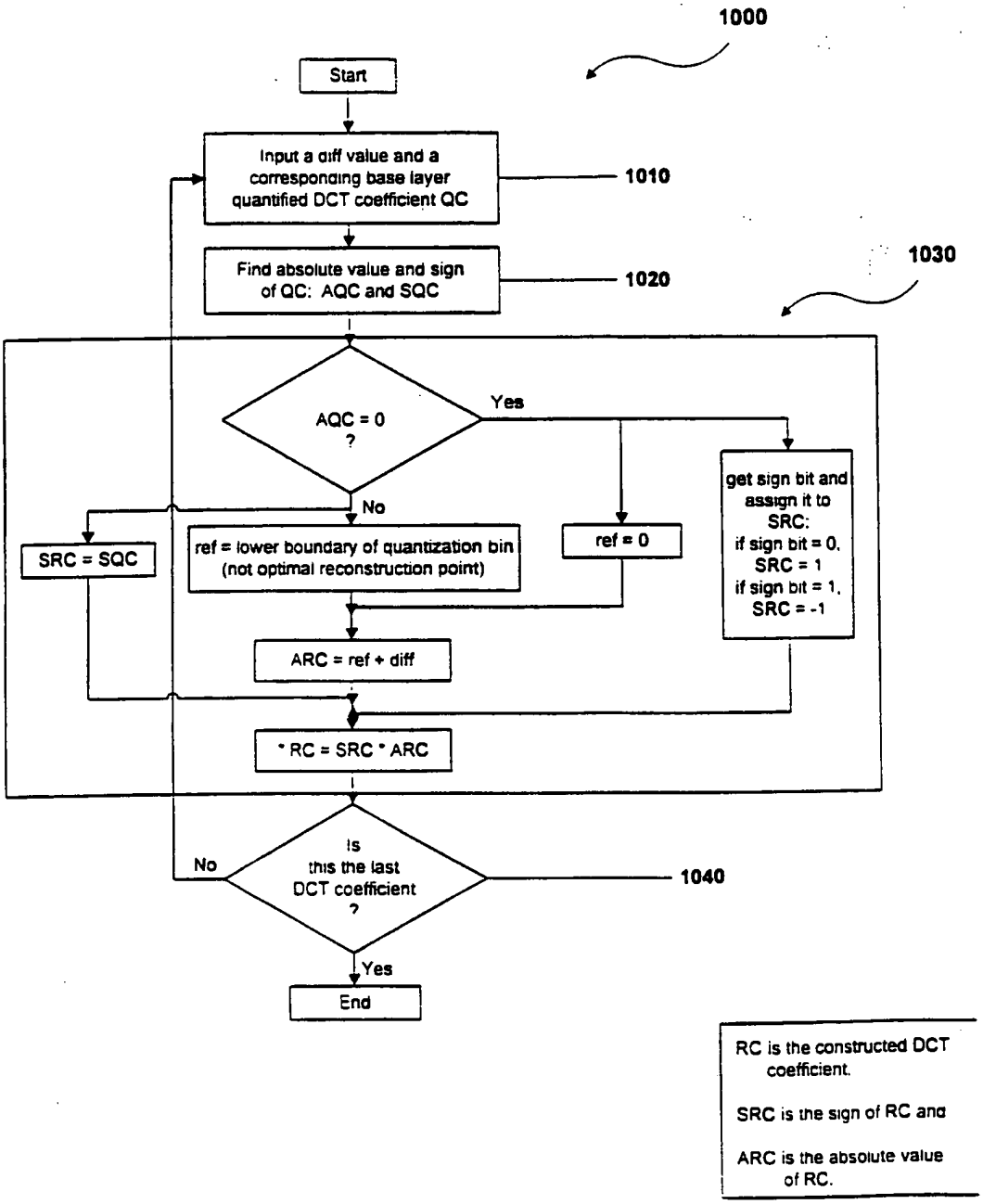

1000
Start
Input a diff value and a corresponding base layer quantified DCT coefficient QC
1010
Find absolute value and sign of QC: AQC and SQC
1020
1030
AQC = 0 ?
Yes
No
SRC = SQC
ref = lower boundary of quantization bin (not optimal reconstruction point)
ref = 0
get sign bit and assign it to SRC:
if sign bit = 0, SRC = 1
if sign bit = 1, SRC = -1
ARC = ref + diff
* RC = SRC * ARC
Is this the last DCT coefficient ?
No
1040
Yes
End
RC is the constructed DCT coefficient.
SRC is the sign of RC and
ARC is the absolute value of RC.

FIG. 10

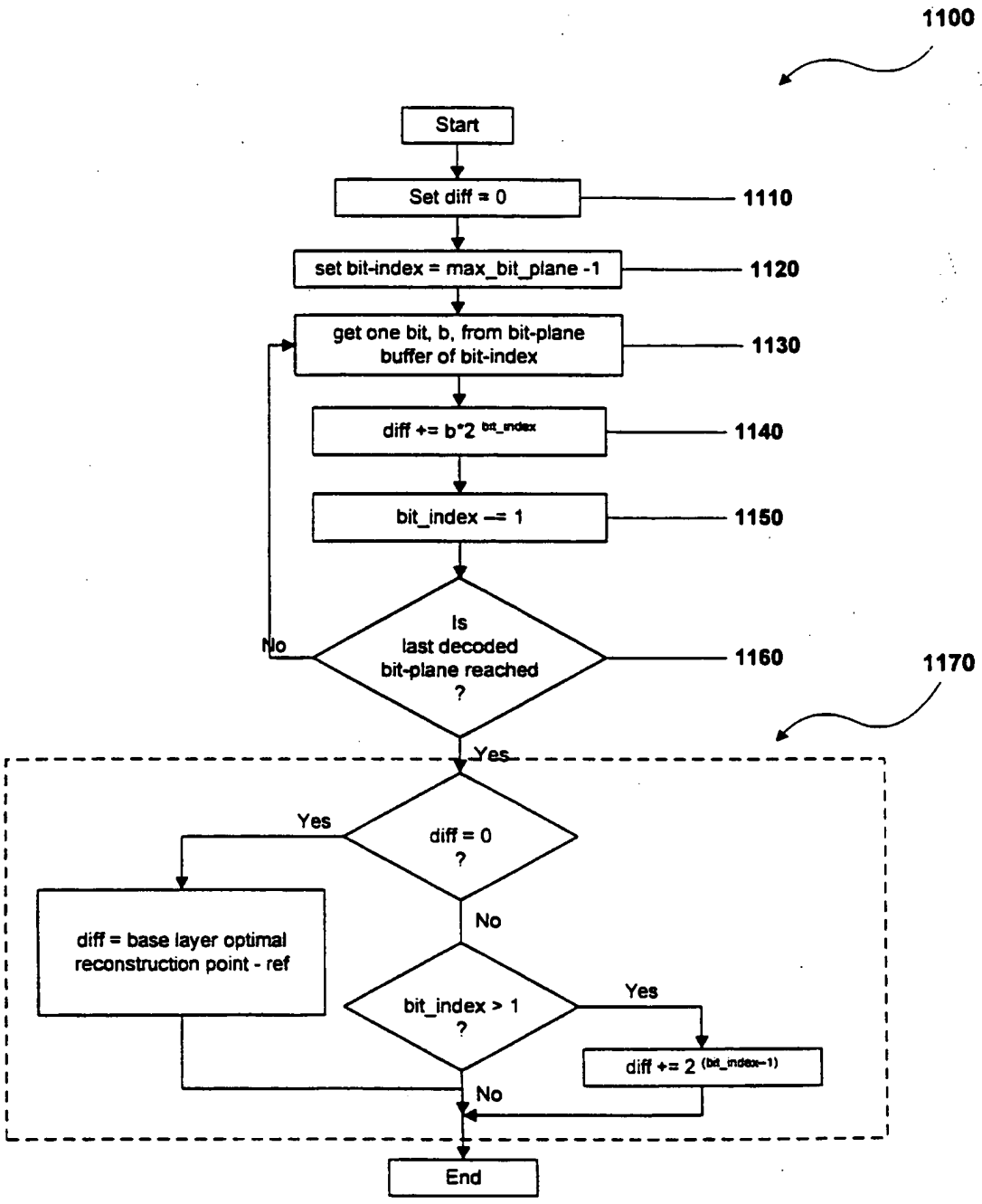

1100
Start
Set diff = 0
1110
set bit-index = max_bit_plane -1
1120
get one bit, b, from bit-plane buffer of bit-index
1130
diff += b*2 bit_index
1140
bit_index -= 1
1150
Is last decoded bit-plane reached ?
1160
No
Yes
1170
diff = 0 ?
Yes
No
diff = base layer optimal reconstruction point - ref
bit_index > 1 ?
Yes
diff += 2 (bit_index-1)
No
End

FIG. 11

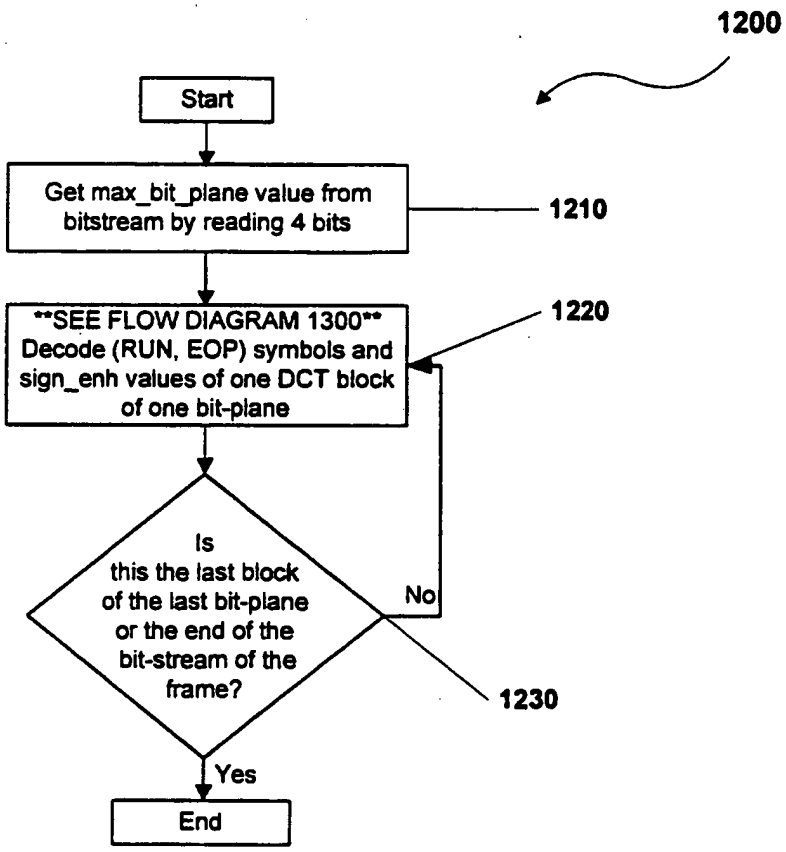

1200
Start
Get max_bit_plane value from bitstream by reading 4 bits
1210
SEE FLOW DIAGRAM 1300 Decode (RUN, EOP) symbols and sign_enh values of one DCT block of one bit-plane
1220
Is this the last block of the last bit-plane or the end of the bit-stream of the frame?
No
1230
Yes
End

FIG. 12

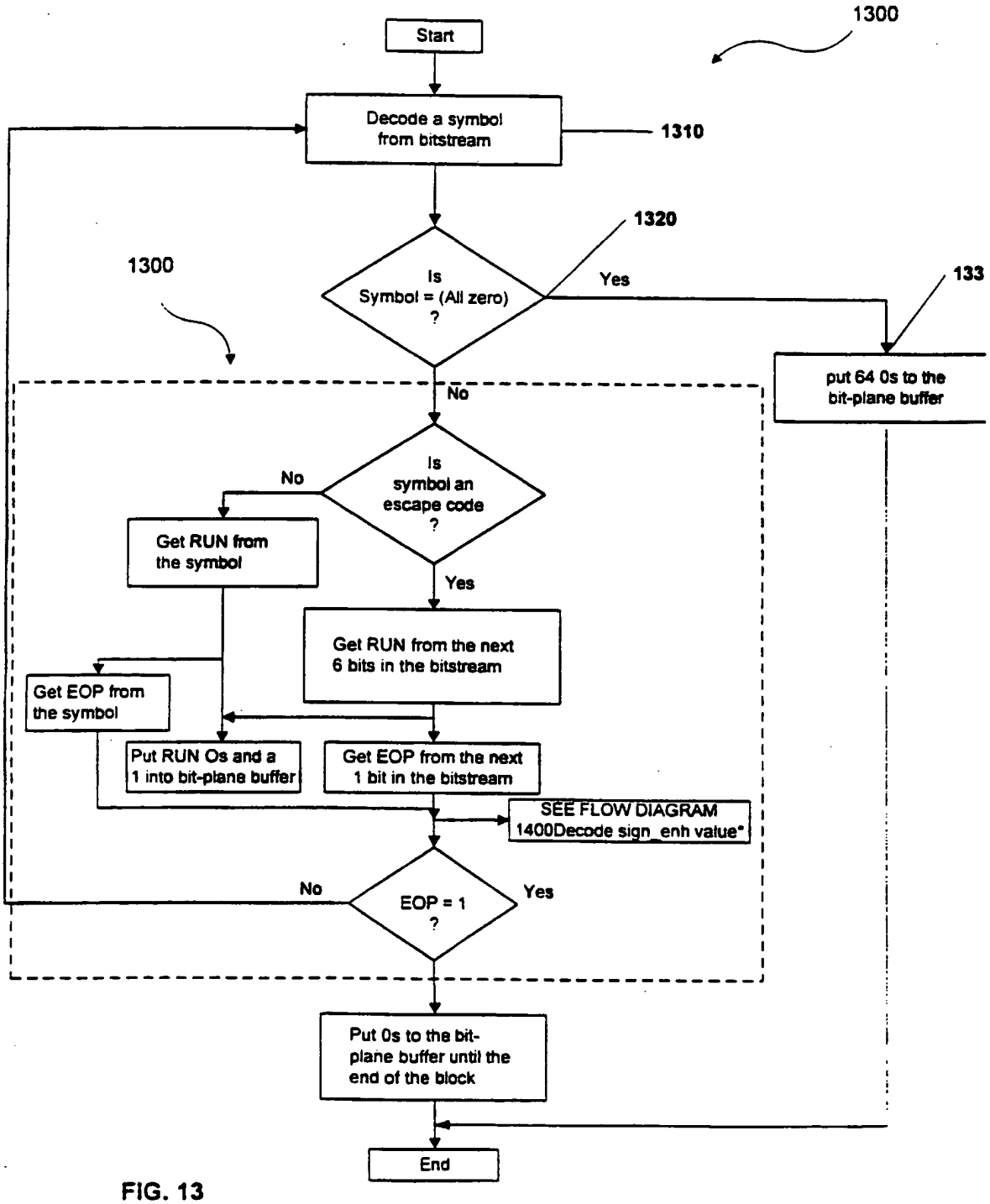

1300
Start
Decode a symbol from bitstream
1310
1320
1300
Is Symbol = (All zero) ?
Yes
133
put 64 0s to the bit-plane buffer
No
Is symbol an escape code ?
No
Get RUN from the symbol
Yes
Get RUN from the next 6 bits in the bitstream
Get EOP from the symbol
Put RUN Os and a 1 into bit-plane buffer
Get EOP from the next 1 bit in the bitstream
SEE FLOW DIAGRAM 1400Decode sign_enh value*
No
EOP = 1 ?
Yes
Put 0s to the bit-plane buffer until the end of the block
End

FIG. 13

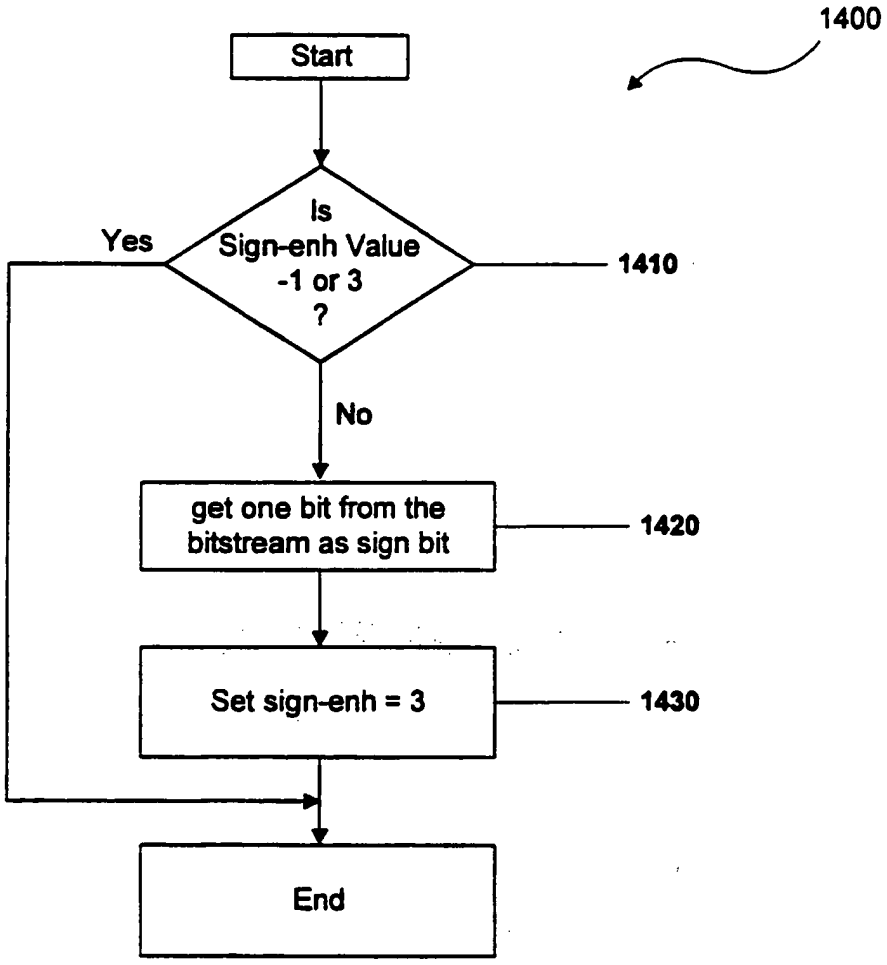

1400
Start
Is
Sign-enh Value
-1 or 3
?
Yes
1410
No
get one bit from the
bitstream as sign bit
1420
Set sign-enh = 3
1430
End

FIG. 14

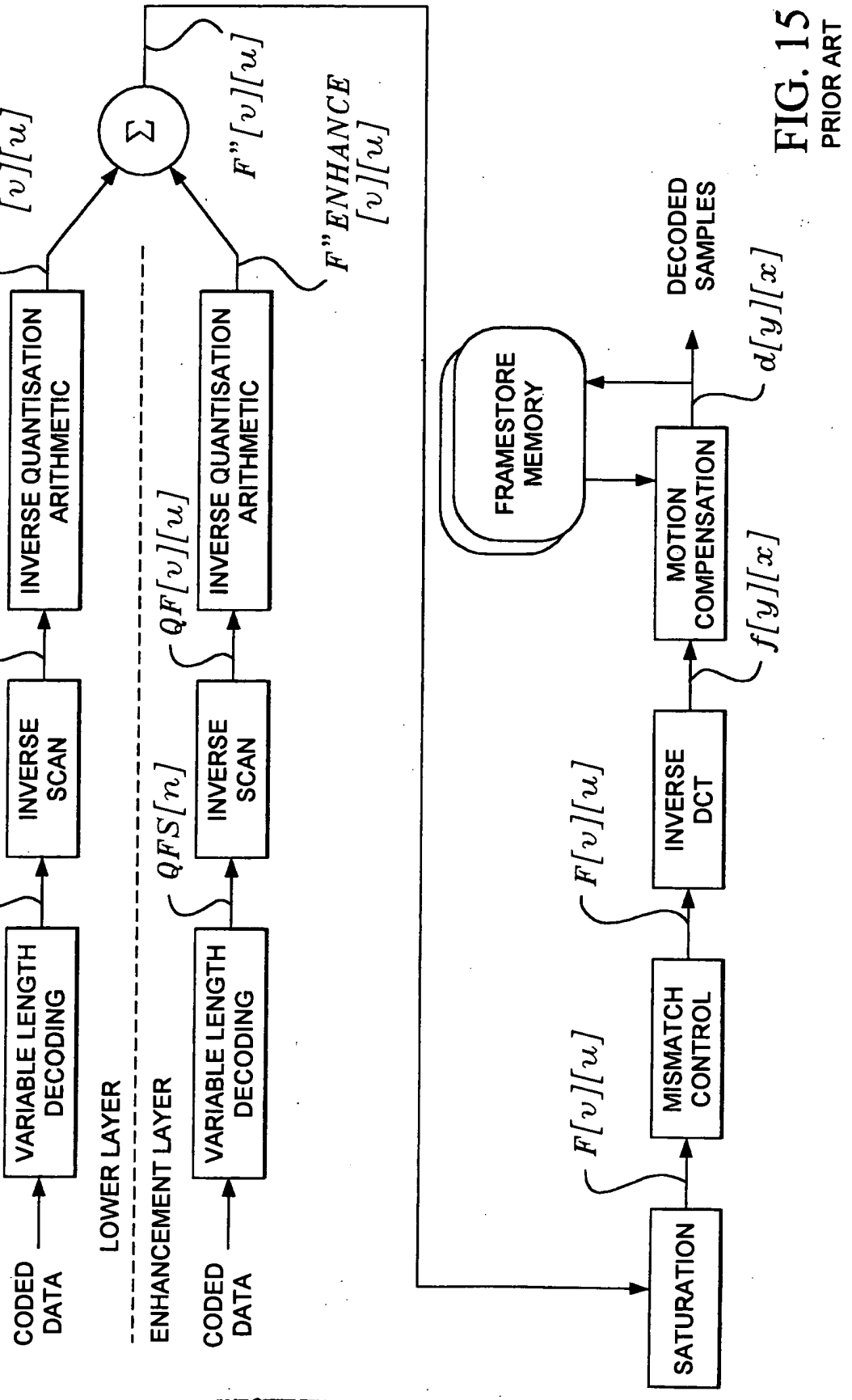


FIG. 15
PRIOR ART

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 7 : H04L 9/00 | A2

(11) International Publication Number: **WO 00/59152**

(43) International Publication Date: 5 October 2000 (05.10.00)

(21) International Application Number: PCT/US00/04983

(22) International Filing Date: 25 February 2000 (25.02.00)

(30) Priority Data:

60/126,614	27 March 1999 (27.03.99)	US
09/290,363	12 April 1999 (12.04.99)	US
09/482,928	13 January 2000 (13.01.00)	US

(71) Applicant: MICROSOFT CORPORATION [US/US]; One Microsoft Way, Redmond, WA 98052 (US).

(72) Inventors: BLINN, Arnold, N.; 9401 NE 27th Street, Bellevue, WA 98004 (US). JONES, Thomas, C.; 23617 NE 6th Street, Redmond, WA 98053–3618 (US).

(74) Agents: ROCCI, Steven, J. et al.; Woodcock Washburn Kurtz Mackiewicz & Norris LLP, 46th floor, One Liberty Place, Philadelphia, PA 19103 (US).

(81) Designated States: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, DM, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

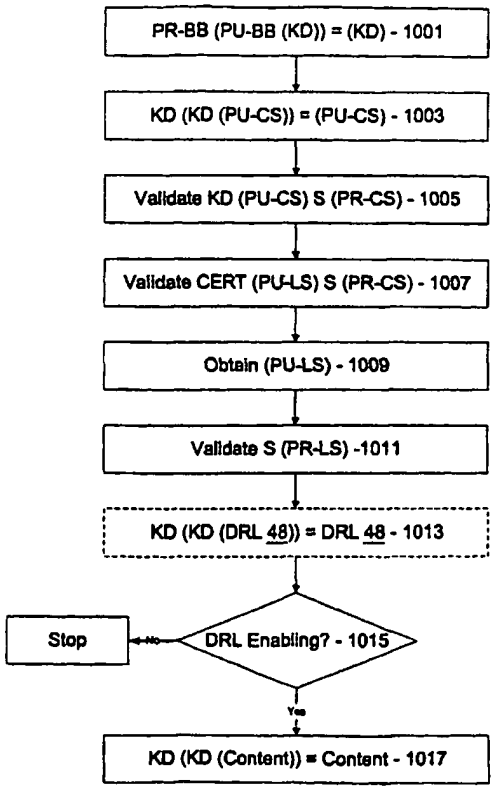
Published

Without international search report and to be republished upon receipt of that report.

(54) Title: METHOD FOR INTERDEPENDENTLY VALIDATING A DIGITAL CONTENT PACKAGE AND A CORRESPONDING DIGITAL LICENSE

(57) Abstract

A method is disclosed for a device to interdependently validate a digital content package having a piece of digital content in an encrypted form, and a corresponding digital license for rendering the digital content. A first key is derived from a source available to the device, and a first digital signature is obtained from the digital content package. The first key is applied to the first digital signature to validate the first digital signature and the digital content package. A second key is derived based on the first digital signature, and a second digital signature is obtained from the license. The second key is applied to the second digital signature to validate the second digital signature and the license.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

METHOD FOR INTERDEPENDENTLY VALIDATING A DIGITAL CONTENT PACKAGE AND A CORRESPONDING DIGITAL LICENSE

CROSS-REFERENCE TO RELATED APPLICATIONS

This application is a continuation of U.S. Patent Application No. 09/290,363, filed April 12, 1999 and entitled "ENFORCEMENT ARCHITECTURE AND METHOD FOR DIGITAL RIGHTS MANAGEMENT", and claims the benefit of U.S. Provisional Application No. 60/21,614, filed March 27, 1999 and entitled "ENFORCEMENT ARCHITECTURE AND METHOD FOR DIGITAL RIGHTS MANAGEMENT", both of which are hereby incorporated by reference.

TECHNICAL FIELD

The present invention relates to an architecture for enforcing rights in digital content. More specifically, the present invention relates to such an enforcement architecture that allows access to encrypted digital content only in accordance with parameters specified by license rights acquired by a user of the digital content.

BACKGROUND OF THE INVENTION

Digital rights management and enforcement is highly desirable in connection with digital content such as digital audio, digital video, digital text, digital data, digital multimedia, etc., where such digital content is to be distributed to users. Typical modes of distribution include tangible devices such as a magnetic (floppy) disk, a magnetic tape, an optical (compact) disk (CD), etc., and intangible media such as an electronic bulletin board, an electronic network, the Internet, etc. Upon being received by the user, such user renders or 'plays' the digital content with the aid of an appropriate rendering device such as a media player on a personal computer or the like.

Typically, a content owner or rights-owner, such as an author, a publisher, a broadcaster, etc. (hereinafter "content owner"), wishes to distribute such digital content to a user or recipient in exchange for a license fee or some other consideration. Such content owner, given the choice, would likely wish to restrict what

the user can do with such distributed digital content. For example, the content owner would like to restrict the user from copying and re-distributing such content to a second user, at least in a manner that denies the content owner a license fee from such second user.

In addition, the content owner may wish to provide the user with the flexibility to purchase different types of use licenses at different license fees, while at the same time holding the user to the terms of whatever type of license is in fact purchased. For example, the content owner may wish to allow distributed digital content to be played only a limited number of times, only for a certain total time, only on a certain type of machine, only on a certain type of media player, only by a certain type of user, etc.

However, after distribution has occurred, such content owner has very little if any control over the digital content. This is especially problematic in view of the fact that practically every new or recent personal computer includes the software and hardware necessary to make an exact digital copy of such digital content, and to download such exact digital copy to a write-able magnetic or optical disk, or to send such exact digital copy over a network such as the Internet to any destination.

Of course, as part of the legitimate transaction where the license fee was obtained, the content owner may require the user of the digital content to promise not to re-distribute such digital content. However, such a promise is easily made and easily broken. A content owner may attempt to prevent such re-distribution through any of several known security devices, usually involving encryption and decryption. However, there is likely very little that prevents a mildly determined user from decrypting encrypted digital content, saving such digital content in an un-encrypted form, and then re-distributing same.

A need exists, then, for providing an enforcement architecture and method that allows the controlled rendering or playing of arbitrary forms of digital content, where such control is flexible and definable by the content owner of such digital content. A need also exists for providing a controlled rendering environment

on a computing device such as a personal computer, where the rendering environment includes at least a portion of such enforcement architecture. Such controlled rendering environment allows that the digital content will only be rendered as specified by the content owner, even though the digital content is to be rendered on a computing device which is not under the control of the content owner.

Further, a need exists for a trusted component running on the computing device, where the trusted component enforces the rights of the content owner on such computing device in connection with a piece of digital content, even against attempts by the user of such computing device to access such digital content in ways not permitted by the content owner. As but one example, such a trusted software component prevents a user of the computing device from making a copy of such digital content, except as otherwise allowed for by the content owner thereof.

SUMMARY OF THE INVENTION

The aforementioned needs are satisfied at least in part by an enforcement architecture and method for digital rights management, where the architecture and method enforce rights in protected (secure) digital content available on a medium such as the Internet, an optical disk, etc. For purposes of making content available, the architecture includes a content server from which the digital content is accessible over the Internet or the like in an encrypted form. The content server may also supply the encrypted digital content for recording on an optical disk or the like, wherein the encrypted digital content may be distributed on the optical disk itself. At the content server, the digital content is encrypted using an encryption key, and public / private key techniques are employed to bind the digital content with a digital license at the user's computing device or client machine.

When a user attempts to render the digital content on a computing device, the rendering application invokes a Digital Rights Management (DRM) system on such user's computing device. If the user is attempting to render the digital content for the first time, the DRM system either directs the user to a license server to obtain a license to render such digital content in the manner sought, or transparently obtains

such license from such license server without any action necessary on the part of the user. The license includes:

- a decryption key (KD) that decrypts the encrypted digital content;
- a description of the rights (play, copy, etc.) conferred by the license and related conditions (begin date, expiration date, number of plays, etc.), where such description is in a digitally readable form; and
- a digital signature that ensures the integrity of the license.

The user cannot decrypt and render the encrypted digital content without obtaining such a license from the license server. The obtained license is stored in a license store in the user's computing device.

Importantly, the license server only issues a license to a DRM system that is 'trusted' (i.e., that can authenticate itself). To implement 'trust', the DRM system is equipped with a 'black box' that performs decryption and encryption functions for such DRM system. The black box includes a public / private key pair, a version number and a unique signature, all as provided by an approved certifying authority. The public key is made available to the license server for purposes of encrypting portions of the issued license, thereby binding such license to such black box. The private key is available to the black box only, and not to the user or anyone else, for purposes of decrypting information encrypted with the corresponding public key. The DRM system is initially provided with a black box with a public / private key pair, and the user is prompted to download from a black box server an updated secure black box when the user first requests a license. The black box server provides the updated black box, along with a unique public/private key pair. Such updated black box is written in unique executable code that will run only on the user's computing device, and is re-updated on a regular basis. When a user requests a license, the client machine sends the black box public key, version number, and signature to the license server, and such license server issues a license only if the version number is current and the signature is valid. A license request also includes an identification of the digital content for which a license is requested and a key ID that identifies the

decryption key associated with the requested digital content. The license server uses the black box public key to encrypt the decryption key, and the decryption key to encrypt the license terms, then downloads the encrypted decryption key and encrypted license terms to the user's computing device along with a license signature.

Once the downloaded license has been stored in the DRM system license store, the user can render the digital content according to the rights conferred by the license and specified in the license terms. When a request is made to render the digital content, the black box is caused to decrypt the decryption key and license terms, and a DRM system license evaluator evaluates such license terms. The black box decrypts the encrypted digital content only if the license evaluation results in a decision that the requestor is allowed to play such content. The decrypted content is provided to the rendering application for rendering.

BRIEF DESCRIPTION OF THE DRAWINGS

The foregoing summary, as well as the following detailed description of the embodiments of the present invention, will be better understood when read in conjunction with the appended drawings. For the purpose of illustrating the invention, there are shown in the drawings embodiments which are presently preferred. As should be understood, however, the invention is not limited to the precise arrangements and instrumentalities shown. In the drawings:

Fig. 1 is a block diagram showing an enforcement architecture in accordance with one embodiment of the present invention;

Fig. 2 is a block diagram of the authoring tool of the architecture of Fig. 1 in accordance with one embodiment of the present invention;

Fig. 3 is a block diagram of a digital content package having digital content for use in connection with the architecture of Fig. 1 in accordance with one embodiment of the present invention;

Fig. 4 is a block diagram of the user's computing device of Fig. 1 in accordance with one embodiment of the present invention;

Figs. 5A and 5B are flow diagrams showing the steps performed in connection with the Digital Rights Management (DRM) system of the computing device of Fig. 4 to render content in accordance with one embodiment of the present invention;

Fig. 6 is a flow diagram showing the steps performed in connection with the DRM system of Fig. 4 to determine whether any valid, enabling licenses are present in accordance with one embodiment of the present invention;

Fig. 7 is a flow diagram showing the steps performed in connection with the DRM system of Fig. 4 to obtain a license in accordance with one embodiment of the present invention;

Fig. 8 is a block diagram of a digital license for use in connection with the architecture of Fig. 1 in accordance with one embodiment of the present invention;

Fig. 9 is a flow diagram showing the steps performed in connection with the DRM system of Fig. 4 to obtain a new black box in accordance with one embodiment of the present invention;

Fig. 10 is a flow diagram showing the key transaction steps performed in connection with the DRM system of Fig. 4 to validate a license and a piece of digital content and render the content in accordance with one embodiment of the present invention;

Fig. 11 is a block diagram showing the license evaluator of Fig. 4 along with a Digital Rights License (DRL) of a license and a language engine for interpreting the DRL in accordance with one embodiment of the present invention; and

Fig. 12 is a block diagram representing a general purpose computer system in which aspects of the present invention and/or portions thereof may be incorporated.

Detailed Description of the Invention

Referring to the drawings in details, wherein like numerals are used to indicate like elements throughout, there is shown in Fig. 1 an enforcement architecture 10 in accordance with one embodiment of the present invention. Overall, the enforcement architecture 10 allows an owner of digital content 12 to specify license rules that must be satisfied before such digital content 12 is allowed to be rendered on a user's computing device 14. Such license rules are embodied within a digital license 16 that the user / user's computing device 14 (hereinafter, such terms are interchangeable unless circumstances require otherwise) must obtain from the content owner or an agent thereof. The digital content 12 is distributed in an encrypted form, and may be distributed freely and widely. Preferably, the decrypting key (KD) for decrypting the digital content 12 is included with the license 16.

COMPUTER ENVIRONMENT

Fig. 12 and the following discussion are intended to provide a brief general description of a suitable computing environment in which the present invention and/or portions thereof may be implemented. Although not required, the invention is described in the general context of computer-executable instructions, such as program modules, being executed by a computer, such as a client workstation or a server. Generally, program modules include routines, programs, objects, components, data structures and the like that perform particular tasks or implement particular abstract data types. Moreover, it should be appreciated that the invention and/or portions thereof may be practiced with other computer system configurations, including hand-held devices, multi-processor systems, microprocessor-based or programmable consumer electronics, network PCs, minicomputers, mainframe computers and the like. The invention may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote memory storage devices.

As shown in Fig. 12, an exemplary general purpose computing system

includes a conventional personal computer 120 or the like, including a processing unit 121, a system memory 122, and a system bus 18 that couples various system components including the system memory to the processing unit 121. The system bus 18 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. The system memory includes read-only memory (ROM) 19 and random access memory (RAM) 20. A basic input/output system 21 (BIOS), containing the basic routines that help to transfer information between elements within the personal computer 120, such as during start-up, is stored in ROM 19.

The personal computer 120 may further include a hard disk drive 22 for reading from and writing to a hard disk (not shown), a magnetic disk drive 128 for reading from or writing to a removable magnetic disk 129, and an optical disk drive 25 for reading from or writing to a removable optical disk 131 such as a CD-ROM or other optical media. The hard disk drive 22, magnetic disk drive 128, and optical disk drive 25 are connected to the system bus 18 by a hard disk drive interface 27, a magnetic disk drive interface 28, and an optical drive interface 29, respectively. The drives and their associated computer-readable media provide non-volatile storage of computer readable instructions, data structures, program modules and other data for the personal computer 20.

Although the exemplary environment described herein employs a hard disk, a removable magnetic disk 129, and a removable optical disk 131, it should be appreciated that other types of computer readable media which can store data that is accessible by a computer may also be used in the exemplary operating environment. Such other types of media include a magnetic cassette, a flash memory card, a digital video disk, a Bernoulli cartridge, a random access memory (RAM), a read-only memory (ROM), and the like.

A number of program modules may be stored on the hard disk, magnetic disk 129, optical disk 131, ROM 19 or RAM 20, including an operating system 30, one or more application programs 136, other program modules 137 and

program data 138. A user may enter commands and information into the personal computer 120 through input devices such as a keyboard 35 and pointing device 142. Other input devices (not shown) may include a microphone, joystick, game pad, satellite disk, scanner, or the like. These and other input devices are often connected to the processing unit 121 through a serial port interface 41 that is coupled to the system bus, but may be connected by other interfaces, such as a parallel port, game port, or universal serial bus (USB). A monitor 42 or other type of display device is also connected to the system bus 18 via an interface, such as a video adapter 148. In addition to the monitor 42, a personal computer typically includes other peripheral output devices (not shown), such as speakers and printers. The exemplary system of Fig. 12 also includes a host adapter 50, a Small Computer System Interface (SCSI) bus 156, and an external storage device 162 connected to the SCSI bus 156.

The personal computer 120 may operate in a networked environment using logical connections to one or more remote computers, such as a remote computer 149. The remote computer 149 may be another personal computer, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to the personal computer 120, although only a memory storage device 150 has been illustrated in Fig. 12. The logical connections depicted in Fig. 12 include a local area network (LAN) 46 and a wide area network (WAN) 47. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets, and the Internet.

When used in a LAN networking environment, the personal computer 120 is connected to the LAN 46 through a network interface or adapter 48. When used in a WAN networking environment, the personal computer 120 typically includes a modem 49 or other means for establishing communications over the wide area network 47, such as the Internet. The modem 49, which may be internal or external, is connected to the system bus 18 via the serial port interface 41. In a networked environment, program modules depicted relative to the personal computer 120, or portions thereof, may be stored in the remote memory storage device. It will be

appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used.

ARCHITECTURE

Referring again to Fig. 1, in one embodiment of the present invention, the architecture 10 includes an authoring tool 18, a content-key database 20, a content server 22, a license server 24, and a black box server 26, as well as the aforementioned user's computing device 14.

ARCHITECTURE - Authoring Tool 18

The authoring tool 18 is employed by a content owner to package a piece of digital content 12 into a form that is amenable for use in connection with the architecture 10 of the present invention. In particular, the content owner provides the authoring tool 18 with the digital content 12, instructions and/or rules that are to accompany the digital content 12, and instructions and/or rules as to how the digital content 12 is to be packaged. The authoring tool 18 then produces a digital content package 12p having the digital content 12 encrypted according to an encryption / decryption key, and the instructions and/or rules that accompany the digital content 12.

In one embodiment of the present invention, the authoring tool 18 is instructed to serially produce several different digital content 12 packages 12p, each having the same digital content 12 encrypted according to a different encryption / decryption key. As should be understood, having several different packages 12p with the same digital content 12 may be useful for tracking the distribution of such packages 12p / content 12 (hereinafter simply "digital content 12", unless circumstances require otherwise). Such distribution tracking is not ordinarily necessary, but may be used by an investigative authority in cases where the digital content 12 has been illegally sold or broadcast.

In one embodiment of the present invention, the encryption / decryption key that encrypts the digital content 12 is a symmetric key, in that the encryption key is also the decryption key (KD). As will be discussed below in more detail, such decryption key (KD) is delivered to a user's computing device 14 in a hidden form as

part of a license 16 for such digital content 12. Preferably, each piece of digital content 12 is provided with a content ID (or each package 12p is provided with a package ID), each decryption key (KD) has a key ID, and the authoring tool 18 causes the decryption key (KD), key ID, and content ID (or package ID) for each piece of digital content 12 (or each package 12p) to be stored in the content-key database 20. In addition, license data regarding the types of licenses 16 to be issued for the digital content 12 and the terms and conditions for each type of license 16 may be stored in the content-key database 20, or else in another database (not shown). Preferably, the license data can be modified by the content owner at a later time as circumstances and market conditions may require.

In use, the authoring tool 18 is supplied with information including, among other things:

- the digital content 12 to be packaged;
- the type and parameters of watermarking and/or fingerprinting to be employed, if any;
- the type and parameters of data compression to be employed, if any;
- the type and parameters of encryption to be employed;
- the type and parameters of serialization to be employed, if any; and
- the instructions and/or rules that are to accompany the digital content 12.

As is known, a watermark is a hidden, computer-readable signal that is added to the digital content 12 as an identifier. A fingerprint is a watermark that is different for each instance. As should be understood, an instance is a version of the digital content 12 that is unique. Multiple copies of any instance may be made, and any copy is of a particular instance. When a specific instance of digital content 12 is illegally sold or broadcast, an investigative authority can perhaps identify suspects according to the watermark / fingerprint added to such digital content 12.

Data compression may be performed according to any appropriate compression algorithm without departing from the spirit and scope of the present

invention. For example, the .mp3 or .wav compression algorithm may be employed. Of course, the digital content 12 may already be in a compressed state, in which case no additional compression is necessary.

The instructions and/or rules that are to accompany the digital content 12 may include practically any appropriate instructions, rules, or other information without departing from the spirit and scope of the present invention. As will be discussed below, such accompanying instructions / rules / information are primarily employed by the user and the user's computing device 14 to obtain a license 16 to render the digital content 12. Accordingly, such accompanying instructions / rules / information may include an appropriately formatted license acquisition script or the like, as will be described in more detail below. In addition, or in the alternative, such accompanying instructions / rules / information may include 'preview' information designed to provide a user with a preview of the digital content 12.

With the supplied information, the authoring tool 18 then produces one or more packages 12p corresponding to the digital content 12. Each package 12p may then be stored on the content server 22 for distribution to the world.

In one embodiment of the present invention, and referring now to Fig. 2, the authoring tool 18 is a dynamic authoring tool 18 that receives input parameters which can be specified and operated on. Accordingly, such authoring tool 18 can rapidly produce multiple variations of package 12p for multiple pieces of digital content 12. Preferably, the input parameters are embodied in the form of a dictionary 28, as shown, where the dictionary 28 includes such parameters as:

- the name of the input file 29a having the digital content 12;
- the type of encoding that is to take place
- the encryption / decryption key (KD) to be employed,
- the accompanying instructions / rules / information ('header information') to be packaged with the digital content 12 in the package 12p.
- the type of muxing that is to occur: and

- the name of the output file 29b to which the package 12p based on the digital content 12 is to be written.

As should be understood, such dictionary 28 is easily and quickly modifiable by an operator of the authoring tool 18 (human or machine), and therefore the type of authoring performed by the authoring tool 18 is likewise easily and quickly modifiable in a dynamic manner. In one embodiment of the present invention, the authoring tool 18 includes an operator interface (not shown) displayable on a computer screen to a human operator. Accordingly, such operator may modify the dictionary 28 by way of the interface, and further may be appropriately aided and/or restricted in modifying the dictionary 28 by way of the interface.

In the authoring tool 18, and as seen in Fig. 2, a source filter 18a receives the name of the input file 29a having the digital content 12 from the dictionary 28, and retrieves such digital content 12 from such input file and places the digital content 12 into a memory 29c such as a RAM or the like. An encoding filter 18b then performs encoding on the digital content 12 in the memory 29c to transfer the file from the input format to the output format according to the type of encoding specified in the dictionary 28 (i.e., .wav to .asp, .mp3 to .asp, etc.). and places the encoded digital content 12 in the memory 29c. As shown, the digital content 12 to be packaged (music, e.g.) is received in a compressed format such as the .wav or .mp3 format, and is transformed into a format such as the .asp (active streaming protocol) format. Of course, other input and output formats may be employed without departing from the spirit and scope of the present invention.

Thereafter, an encryption filter 18c encrypts the encoded digital content 12 in the memory 29c according to the encryption / decryption key (KD) specified in the dictionary 28, and places the encrypted digital content 12 in the memory 29c. A header filter 18d then adds the header information specified in the dictionary 28 to the encrypted digital content 12 in the memory 29c.

As should be understood, depending on the situation, the package 12p may include multiple streams of temporally aligned digital content 12 (one stream

being shown in Fig. 2), where such multiple streams are multiplexed (i.e., 'muxed'). Accordingly, a mux filter 18e performs muxing on the header information and encrypted digital content 12 in the memory 29c according to the type of muxing specified in the dictionary 28, and places the result in the memory 29c. A file writer filter 18f then retrieves the result from the memory 29c and writes such result to the output file 29b specified in the dictionary 28 as the package 12p.

It should be noted that in certain circumstances, the type of encoding to be performed will not normally change. Since the type of muxing typically is based on the type of encoding, it is likewise the case that the type of muxing will not normally change, either. If this is in fact the case, the dictionary 28 need not include parameters on the type of encoding and/or the type of muxing. Instead, it is only necessary that the type of encoding be 'hardwired' into the encoding filter and/or that the type of muxing be 'hardwired' into the mux filter. Of course, as circumstance require, the authoring tool 18 may not include all of the aforementioned filters, or may include other filters, and any included filter may be hardwired or may perform its function according to parameters specified in the dictionary 28, all without departing from the spirit and scope of the present invention.

Preferably, the authoring tool 18 is implemented on an appropriate computer, processor, or other computing machine by way of appropriate software. The structure and operation of such machine and such software should be apparent based on the disclosure herein and therefore do not require any detailed discussion in the present disclosure.

ARCHITECTURE - Content Server 22

Referring again to Fig. 1, in one embodiment of the present invention, the content server 22 distributes or otherwise makes available for retrieval the packages 12p produced by the authoring tool 18. Such packages 12p may be distributed as requested by the content server 22 by way of any appropriate distribution channel without departing from the spirit and scope of the present invention. For example, such distribution channel may be the Internet or another network, an electronic bulletin

board, electronic mail, or the like. In addition, the content server 22 may be employed to copy the packages 12p onto magnetic or optical disks or other storage devices, and such storage devices may then be distributed.

It will be appreciated that the content server 22 distributes packages 12p without regard to any trust or security issues. As discussed below, such issues are dealt with in connection with the license server 24 and the relationship between such license server 24 and the user's computing device 14. In one embodiment of the present invention, the content server 22 freely releases and distributes packages 12p having digital content 12 to any distributee requesting same. However, the content server 22 may also release and distribute such packages 12p in a restricted manner without departing from the spirit and scope of the present invention. For example, the content server 22 may first require payment of a pre-determined distribution fee prior to distribution, or may require that a distributee identify itself, or may indeed make a determination of whether distribution is to occur based on an identification of the distributee.

In addition, the content server 22 may be employed to perform inventory management by controlling the authoring tool 18 to generate a number of different packages 12p in advance to meet an anticipated demand. For example, the server could generate 100 packages 12p based on the same digital content 12, and serve each package 12p 10 times. As supplies of packages 12p dwindle to 20, for example, the content server 22 may then direct the authoring tool 18 to generate 80 additional packages 12p, again for example.

Preferably, the content server 22 in the architecture 10 has a unique public / private key pair (PU-CS, PR-CS) that is employed as part of the process of evaluating a license 16 and obtaining a decryption key (KD) for decrypting corresponding digital content 12, as will be explained in more detail below. As is known, a public / private key pair is an asymmetric key, in that what is encrypted in one of the keys in the key pair can only be decrypted by the other of the keys in the key pair. In a public / private key pair encryption system, the public key may be made

known to the world, but the private key should always be held in confidence by the owner of such private key. Accordingly, if the content server 22 encrypts data with its private key (PR-CS), it can send the encrypted data out into the world with its public key (PU-CS) for decryption purposes. Correspondingly, if an external device wants to send data to the content server 22 so that only such content server 22 can decrypt such data, such external device must first obtain the public key of the content server 22 (PU-CS) and then must encrypt the data with such public key. Accordingly, the content server 22 (and only the content server 22) can then employ its private key (PR-CS) to decrypt such encrypted data.

As with the authoring tool 18, the content server 22 is implemented on an appropriate computer, processor, or other computing machine by way of appropriate software. The structure and operation of such machine and such software should be apparent based on the disclosure herein and therefore do not require any detailed discussion in the present disclosure. Moreover, in one embodiment of the present invention, the authoring tool 18 and the content server 22 may reside on a single computer, processor, or other computing machine, each in a separate work space. It should be recognized, moreover, that the content server 22 may in certain circumstances include the authoring tool 18 and/or perform the functions of the authoring tool 18, as discussed above.

Structure of Digital Content Package 12p

Referring now to Fig. 3, in one embodiment of the present invention, the digital content package 12p as distributed by the content server 22 includes:

- the digital content 12 encrypted with the encryption / decryption key (KD), as was discussed above (i.e., (KD(CONTENT)));
- the content ID (or package ID) of such digital content 12 (or package 12p);
- the key ID of the decryption key (KD);
- license acquisition information, preferably in an un-encrypted form; and

- the key KD encrypting the content server 22 public key (PU-CS), signed by the content server 22 private key (PR-CS) (i.e., (KD (PU-CS) S (PR-CS))).

With regard to (KD (PU-CS) S (PR-CS)), it is to be understood that such item is to be used in connection with validating the digital content 12 and/or package 12p, as will be explained below. Unlike a certificate with a digital signature (see below), the key (PU-CS) is not necessary to get at (KD (PU-CS)). Instead, the key (PU-CS) is obtained merely by applying the decryption key (KD). Once so obtained, such key (PU-CS) may be employed to test the validity of the signature (S (PR-CS)).

It should also be understood that for such package 12p to be constructed by the authoring tool 18, such authoring tool 18 must already possess the license acquisition information and (KD (PU-CS) S (PR-CS)), presumably as header information supplied by the dictionary 28. Moreover, the authoring tool 18 and the content server 22 must presumably interact to construct (KD (PU-CS) S (PR-CS)). Such interaction may for example include the steps of:

- the content server 22 sending (PU-CS) to the authoring tool 18;
- the authoring tool 18 encrypting (PU-CS) with (KD) to produce (KD (PU-CS));
- the authoring tool 18 sending (KD (PU-CS)) to the content server 22;
- the content server 22 signing (KD (PU-CS)) with (PR-CS) to produce (KD (PU-CS) S (PR-CS)); and
- the content server 22 sending (KD (PU-CS) S (PR-CS)) to the authoring tool 18.

ARCHITECTURE - License Server 24

Referring again to Fig. 1, in one embodiment of the present invention, the license server 24 performs the functions of receiving a request for a license 16 from a user's computing device 14 in connection with a piece of digital content 12,

determining whether the user's computing device 14 can be trusted to honor an issued license 16, negotiating such a license 16, constructing such license 16, and transmitting such license 16 to the user's computing device 14. Preferably, such transmitted license 16 includes the decryption key (KD) for decrypting the digital content 12. Such license server 24 and such functions will be explained in more detail below. Preferably, and like the content server 22, the license server 24 in the architecture 10 has a unique public / private key pair (PU-LS, PR-LS) that is employed as part of the process of evaluating a license 16 and obtaining a decryption key (KD) for decrypting corresponding digital content 12, as will be explained in more detail below.

As with the authoring tool 18 and the content server 22, the license server 24 is implemented on an appropriate computer, processor, or other computing machine by way of appropriate software. The structure and operation of such machine and such software should be apparent based on the disclosure herein and therefore do not require any detailed discussion in the present disclosure. Moreover, in one embodiment of the present invention the authoring tool 18 and/or the content server 22 may reside on a single computer, processor, or other computing machine together with the license server 24, each in a separate work space.

In one embodiment of the present invention, prior to issuance of a license 16, the license server 24 and the content server 22 enter into an agency agreement or the like, wherein the license server 24 in effect agrees to be the licensing authority for at least a portion of the digital content 12 distributed by the content server 22. As should be understood, one content server 22 may enter into an agency agreement or the like with several license servers 24, and/or one license server 24 may enter into an agency agreement or the like with several content servers 22, all without departing from the spirit and scope of the present invention.

Preferably, the license server 24 can show to the world that it does in fact have the authority to issue a license 16 for digital content 12 distributed by the content server 22. To do so, it is preferable that the license server 24 send to the content server 22 the license server 24 public key (PU-LS), and that the content server

22 then send to the license server 24 a digital certificate containing PU-LS as the contents signed by the content server 22 private key (CERT (PU-LS) S (PR-CS)). As should be understood, the contents (PU-LS) in such certificate can only be accessed with the content server 22 public key (PU-CS). As should also be understood, in general, a digital signature of underlying data is an encrypted form of such data, and will not match such data when decrypted if such data has been adulterated or otherwise modified.

As a licensing authority in connection with a piece of digital content 12, and as part of the licensing function, the license server 24 must have access to the decryption key (KD) for such digital content 12. Accordingly, it is preferable that license server 24 have access to the content-key database 20 that has the decryption key (KD), key ID, and content ID (or package ID) for such digital content 12 (or package 12p).

ARCHITECTURE - Black Box Server 26

Still referring to Fig. 1, in one embodiment of the present invention, the black box server 26 performs the functions of installing and/or upgrading a new black box 30 in a user's computing device 14. As will be explained in more detail below, the black box 30 performs encryption and decryption functions for the user's computing device 14. As will also be explained in more detail below, the black box 30 is intended to be secure and protected from attack. Such security and protection is provided, at least in part, by upgrading the black box 30 to a new version as necessary by way of the black box server 26, as will be explained in more detail below.

As with the authoring tool 18, the content server 22, and the license server 24, the black box server 26 is implemented on an appropriate computer, processor, or other computing machine by way of appropriate software. The structure and operation of such machine and such software should be apparent based on the disclosure herein and therefore do not require any detailed discussion in the present disclosure. Moreover, in one embodiment of the present invention the license server 24, the authoring tool 18, and/or the content server 22 may reside on a single computer.

processor, or other computing machine together with the black box server 26, each in a separate work space. Note, though, that for security purposes. it may be wise to have the black box server 26 on a separate machine.

ARCHITECTURE - User's Computing Device 14

Referring now to Fig. 4, in one embodiment of the present invention, the user's computing device 14 is a personal computer or the like, having elements including a keyboard, a mouse, a screen, a processor, RAM, ROM, a hard drive, a floppy drive, a CD player, and/or the like. However, the user's computing device 14 may also be a dedicated viewing device such as a television or monitor, a dedicated audio device such as a stereo or other music player, a dedicated printer, or the like, among other things, all without departing from the spirit and scope of the present invention.

The content owner for a piece of digital content 12 must trust that the user's computing device 14 will abide by the rules specified by such content owner, i.e. that the digital content 12 will not be rendered unless the user obtains a license 16 that permits the rendering in the manner sought. Preferably, then, the user's computing device 14 must provide a trusted component or mechanism 32 that can satisfy to the content owner that such computing device 14 will not render the digital content 12 except according to the license rules embodied in the license 16 associated with the digital content 12 and obtained by the user.

Here, the trusted mechanism 32 is a Digital Rights Management (DRM) system 32 that is enabled when a user requests that a piece of digital content 12 be rendered, that determines whether the user has a license 16 to render the digital content 12 in the manner sought, that effectuates obtaining such a license 16 if necessary, that determines whether the user has the right to play the digital content 12 according to the license 16, and that decrypts the digital content 12 for rendering purposes if in fact the user has such right according to such license 16. The contents and function of the DRM system 32 on the user's computing device 14 and in connection with the architecture 10 are described below.

DRM SYSTEM 32

The DRM system 32 performs four main functions with the architecture 10 disclosed herein: (1) content acquisition, (2) license acquisition, (3) content rendering, and (4) black box 30 installation / update. Preferably, any of the functions can be performed at any time, although it is recognized that some of the functions already require that digital content 12 be acquired.

DRM SYSTEM 32 - Content Acquisition

Acquisition of digital content 12 by a user and/or the user's computing device 14 is typically a relatively straight-forward matter and generally involves placing a file having encrypted digital content 12 on the user's computing device 14. Of course, to work with the architecture 10 and the DRM system 32 disclosed herein, it is necessary that the encrypted digital content 12 be in a form that is amenable to such architecture 10 and DRM system 32, such as the digital package 12p as will be described below.

As should be understood, the digital content 12 may be obtained in any manner from a content server 22, either directly or indirectly, without departing from the spirit and scope of the present invention. For example, such digital content 12 may be downloaded from a network such as the Internet, located on an obtained optical or magnetic disk or the like, received as part of an E-mail message or the like, or downloaded from an electronic bulletin board or the like.

Such digital content 12, once obtained, is preferably stored in a manner such that the obtained digital content 12 is accessible by a rendering application 34 (to be described below) running on the computing device 14, and by the DRM system 32. For example, the digital content 12 may be placed as a file on a hard drive (not shown) of the user's computing device 14, or on a network server (not shown) accessible to the computing device 14. In the case where the digital content 12 is obtained on an optical or magnetic disk or the like, it may only be necessary that such disk be present in an appropriate drive (not shown) coupled to the user's computing device 14.

In the present invention, it is not envisioned that any special tools are

necessary to acquire digital content 12, either from the content server 22 as a direct distribution source or from some intermediary as an indirect distribution source. That is, it is preferable that digital content 12 be as easily acquired as any other data file. However, the DRM system 32 and/or the rendering application 34 may include an interface (not shown) designed to assist the user in obtaining digital content 12 . For example, the interface may include a web browser especially designed to search for digital content 12, links to pre-defined Internet web sites that are known to be sources of digital content 12, and the like.

DRM SYSTEM 32 - Content Rendering, Part 1

Referring now to Fig. 5A, in one embodiment of the present invention, assuming the encrypted digital content 12 has been distributed to and received by a user and placed by the user on the computing device 14 in the form of a stored file, the user will attempt to render the digital content 12 by executing some variation on a render command (step 501). For example, such render command may be embodied as a request to 'play' or 'open' the digital content 12. In some computing environments, such as for example the "MICROSOFT WINDOWS" operating system, distributed by MICROSOFT Corporation of Redmond, Washington, such play or open command may be as simple as 'clicking' on an icon representative of the digital content 12. Of course, other embodiments of such render command may be employed without departing from the spirit and scope of the present invention. In general, such render command may be considered to be executed whenever a user directs that a file having digital content 12 be opened, run, executed, and/or the like.

Importantly, and in addition, such render command may be embodied as a request to copy the digital content 12 to another form, such as to a printed form, a visual form, an audio form, etc. As should be understood, the same digital content 12 may be rendered in one form, such as on a computer screen, and then in another form, such as a printed document. In the present invention, each type of rendering is performed only if the user has the right to do so, as will be explained below.

In one embodiment of the present invention, the digital content 12 is in

the form of a digital file having a file name ending with an extension, and the computing device 14 can determine based on such extension to start a particular kind of rendering application 34. For example, if the file name extension indicates that the digital content 12 is a text file, the rendering application 34 is some form of word processor such as the "MICROSOFT WORD", distributed by MICROSOFT Corporation of Redmond, Washington. Likewise, if the file name extension indicates that the digital content 12 is an audio, video, and/or multimedia file, the rendering application 34 is some form of multimedia player, such as "MICROSOFT MEDIA PLAYER", also distributed by MICROSOFT Corporation of Redmond, Washington.

Of course, other methods of determining a rendering application may be employed without departing from the spirit and scope of the present invention. As but one example, the digital content 12 may contain meta-data in an un-encrypted form (i.e., the aforementioned header information), where the meta-data includes information on the type of rendering application 34 necessary to render such digital content 12.

Preferably, such rendering application 34 examines the digital content 12 associated with the file name and determines whether such digital content 12 is encrypted in a rights-protected form (steps 503, 505). If not protected, the digital content 12 may be rendered without further ado (step 507). If protected, the rendering application 34 determines from the encrypted digital content 12 that the DRM system 32 is necessary to play such digital content 12. Accordingly, such rendering application 34 directs the user's computing device 14 to run the DRM system 32 thereon (step 509). Such rendering application 34 then calls such DRM system 32 to decrypt the digital content 12 (step 511). As will be discussed in more detail below, the DRM system 32 in fact decrypts the digital content 12 only if the user has a valid license 16 for such digital content 12 and the right to play the digital content 12 according to the license rules in the valid license 16. Preferably, once the DRM system 32 has been called by the rendering application 34, such DRM system 32 assumes control from the rendering application 34, at least for purposes of determining whether

the user has a right to play such digital content 12 (step 513).

DRM System 32 Components

In one embodiment of the present invention, and referring again to Fig. 4, the DRM system 32 includes a license evaluator 36, the black box 30, a license store 38, and a state store 40.

DRM System 32 Components - License Evaluator 36

The license evaluator 36 locates one or more licenses 16 that correspond to the requested digital content 12, determines whether such licenses 16 are valid, reviews the license rules in such valid licenses 16, and determines based on the reviewed license rules whether the requesting user has the right to render the requested digital content 12 in the manner sought, among other things. As should be understood, the license evaluator 36 is a trusted component in the DRM system 32. In the present disclosure, to be 'trusted' means that the license server 24 (or any other trusting element) is satisfied that the trusted element will carry out the wishes of the owner of the digital content 12 according to the rights description in the license 16, and that a user cannot easily alter such trusted element for any purpose, nefarious or otherwise.

The license evaluator 36 has to be trusted in order to ensure that such license evaluator 36 will in fact evaluate a license 16 properly, and to ensure that such license evaluator 36 has not been adulterated or otherwise modified by a user for the purpose of bypassing actual evaluation of a license 16. Accordingly, the license evaluator 36 is run in a protected or shrouded environment such that the user is denied access to such license evaluator 36. Other protective measures may of course be employed in connection with the license evaluator 36 without departing from the spirit and scope of the present invention.

DRM System 32 Components - Black Box 30

Primarily, and as was discussed above, the black box 30 performs encryption and decryption functions in the DRM system 32. In particular, the black box 30 works in conjunction with the license evaluator 36 to decrypt and encrypt certain information as part of the license evaluation function. In addition, once the

license evaluator 36 determines that a user does in fact have the right to render the requested digital content 12 in the manner sought, the black box 30 is provided with a decryption key (KD) for such digital content 12, and performs the function of decrypting such digital content 12 based on such decryption key (KD).

The black box 30 is also a trusted component in the DRM system 32. In particular, the license server 24 must trust that the black box 30 will perform the decryption function only in accordance with the license rules in the license 16, and also trust that such black box 30 will not operate should it become adulterated or otherwise modified by a user for the nefarious purpose of bypassing actual evaluation of a license 16. Accordingly, the black box 30 is also run in a protected or shrouded environment such that the user is denied access to such black box 30. Again, other protective measures may be employed in connection with the black box 30 without departing from the spirit and scope of the present invention. Preferably, and like the content server 22 and license server 24, the black box 30 in the DRM system 32 has a unique public / private key pair (PU-BB, PR-BB) that is employed as part of the process of evaluating the license 16 and obtaining a decryption key (KD) for decrypting the digital content 12, as will be described in more detail below.

DRM System 32 Components - License Store 38

The license store 38 stores licenses 16 received by the DRM system 32 for corresponding digital content 12. The license store 38 itself need not be trusted since the license store 38 merely stores licenses 16, each of which already has trust components built thereinto, as will be described below. In one embodiment of the present invention, the license store 38 is merely a sub-directory of a drive such as a hard disk drive or a network drive. However, the license store 38 may be embodied in any other form without departing from the spirit and scope of the present invention, so long as such license store 38 performs the function of storing licenses 16 in a location relatively convenient to the DRM system 32.

DRM System 32 Components - State Store 40

The state store 40 performs the function of maintaining state

information corresponding to licenses 16 presently or formerly in the license store 38. Such state information is created by the DRM system 32 and stored in the state store 40 as necessary. For example, if a particular license 16 only allows a pre-determined number of renderings of a piece of corresponding digital content 12, the state store 40 maintains state information on how many renderings have in fact taken place in connection with such license 16. The state store 40 continues to maintain state information on licenses 16 that are no longer in the license store 38 to avoid the situation where it would otherwise be advantageous to delete a license 16 from the license store 38 and then obtain an identical license 16 in an attempt to delete the corresponding state information from the state store 40.

The state store 40 also has to be trusted in order to ensure that the information stored therein is not reset to a state more favorable to a user. Accordingly, the state store 40 is likewise run in a protected or shrouded environment such that the user is denied access to such state store 40. Once again, other protective measures may of course be employed in connection with the state store 40 without departing from the spirit and scope of the present invention. For example, the state store 40 may be stored by the DRM system 32 on the computing device 14 in an encrypted form.

DRM SYSTEM 32 - Content Rendering, Part 2

Referring again to Fig. 5A, and again discussing content rendering in one embodiment of the present invention, once the DRM system 32 has assumed control from the calling rendering application 34, such DRM system 32 then begins the process of determining whether the user has a right to render the requested digital content 12 in the manner sought. In particular, the DRM system 32 either locates a valid, enabling license 16 in the license store (steps 515, 517) or attempts to acquire a valid, enabling license 16 from the license server 24 (i.e. performs the license acquisition function as discussed below and as shown in Fig. 7).

As a first step, and referring now to Fig. 6, the license evaluator 36 of such DRM system 32 checks the license store 38 for the presence of one or more received licenses 16 that correspond to the digital content 12 (step 601). Typically, the

license 16 is in the form of a digital file, as will be discussed below, although it will be recognized that the license 16 may also be in other forms without departing from the spirit and scope of the present invention. Typically, the user will receive the digital content 12 without such license 16, although it will likewise be recognized that the digital content 12 may be received with a corresponding license 16 without departing from the spirit and scope of the present invention.

As was discussed above in connection with Fig. 3, each piece of digital content 12 is in a package 12p with a content ID (or package ID) identifying such digital content 12 (or package 12p), and a key ID identifying the decryption key (KD) that will decrypt the encrypted digital content 12. Preferably, the content ID (or package ID) and the key ID are in an un-encrypted form. Accordingly, and in particular, based on the content ID of the digital content 12, the license evaluator 36 looks for any license 16 in the license store 38 that contains an identification of applicability to such content ID. Note that multiple such licenses 16 may be found, especially if the owner of the digital content 12 has specified several different kinds of licenses 16 for such digital content 12, and the user has obtained multiple ones of such licenses 16. If in fact the license evaluator 36 does not find in the license store 38 any license 16 corresponding to the requested digital content 12, the DRM system 32 may then perform the function of license acquisition (step 519 of Fig. 5), to be described below.

Assume now that the DRM system 32 has been requested to render a piece of digital content 12, and one or more licenses 16 corresponding thereto are present in the license store 38. In one embodiment of the present invention, then, the license evaluator 36 of the DRM system 32 proceeds to determine for each such license 16 whether such license 16 itself is valid (steps 603 and 605 of Fig. 6). Preferably, and in particular, each license 16 includes a digital signature 26 based on the content 28 of the license 16. As should be understood, the digital signature 26 will not match the license 16 if the content 28 has been adulterated or otherwise modified. Thus, the license evaluator 36 can determine based on the digital signature 26 whether the

content 28 is in the form that it was received from the license server 24 (i.e., is valid). If no valid license 16 is found in the license store 38, the DRM system 32 may then perform the license acquisition function described below to obtain such a valid license 16.

Assuming that one or more valid licenses 16 are found, for each valid license 16, the license evaluator 36 of the DRM system 32 next determines whether such valid license 16 gives the user the right to render the corresponding digital content 12 in the manner desired (i.e., is enabling) (steps 607 and 609). In particular, the license evaluator 36 determines whether the requesting user has the right to play the requested digital content 12 based on the rights description in each license 16 and based on what the user is attempting to do with the digital content 12. For example, such rights description may allow the user to render the digital content 12 into a sound, but not into a decrypted digital copy.

As should be understood, the rights description in each license 16 specifies whether the user has rights to play the digital content 12 based on any of several factors, including who the user is, where the user is located, what type of computing device 14 the user is using, what rendering application 34 is calling the DRM system 32, the date, the time, etc. In addition, the rights description may limit the license 16 to a pre-determined number of plays, or pre-determined play time, for example. In such case, the DRM system 32 must refer to any state information with regard to the license 16, (i.e., how many times the digital content 12 has been rendered, the total amount of time the digital content 12 has been rendered, etc.), where such state information is stored in the state store 40 of the DRM system 32 on the user's computing device 14.

Accordingly, the license evaluator 36 of the DRM system 32 reviews the rights description of each valid license 16 to determine whether such valid license 16 confers the rights sought to the user. In doing so, the license evaluator 36 may have to refer to other data local to the user's computing device 14 to perform a determination of whether the user has the rights sought. As seen in Fig. 4, such data

may include an identification 42 of the user's computing device (machine) 14 and particular aspects thereof, an identification 44 of the user and particular aspects thereof, an identification of the rendering application 34 and particular aspects thereof, a system clock 46, and the like. If no valid license 16 is found that provides the user with the right to render the digital content 12 in the manner sought. the DRM system 32 may then perform the license acquisition function described below to obtain such a license 16, if in fact such a license 16 is obtainable.

Of course, in some instances the user cannot obtain the right to render the digital content 12 in the manner requested, because the content owner of such digital content 12 has in effect directed that such right not be granted. For example, the content owner of such digital content 12 may have directed that no license 16 be granted to allow a user to print a text document, or to copy a multimedia presentation into an un-encrypted form. In one embodiment of the present invention, the digital content 12 includes data on what rights are available upon purchase of a license 16, and types of licenses 16 available. However, it will be recognized that the content owner of a piece of digital content 12 may at any time change the rights currently available for such digital content 12 by changing the licenses 16 available for such digital content 12.

DRM SYSTEM 32 - License Acquisition

Referring now to Fig. 7, if in fact the license evaluator 36 does not find in the license store 38 any valid, enabling license 16 corresponding to the requested digital content 12, the DRM system 32 may then perform the function of license acquisition. As shown in Fig. 3, each piece of digital content 12 is packaged with information in an un-encrypted form regarding how to obtain a license 16 for rendering such digital content 12 (i.e., license acquisition information).

In one embodiment of the present invention. such license acquisition information may include (among other things) types of licenses 16 available, and one or more Internet web sites or other site information at which one or more appropriate license servers 24 may be accessed, where each such license server 24 is in fact capable

of issuing a license 16 corresponding to the digital content 12. Of course, the license 16 may be obtained in other manners without departing from the spirit and scope of the present invention. For example, the license 16 may be obtained from a license server 24 at an electronic bulletin board, or even in person or via regular mail in the form of a file on a magnetic or optical disk or the like.

Assuming that the location for obtaining a license 16 is in fact a license server 24 on a network, the license evaluator 36 then establishes a network connection to such license server 24 based on the web site or other site information, and then sends a request for a license 16 from such connected license server 24 (steps 701, 703). In particular, once the DRM system 32 has contacted the license server 24, such DRM system 32 transmits appropriate license request information 36 to such license server 24. In one embodiment of the present invention, such license 16 request information 36 may include:

- the public key of the black box 30 of the DRM system 32 (PU-BB);
- the version number of the black box 30 of the DRM system 32;
- a certificate with a digital signature from a certifying authority certifying the black box 30 (where the certificate may in fact include the aforementioned public key and version number of the black box 30);
- the content ID (or package ID) that identifies the digital content 12 (or package 12p);
- the key ID that identifies the decryption key (KD) for decrypting the digital content 12;
- the type of license 16 requested (if in fact multiple types are available);
- the type of rendering application 34 that requested rendering of the digital content 12;

and/or the like, among other things. Of course, greater or lessor amounts of license 16 request information 36 may be transmitted to the license server 24 by the DRM system

32 without departing from the spirit and scope of the present invention. For example, information on the type of rendering application 34 may not be necessary, while additional information about the user and/or the user's computing device 14 may be necessary.

Once the license server 24 has received the license 16 request information 36 from the DRM system 32, the license server 24 may then perform several checks for trust / authentication and for other purposes. In one embodiment of the present invention, such license server 24 checks the certificate with the digital signature of the certifying authority to determine whether such has been adulterated or otherwise modified (steps 705, 707). If so, the license server 24 refuses to grant any license 16 based on the request information 36. The license server 24 may also maintain a list of known 'bad' users and/or user's computing devices 14, and may refuse to grant any license 16 based on a request from any such bad user and/or bad user's computing device 14 on the list. Such 'bad' list may be compiled in any appropriate manner without departing from the spirit and scope of the present invention.

Based on the received request and the information associated therewith, and particularly based on the content ID (or package ID) in the license request information, the license server 24 can interrogate the content-key database 20 (Fig. 1) and locate a record corresponding to the digital content 12 (or package 12p) that is the basis of the request. As was discussed above, such record contains the decryption key (KD), key ID, and content ID for such digital content 12. In addition, such record may contain license data regarding the types of licenses 16 to be issued for the digital content 12 and the terms and conditions for each type of license 16. Alternatively, such record may include a pointer, link, or reference to a location having such additional information.

As mentioned above, multiple types of licenses 16 may be available. For example, for a relatively small license fee, a license 16 allowing a limited number of renderings may be available. For a relatively greater license fee, a license 16

allowing unlimited renderings until an expiration date may be available. For a still greater license fee, a license 16 allowing unlimited renderings without any expiration date may be available. Practically any type of license 16 having any kind of license terms may be devised and issued by the license server 24 without departing from the spirit and scope of the present invention.

In one embodiment of the present invention, the request for a license 16 is accomplished with the aid of a web page or the like as transmitted from the license server 24 to the user's computing device 14. Preferably, such web page includes information on all types of licenses 16 available from the license server 24 for the digital content 12 that is the basis of the license 16 request.

In one embodiment of the present invention, prior to issuing a license 16, the license server 24 checks the version number of the black box 30 to determine whether such black box 30 is relatively current (steps 709, 711). As should be understood, the black box 30 is intended to be secure and protected from attacks from a user with nefarious purposes (i.e., to improperly render digital content 12 without a license 16, or outside the terms of a corresponding license 16). However, it is to be recognized that no system and no software device is in fact totally secure from such an attack.

As should also be understood, if the black box 30 is relatively current, i.e., has been obtained or updated relatively recently, it is less likely that such black box 30 has been successfully attacked by such a nefarious user. Preferably, and as a matter of trust, if the license server 24 receives a license request with request information 36 including a black box 30 version number that is not relatively current, such license server 24 refuses to issue the requested license 16 until the corresponding black box 30 is upgraded to a current version, as will be described below. Put simply, the license server 24 will not trust such black box 30 unless such black box 30 is relatively current.

In the context of the black box 30 of the present invention, the term 'current' or 'relatively current' may have any appropriate meaning without departing

from the spirit and scope of the present invention. consistent with the function of providing trust in the black box 30 based on the age or use thereof. For example, 'current' may be defined according to age (i.e., less than one month old). As an alternative example, 'current' may be defined based on a number of times that the black box 30 has decrypted digital content 12 (i.e., less than 200 instances of decryption). Moreover, 'current' may be based on policy as set by each license server 24, where one license server 24 may define 'current' differently from another license server 24, and a license server 24 may further define 'current' differently depending on the digital content 12 for which a license 16 is requested. or depending on the type of license 16 requested, among other things.

Assuming that the license server 24 is satisfied from the version number of a black box 30 or other indicia thereof that such black box 30 is current, the license server 24 then proceeds to negotiate terms and conditions for the license 16 with the user (step 713). Alternatively, the license server 24 negotiates the license 16 with the user, then satisfies itself from the version number of the black box 30 that such black box 30 is current (i.e., performs step 713, then step 711). Of course, the amount of negotiation varies depending on the type of license 16 to be issued, and other factors. For example, if the license server 24 is merely issuing a paid-up unlimited use license 16, very little need be negotiated. On the other hand, if the license 16 is to be based on such items as varying values, sliding scales, break points, and other details, such items and details may need to be worked out between the license server 24 and the user before the license 16 can be issued.

As should be understood, depending on the circumstances, the license negotiation may require that the user provide further information to the license server 24 (for example, information on the user, the user's computing device 14, etc.). Importantly, the license negotiation may also require that the user and the license server 24 determine a mutually acceptable payment instrument (a credit account, a debit account, a mailed check, etc.) and/or payment method (paid-up immediately, spread over a period of time, etc.), among other things.

Once all the terms of the license 16 have been negotiated and agreed to by both the license server 24 and user (step 715). a digital license 16 is generated by the license server 24 (step 719), where such generated license 16 is based at least in part on the license request, the black box 30 public key (PU-BB), and the decryption key (KD) for the digital content 12 that is the basis of the request as obtained from the content-key database 20. In one embodiment of the present invention, and as seen in Fig. 8, the generated license 16 includes:

- the content ID of the digital content 12 to which the license 16 applies;
- a Digital Rights License (DRL) 48 (i.e., the rights description or actual terms and conditions of the license 16 written in a predetermined form that the license evaluator 36 can interrogate), perhaps encrypted with the decryption key (KD) (i.e., KD (DRL));
- the decryption key (KD) for the digital content 12 encrypted with the black box 30 public key (PU-BB) as receive in the license request (i.e.,(PU-BB (KD));
- a digital signature from the license server 24 (without any attached certificate) based on (KD (DRL)) and (PU-BB (KD)) and encrypted with the license server 24 private key (i.e., (S (PR-LS))); and
- the certificate that the license server 24 obtained previously from the content server 22, such certificate indicating that the license server 24 has the authority from the content server 22 to issue the license 16 (i.e., (CERT (PU-LS) S (PR-CS))).

As should be understood, the aforementioned elements and perhaps others are packaged into a digital file or some other appropriate form. As should also be understood, if the DRL 48 or (PU-BB (KD)) in the license 16 should become adulterated or otherwise modified, the digital signature (S (PR-LS)) in the license 16 will not match and therefore will not validate such license 16. For this reason, the DRL 48 need not necessarily be in an encrypted form (i.e., (KD(DRL)) as mentioned

above), although such encrypted form may in some instances be desirable and therefore may be employed without departing from the spirit and scope of the present invention.

Once the digital license 16 has been prepared. such license 16 is then issued to the requestor (i.e., the DRM system 32 on the user's computing device 14) (step 719 of Fig. 7). Preferably, the license 16 is transmitted over the same path through which the request therefor was made (i.e., the Internet or another network), although another path may be employed without departing from the spirit and scope of the present invention. Upon receipt, the requesting DRM system 32 preferably automatically places the received digital license 16 in the license store 38 (step 721).

It is to be understood that a user's computing device 14 may on occasion malfunction, and licenses 16 stored in the license store 38 of the DRM system 32 on such user's computing device 14 may become irretrievably lost. Accordingly, it is preferable that the license server 24 maintain a database 50 of issued licenses 16 (Fig. 1), and that such license server 24 provide a user with a copy or re-issue (hereinafter 're-issue') of an issued license 16 if the user is in fact entitled to such re-issue. In the aforementioned case where licenses 16 are irretrievably lost, it is also likely the case that state information stored in the state store 40 and corresponding to such licenses 16 is also lost. Such lost state information should be taken into account when re-issuing a license 16. For example. a fixed number of renderings license 16 might legitimately be re-issued in a pro-rated form after a relatively short period of time, and not re-issued at all after a relatively longer period of time.

DRM SYSTEM 32 - Installation/Upgrade of Black Box 30

As was discussed above, as part of the function of acquiring a license 16, the license server 24 may deny a request for a license 16 from a user if the user's computing device 14 has a DRM system 32 with a black box 30 that is not relatively current, i.e., has a relatively old version number. In such case. it is preferable that the black box 30 of such DRM system 32 be upgraded so that the license acquisition function can then proceed. Of course, the black box 30 may be upgraded at other times

without departing from the spirit and scope of the present invention.

Preferably, as part of the process of installing the DRM system 32 on a user's computing device 14, a non-unique 'lite' version of a black box 30 is provided. Such 'lite' black box 30 is then upgraded to a unique regular version prior to rendering a piece of digital content 12. As should be understood, if each black box 30 in each DRM system 32 is unique, a security breach into one black box 30 cannot easily be replicated with any other black box 30.

Referring now to Fig. 9, the DRM system 32 obtains the unique black box 30 by requesting same from a black box server 26 or the like (as was discussed above and as shown in Fig. 1) (step 901). Typically, such request is made by way of the Internet, although other means of access may be employed without departing from the spirit and scope of the present invention. For example, the connection to a black box server 26 may be a direct connection, either locally or remotely. An upgrade from one unique non-lite black box 30 to another unique non-lite black box 30 may also be requested by the DRM system 32 at any time, such as for example a time when a license server 24 deems the black box 30 not current, as was discussed above.

Thereafter, the black box server 26 generates a new unique black box 30 (step 903). As seen in Fig. 3, each new black box 30 is provided with a version number and a certificate with a digital signature from a certifying authority. As was discussed above in connection with the license acquisition function, the version number of the black box 30 indicates the relative age and/or use thereof. The certificate with the digital signature from the certifying authority, also discussed above in connection with the license acquisition function, is a proffer or vouching mechanism from the certifying authority that a license server 24 should trust the black box 30. Of course, the license server 24 must trust the certifying authority to issue such a certificate for a black box 30 that is in fact trustworthy. It may be the case, in fact, that the license server 24 does not trust a particular certifying authority, and refuses to honor any certificate issued by such certifying authority. Trust may not occur, for example, if a particular certifying authority is found to be engaging in a pattern of

improperly issuing certificates.

Preferably, and as was discussed above, the black box server 26 includes a new unique public / private key pair (PU-BB, PR-BB) with the newly generated unique black box 30 (step 903 of Fig. 9). Preferably, the private key for the black box 30 (PR-BB) is accessible only to such black box 30, and is hidden from and inaccessible by the remainder of the world, including the computing device 14 having the DRM system 32 with such black box 30, and the user thereof.

Most any hiding scheme may be employed without departing from the spirit and scope of the present invention, so long as such hiding scheme in fact performs the function of hiding the private key (PR-BB) from the world. As but one example, the private key (PR-BB) may be split into several sub-components, and each sub-component may be encrypted uniquely and stored in a different location. In such a situation, it is preferable that such sub-components are never assembled in full to produce the entire private key (PR-BB).

In one embodiment of the present invention, such private key (PR-BB) is encrypted according to code-based encryption techniques. In particular, in such embodiment, the actual software code of the black box 30 (or other software code) is employed as encrypting key(s). Accordingly, if the code of the black box 30 (or the other software code) becomes adulterated or otherwise modified, for example by a user with nefarious purposes, such private key (PR-BB) cannot be decrypted.

Although each new black box 30 is delivered with a new public / private key pair (PU-BB, PR-BB), such new black box 30 is also preferably given access to old public / private key pairs from old black boxes 30 previously delivered to the DRM system 32 on the user's computing device 14 (step 905). Accordingly, the upgraded black box 30 can still employ the old key pairs to access older digital content 12 and older corresponding licenses 16 that were generated according to such old key pairs, as will be discussed in more detail below.

Preferably, the upgraded black box 30 delivered by the black box server 26 is tightly tied to or associated with the user's computing device 14. Accordingly,

the upgraded black box 30 cannot be operably transferred among multiple computing devices 14 for nefarious purposes or otherwise. In one embodiment of the present invention, as part of the request for the black box 30 (step 901) the DRM system 32 provides hardware information unique to such DRM system 32 and/or unique to the user's computing device 14 to the black box server 26, and the black box server 26 generates a black box 30 for the DRM system 32 based in part on such provided hardware information. Such generated upgraded black box 30 is then delivered to and installed in the DRM system 32 on the user's computing device 14 (steps 907, 909). If the upgraded black box 30 is then somehow transferred to another computing device 14, the transferred black box 30 recognizes that it is not intended for such other computing device 14, and does not allow any requested rendering to proceed on such other computing device 14.

Once the new black box 30 is installed in the DRM system 32, such DRM system 32 can proceed with a license acquisition function or with any other function.

DRM SYSTEM 32 - Content Rendering, Part 3

Referring now to Fig. 5B, and assuming, now, that the license evaluator 36 has found at least one valid license 16 and that at least one of such valid licenses 16 provides the user with the rights necessary to render the corresponding digital content 12 in the manner sought (i.e., is enabling), the license evaluator 36 then selects one of such licenses 16 for further use (step 519). Specifically, to render the requested digital content 12, the license evaluator 36 and the black box 30 in combination obtain the decryption key (KD) from such license 16, and the black box 30 employs such decryption key (KD) to decrypt the digital content 12. In one embodiment of the present invention, and as was discussed above, the decryption key (KD) as obtained from the license 16 is encrypted with the black box 30 public key (PU-BB(KD)), and the black box 30 decrypts such encrypted decryption key with its private key (PR-BB) to produce the decryption key (KD) (steps 521, 523). However, other methods of obtaining the decryption key (KD) for the digital content 12 may be employed without

departing from the spirit and scope of the present invention.

Once the black box 30 has the decryption key (KD) for the digital content 12 and permission from the license evaluator 36 to render the digital content 12, control may be returned to the rendering application 34 (steps 525, 527). In one embodiment of the present invention, the rendering application 34 then calls the DRM system 32 / black box 30 and directs at least a portion of the encrypted digital content 12 to the black box 30 for decryption according to the decryption key (KD) (step 529). The black box 30 decrypts the digital content 12 based upon the decryption key (KD) for the digital content 12, and then the black box 30 returns the decrypted digital content 12 to the rendering application 34 for actual rendering (steps 533, 535). The rendering application 34 may either send a portion of the encrypted digital content 12 or the entire digital content 12 to the black box 30 for decryption based on the decryption key (KD) for such digital content 12 without departing from the spirit and scope of the present invention.

Preferably, when the rendering application 34 sends digital content 12 to the black box 30 for decryption, the black box 30 and/or the DRM system 32 authenticates such rendering application 34 to ensure that it is in fact the same rendering application 34 that initially requested the DRM system 32 to run (step 531). Otherwise, the potential exists that rendering approval may be obtained improperly by basing the rendering request on one type of rendering application 34 and in fact rendering with another type of rendering application 34. Assuming the authentication is successful and the digital content 12 is decrypted by the black box 30, the rendering application 34 may then render the decrypted digital content 12 (steps 533, 535).

Sequence of Key Transactions

Referring now to Fig. 10, in one embodiment of the present invention, a sequence of key transactions is performed to obtain the decryption key (KD) and evaluate a license 16 for a requested piece of digital content 12 (i.e., to perform steps 515-523 of Figs. 5A and 5B). Mainly, in such sequence, the DRM system 32 obtains the decryption key (KD) from the license 16, uses information obtained from the

license 16 and the digital content 12 to authenticate or ensure the validity of both, and then determines whether the license 16 in fact provides the right to render the digital content 12 in the manner sought. If so, the digital content 12 may be rendered.

Bearing in mind that each license 16 for the digital content 12, as seen in Fig. 8, includes:

- the content ID of the digital content 12 to which the license 16 applies;
- the Digital Rights License (DRL) 48, perhaps encrypted with the decryption key (KD) (i.e., KD (DRL));
- the decryption key (KD) for the digital content 12 encrypted with the black box 30 public key (PU-BB) (i.e.,(PU-BB (KD));
- the digital signature from the license server 24 based on (KD (DRL)) and (PU-BB (KD)) and encrypted with the license server 24 private key (i.e., (S (PR-LS))); and
- the certificate that the license server 24 obtained previously from the content server 22 (i.e., (CERT (PU-LS) S (PR-CS))),

and also bearing in mind that the package 12p having the digital content 12, as seen in Fig. 3, includes:

- the content ID of such digital content 12;
- the digital content 12 encrypted by KD (i.e., (KD(CONTENT)));
- a license acquisition script that is not encrypted; and
- the key KD encrypting the content server 22 public key (PU-CS), signed by the content server 22 private key (PR-CS) (i.e., (KD (PU-CS) S (PR-CS))),

in one embodiment of the present invention, the specific sequence of key transactions that are performed with regard to a specific one of the licenses 16 for the digital content 12 is as follows:

1. Based on (PU-BB (KD)) from the license 16, the black box 30 of the DRM system 32 on the user's computing device 14 applies its private key (PR-

BB) to obtain (KD) (step 1001). (PR-BB (PU-BB (KD)) = (KD)). Note, importantly, that the black box 30 could then proceed to employ KD to decrypt the digital content 12 without any further ado. However, and also importantly, the license server 24 trusts the black box 30 not to do so. Such trust was established at the time such license server 24 issued the license 16 based on the certificate from the certifying authority vouching for the trustworthiness of such black box 30. Accordingly, despite the black box 30 obtaining the decryption key (KD) as an initial step rather than a final step, the DRM system 32 continues to perform all license 16 validation and evaluation functions, as described below.

2. Based on (KD (PU-CS) S (PR-CS)) from the digital content 12, the black box 30 applies the newly obtained decryption key (KD) to obtain (PU-CS) (step 1003). (KD (KD (PU-CS)) = (PU-CS)). Additionally, the black box 30 can apply (PU-CS) as against the signature (S (PR-CS)) to satisfy itself that such signature and such digital content 12 / package 12p is valid (step 1005). If not valid, the process is halted and access to the digital content 12 is denied.

3. Based on (CERT (PU-LS) S (PR-CS)) from the license 16, the black box 30 applies the newly obtained content server 22 public key (PU-CS) to satisfy itself that the certificate is valid (step 1007), signifying that the license server 24 that issued the license 16 had the authority from the content server 22 to do so, and then examines the certificate contents to obtain (PU-LS) (step 1009). If not valid, the process is halted and access to the digital content 12 based on the license 16 is denied.

4. Based on (S (PR-LS)) from the license 16, the black box 30 applies the newly obtained license server 24 public key (PU-LS) to satisfy itself that the license 16 is valid (step 1011). If not valid, the process is halted and access to the digital content 12 based on the license 16 is denied.

5. Assuming all validation steps are successful, and that the DRL 48 in the license 16 is in fact encrypted with the decryption key (KD), the license evaluator 36 then applies the already-obtained decryption key (KD) to (KD(DRL)) as obtained from the license 16 to obtain the license terms from the license 16 (i.e., the

DRL 48) (step 1013). Of course, if the DRL 48 in the license 16 is not in fact encrypted with the decryption key (KD), step 1013 may be omitted. The license evaluator 36 then evaluates / interrogates the DRL 48 and determines whether the user's computing device 14 has the right based on the DRL 48 in the license 16 to render the corresponding digital content 12 in the manner sought (i.e., whether the DRL 48 is enabling) (step 1015). If the license evaluator 36 determines that such right does not exist, the process is halted and access to the digital content 12 based on the license 16 is denied.

6. Finally, assuming evaluation of the license 16 results in a positive determination that the user's computing device 14 has the right based on the DRL 48 terms to render the corresponding digital content 12 in the manner sought, the license evaluator 36 informs the black box 30 that such black box 30 can render the corresponding digital content 12 according to the decryption key (KD). The black box 30 thereafter applies the decryption key (KD) to decrypt the digital content 12 from the package 12p (i.e., (KD(KD(CONTENT)) = (CONTENT)) (step 1017).

It is important to note that the above-specified series of steps represents an alternating or 'ping-ponging' between the license 16 and the digital content 12. Such ping-ponging ensures that the digital content 12 is tightly bound to the license 16, in that the validation and evaluation process can only occur if both the digital content 12 and license 16 are present in a properly issued and valid form. In addition, since the same decryption key (KD) is needed to get the content server 22 public key (PU-CS) from the license 16 and the digital content 12 from the package 12p in a decrypted form (and perhaps the license terms (DRL 48) from the license 16 in a decrypted form), such items are also tightly bound. Signature validation also ensures that the digital content 12 and the license 16 are in the same form as issued from the content server 22 and the license server 24, respectively. Accordingly, it is difficult if not impossible to decrypt the digital content 12 by bypassing the license server 24, and also difficult if not impossible to alter and then decrypt the digital content 12 or the license 16.

In one embodiment of the present invention, signature verification, and especially signature verification of the license 16, is alternately performed as follows. Rather than having a signature encrypted by the private key of the license server 16 (PR-LS), as is seen in Fig. 8, each license 16 has a signature encrypted by a private root key (PR-R) (not shown), where the black box 30 of each DRM system 32 includes a public root key (PU-R) (also not shown) corresponding to the private root key (PR-R). The private root key (PR-R) is known only to a root entity, and a license server 24 can only issue licenses 16 if such license server 24 has arranged with the root entity to issue licenses 16.

In particular, in such embodiment:

1. the license server 24 provides its public key (PU-LS) to the root entity;
2. the root entity returns the license server public key (PU-LS) to such license server 24 encrypted with the private root key (PR-R) (i.e., (CERT (PU-LS) S (PR-R))); and
3. the license server 24 then issues a license 16 with a signature encrypted with the license server private key (S (PR-LS)), and also attaches to the license the certificate from the root entity (CERT (PU-LS) S (PR-R)).

For a DRM system 18 to validate such issued license 16, then, the DRM system 18:

1. applies the public root key (PU-R) to the attached certificate (CERT (PU-LS) S (PR-R)) to obtain the license server public key (PU-LS); and
2. applies the obtained license server public key (PU-LS) to the signature of the license 16 (S (PR-LS).

Importantly, it should be recognized that just as the root entity gave the license server 24 permission to issue licenses 16 by providing the certificate (CERT (PU-LS) S (PR-R)) to such license server 24, such license server 24 can provide a

similar certificate to a second license server 24 (i.e.. (CERT (PU-LS2) S (PR-LS1)), thereby allowing the second license server to also issue licenses 16. As should now be evident, a license 16 issued by the second license server would include a first certificate (CERT (PU-LS1) S (PR-R)) and a second certificate (CERT (PU-LS2) S (PR-LS1)). Likewise, such license 16 is validated by following the chain through the first and second certificates. Of course, additional links in the chain may be added and traversed.

One advantage of the aforementioned signature verification process is that the root entity may periodically change the private root key (PR-R), thereby likewise periodically requiring each license server 24 to obtain a new certificate (CERT (PU-LS) S (PR-R)). Importantly, as a requirement for obtaining such new certificate, each license server may be required to upgrade itself. As with the black box 30, if a license server 24 is relatively current, i.e., has been upgraded relatively recently, it is less likely that license server 24 has been successfully attacked. Accordingly, as a matter of trust, each license server 24 is preferably required to be upgraded periodically via an appropriate upgrade trigger mechanism such as the signature verification process. Of course, other upgrade mechanisms may be employed without departing from the spirit and scope of the present invention.

Of course, if the private root key (PR-R) is changed. then the public root key (PU-R) in each DRM system 18 must also be changed. Such change may for example take place during a normal black box 30 upgrade, or in fact may require that a black box 30 upgrade take place. Although a changed public root key (PU-R) may potentially interfere with signature validation for an older license 16 issued based on an older private root key (PR-R), such interference may be minimized by requiring that an upgraded black box 30 remember all old public root keys (PU-R). Alternatively, such interference may be minimized by requiring signature verification for a license 16 only once, for example the first time such license 16 is evaluated by the license evaluator 36 of a DRM system 18. In such case. state information on whether signature verification has taken place should be compiled. and such state information

should be stored in the state store 40 of the DRM system 18.

Digital Rights License 48

In the present invention, the license evaluator 36 evaluates a Digital Rights License (DRL) 48 as the rights description or terms of a license 16 to determine if such DRL 48 allows rendering of a corresponding piece of digital content 12 in the manner sought. In one embodiment of the present invention, the DRL 48 may be written by a licensor (i.e., the content owner) in any DRL language.

As should be understood, there are a multitude of ways to specify a DRL 48. Accordingly, a high degree of flexibility must be allowed for in any DRL language. However, it is impractical to specify all aspects of a DRL 48 in a particular license language, and it is highly unlikely that the author of such a language can appreciate all possible licensing aspects that a particular digital licensor may desire. Moreover, a highly sophisticated license language may be unnecessary and even a hindrance for a licensor providing a relatively simple DRL 48. Nevertheless, a licensor should not be unnecessarily restricted in how to specify a DRL 48. At the same time, the license evaluator 36 should always be able to get answers from a DRL 48 regarding a number of specific license questions.

In the present invention, and referring now to Fig. 11, a DRL 48 can be specified in any license language, but includes a language identifier or tag 54. The license evaluator 36 evaluating the license 16, then, performs the preliminary step of reviewing the language tag 54 to identify such language, and then selects an appropriate license language engine 52 for accessing the license 16 in such identified language. As should be understood, such license language engine 52 must be present and accessible to the license evaluator 36. If not present, the language tag 54 and/or the DRL 48 preferably includes a location 56 (typically a web site) for obtaining such language engine 52.

Typically, the language engine 52 is in the form of an executable file or set of files that reside in a memory of the user's computing device 14, such as a hard drive. The language engine 52 assists the license evaluator 36 to directly interrogate

the DRL 48, the license evaluator 36 interrogates the DRL 48 indirectly via the language engine 48 acting as an intermediary, or the like. When executed, the language engine 52 runs in a work space in a memory of the user's computing device 14, such as RAM. However, any other form of language engine 52 may be employed without departing from the spirit and scope of the present invention.

Preferably, any language engine 52 and any DRL language supports at least a number of specific license questions that the license evaluator 36 expects to be answered by any DRL 48, as will be discussed below. Accordingly, the license evaluator 36 is not tied to any particular DRL language; a DRL 48 may be written in any appropriate DRL language; and a DRL 48 specified in a new license language can be employed by an existing license evaluator 36 by having such license evaluator 36 obtain a corresponding new language engine 52.

DRL Languages

Two examples of DRL languages, as embodied in respective DRLs 48, are provided below. The first, 'simple' DRL 48 is written in a DRL language that specifies license attributes, while the second 'script' DRL 48 is written in a DRL language that can perform functions according to the script specified in the DRL 48. While written in a DRL language, the meaning of each line of code should be apparent based on the linguistics thereof and/or on the attribute description chart that follows:

Simple DRL 48:

```
<LICENSE>
    <DATA>
        <NAME>Beastie Boy's Play</NAME>
        <ID>39384</ID>
        <DESCRIPTION>Play the song 3 times</DESCRIPTION>
        <TERMS></TERMS>
        <VALIDITY>
            <NOTBEFORE>19980102 23:20:14Z</NOTBEFORE>
            <NOTAFTER>19980102 23:20:14Z</NOTAFTER>
        </VALIDITY>
        <ISSUEDDATE>19980102 23:20:14Z</ISSUEDDATE>
        <LICENSORSITE>http://www.foo.com</LICENSORSITE>
```

```
<CONTENT>
        <NAME>Beastie Boy's</NAME>
        <ID>392</ID>
        <KEYID>39292</KEYID>
        <TYPE>MS Encrypted ASF 2.0</TTYPE>
</CONTENT>
<OWNER>
        <ID>939KDKD393KD</ID>
        <NAME>Universal</NAME>
        <PUBLICKEY></PUBLICKEY>
</OWNER>
<LICENSEE>
        <NAME>Arnold</NAME>
        <ID>939KDKD393KD</ID>
        <PUBLICKEY></PUBLICKEY>
</LICENSEE>
<PRINCIPAL TYPE='AND'>
        <PRINCIPAL TYPE='OR'>
                <PRINCIPAL>
                        <TYPE>x86Computer</TYPE>
                        <ID>3939292939d9e939</ID>
                        <NAME>Personal Computer</NAME>
                        <AUTHTYPE>Intel Authenticated Boot PC
                        SHA-1 DSA512</AUTHTYPE>
                        <AUTHDATA>29293939</AUTHDATA>
                </PRINCIPAL>
                <PRINCIPAL>
                        <TYPE>Application</TYPE>
                        <ID>2939495939292</ID>
                        <NAME>Window's Media Player</NAME>
                        <AUTHTYPE>Authenticode SHA-
                        1</AUTHTYPE>
                        <AUTHDATA>93939</AUTHDATA>
                </PRINCIPAL>
        </PRINCIPAL>
        <PRINCIPAL>
                <TYPE>Person</TYPE>
                <ID>39299482010</ID>
                <NAME>Arnold Blinn</NAME>
                <AUTHTYPE>Authenticate user</AUTHTYPE>
                <AUTHDATA>\\redmond\arnoldb</AUTHDATA>
        </PRINCIPAL>
</PRINCIPAL>
```

```
                <DRLTYPE>Simple</DRLTYPE>  [the language tag 54]
                <DRLDATA>
                        <START>19980102 23:20:14Z</START>
                        <END>19980102 23:20:14Z</END>
                        <COUNT>3</COUNT>
                        <ACTION>PLAY</ACTION>
                </DRLDATA>
                <ENABLINGBITS>aaaabbbbccccdddd</ENABLINGBITS>
        </DATA>
        <SIGNATURE>
        <SIGNERNAME>Universal</SIGNERNAME>
                <SIGNERID>9382ABK3939DKD</SIGNERID>
                <HASHALGORITHMID>MD5</HASHALGORITHMID>
                <SIGNALGORITHMID>RSA 128</SIGNALGORITHMID>
                <SIGNATURE>xxxyyyxxxyyyxxxyyy</SIGNATURE>
                <SIGNERPUBLICKEY></SIGNERPUBLICKEY>
                <CONTENTSIGNEDSIGNERPUBLICKEY></CONTENTSIGNEDSI
                GNERPUBLICKEY>
        </SIGNATURE>
</LICENSE>
```

Script DRL 48:

```
<LICENSE>
        <DATA>
                <NAME>Beastie Boy's Play</NAME>
                <ID>39384</ID>
                <DESCRIPTION>Play the song unlimited</DESCRIPTION>
                <TERMS></TERMS>
                <VALIDITY>
                        <NOTBEFORE>19980102 23:20:14Z</NOTBEFORE>
                        <NOTAFTER>19980102 23:20:14Z</NOTAFTER>
                </VALIDITY>
                <ISSUEDDATE>19980102 23:20:14Z</ISSUEDDATE>
                <LICENSORSITE>http://www.foo.com</LICENSORSITE>
                <CONTENT>
                        <NAME>Beastie Boy's</NAME
                        <ID>392</ID>
                        <KEYID>39292</KEYID>
                        <TYPE>MS Encrypted ASF 2.0</TTYPE>
                </CONTENT>
                <OWNER>
                        <ID>939KDKD393KD</ID>
```

```
                <NAME>Universal</NAME>
                <PUBLICKEY></PUBLICKEY>
            </OWNER>
            <LICENSEE>
                <NAME>Arnold</NAME>
                <ID>939KDKD393KD</ID>
                <PUBLICKEY></PUBLICKEY>
            </LICENSEE>
            <DRLTYPE>Script</DRLTYPE>    [the language tag 54]
            <DRLDATA>
                function on_enable(action, args) as boolean
                    result = False
                    if action = "PLAY" then
                        result = True
                    end if
                    on_action = False
                end function
                . . .
            </DRLDATA>
        </DATA>
        <SIGNATURE>
            <SIGNERNAME>Universal</SIGNERNAME>
            <SIGNERID>9382</SIGNERID>
            <SIGNERPUBLICKEY></SIGNERPUBLICKEY>
            <HASHID>MD5</HASHID>
            <SIGNID>RSA 128</SIGNID>
            <SIGNATURE>xxxyyyxxxyyyxxxyyy</SIGNATURE>
            <CONTENTSIGNEDSIGNERPUBLICKEY></CONTENTSIGNEDSI
            GNERPUBLICKEY>
        </SIGNATURE>
    </LICENSE>
```

In the two DRLs 48 specified above, the attributes listed have the following descriptions and data types:

Attribute	Description	Data Type
Id	ID of the license	GUID
Name	Name of the license	String
Content Id	ID of the content	GUID
Content Key Id	ID for the encryption key of the content	GUID
Content Name	Name of the content	String
Content Type	Type of the content	String

Owner Id	ID of the owner of the content	GUID
Owner Name	Name of the owner of the content	String
Owner Public Key	Public key for owner of content. This is a base-64 encoded public key for the owner of the content.	String
Licensee Id	Id of the person getting license. It may be null.	GUID
Licensee Name	Name of the person getting license. It may be null.	String
Licensee Public Key	Public key of the licensee. This is the base-64 encoded public key of the licensee. It may be null.	String
Description	Simple human readable description of the license	String
Terms	Legal terms of the license. This may be a pointer to a web page containing legal prose.	String
Validity Not After	Validity period of license expiration	Date
Validity Not Before	Validity period of license start	Date
Issued Date	Date the license was issued	Date
DRL Type	Type of the DRL. Example include "SIMPLE" or "SCRIPT"	String
DRL Data	Data specific to the DRL	String
Enabling Bits	These are the bits that enable access to the actual content. The interpretation of these bits is up to the application, but typically this will be the private key for decryption of the content. This data will be base-64 encoded. Note that these bits are encrypted using the public key of the individual machine.	String
Signer Id	ID of person signing license	GUID
Signer Name	Name of person signing license	String
Signer Public Key	Public key for person signing license. This is the base-64 encode public key for the signer.	String
Content Signed Signer Public Key	Public key for person signing the license that has been signed by the content server private key. The public key to verify this signature will be encrypted in the content. This is base-64 encoded.	String

Hash Alg Id	Algorithm used to generate hash. This is a string, such as "MD5".	String
Signature Alg Id	Algorithm used to generate signature. This is a string, such as "RSA 128".	String
Signature	Signature of the data. This is base-64 encoded data.	String

Methods

As was discussed above, it is preferable that any language engine 52 and any DRL language support at least a number of specific license questions that the digital license evaluator 36 expects to be answered by any DRL 48. Recognizing such supported questions may include any questions without departing from the spirit and scope of the present invention, and consistent with the terminology employed in the two DRL 48 examples above, in one embodiment of the present invention, such supported questions or 'methods' include 'access methods', 'DRL methods', and 'enabling use methods', as follows:

Access Methods

Access methods are used to query a DRL 48 for top-level attributes.

VARIANT QueryAttribute (BSTR key)

Valid keys include License.Name, License.Id, Content.Name, Content.Id, Content.Type, Owner.Name, Owner.Id, Owner.PublicKey, Licensee.Name, Licensee.Id, Licensee.PublicKey, Description, and Terms, each returning a BSTR variant; and Issued, Validity.Start and Validity.End, each returning a Date Variant.

DRL Methods

The implementation of the following DRL methods varies from DRL 48 to DRL 48. Many of the DRL methods contain a variant parameter labeled 'data' which is intended for communicating more advanced information with a DRL 48. It

is present largely for future expandability.

Boolean IsActivated(Variant data)

This method returns a Boolean indicating whether the DRL 48 / license 16 is activated. An example of an activated license 16 is a limited operation license 16 that upon first play is active for only 48 hours.

Activate(Variant data)

This method is used to activate a license 16. Once a license 16 is activated, it cannot be deactivated.

Variant QueryDRL(Variant data)

This method is used to communicate with a more advanced DRL 48. It is largely about future expandability of the DRL 48 feature set.

Variant GetExpires(BSTR action, Variant data)

This method returns the expiration date of a license 16 with regard to the passed-in action. If the return value is NULL, the license 16 is assumed to never expire or does not yet have an expiration date because it hasn't been activated. or the like.

Variant GetCount(BSTR action, Variant data)

This method returns the number of operations of the passed-in action that are left. If NULL is returned, the operation can be performed an unlimited number of times.

Boolean IsEnabled(BSTR action, Variant data)

This method indicates whether the license 16 supports the requested action at the present time.

Boolean IsSunk(BSTR action, Variant data)

This method indicates whether the license 16 has been paid for. A license 16 that is paid for up front would return TRUE, while a license 16 that is not paid for up front, such as a license 16 that collects payments as it is used, would return FALSE.

Enabling Use Methods.

These methods are employed to enable a license 16 for use in decrypting content.

Boolean Validate (BSTR key)

This method is used to validate a license 16. The passed-in key is the black box 30 public key (PU-BB) encrypted by the decryption key (KD) for the corresponding digital content 12 (i.e.,(KD(PU-BB))) for use in validation of the signature of the license 16. A return value of TRUE indicates that the license 16 is valid. A return value of FALSE indicates invalid.

int OpenLicense 16(BSTR action, BSTR key. Variant data)

This method is used to get ready to access the decrypted enabling bits. The passed-in key is (KD(PU-BB)) as described above. A return value of 0 indicates success. Other return values can be defined.

BSTR GetDecryptedEnablingBits (BSTR action. Variant data)

Variant GetDecryptedEnablingBitsAsBinary (BSTR action. Variant Data)

These methods are used to access the enabling bits in decrypted form. If this is not successful for any of a number of reasons, a null string or null variant is returned.

void CloseLicense 16 (BSTR action, Variant data)

This method is used to unlock access to the enabling bits for performing the passed-in action. If this is not successful for any of a number of reasons. a null string is returned.

Heuristics

As was discussed above, if multiple licenses 16 are present for the same piece of digital content 12, one of the licenses 16 must be chosen for further use. Using the above methods, the following heuristics could be implemented to make such choice. In particular, to perform an action (say "PLAY") on a piece of digital content 12, the following steps could be performed:

1. Get all licenses 16 that apply to the particular piece of digital content 12.
2. Eliminate each license 16 that does not enable the action by calling the IsEnabled function on such license 16.
3. Eliminate each license 16 that is not active by calling IsActivated on such license 16.
4. Eliminate each license 16 that is not paid for up front by calling IsSunk on such license 16.
5. If any license 16 is left, use it. Use an unlimited-number-of-plays license 16 before using a limited-number-of-plays license 16, especially if the unlimited-number-of-plays license 16 has an expiration date. At any time, the user should be allowed to select a specific license 16 that has already been acquired, even if the choice is not cost-effective. Accordingly, the user can select a license 16 based on criteria that are perhaps not apparent to the DRM system 32.
6. If there are no licenses 16 left, return status so indicating. The user would then be given the option of:

   using a license 16 that is not paid for up front, if available;

   activating a license 16, if available; and/or

   performing license acquisition from a license server 24.

CONCLUSION

The programming necessary to effectuate the processes performed in connection with the present invention is relatively straight-forward and should be

apparent to the relevant programming public. Accordingly, such programming is not attached hereto. Any particular programming, then, may be employed to effectuate the present invention without departing from the spirit and scope thereof.

In the foregoing description, it can be seen that the present invention comprises a new and useful enforcement architecture 10 that allows the controlled rendering or playing of arbitrary forms of digital content 12, where such control is flexible and definable by the content owner of such digital content 12. Also, the present invention comprises a new useful controlled rendering environment that renders digital content 12 only as specified by the content owner, even though the digital content 12 is to be rendered on a computing device 14 which is not under the control of the content owner. Further, the present invention comprises a trusted component that enforces the rights of the content owner on such computing device 14 in connection with a piece of digital content 12, even against attempts by the user of such computing device 14 to access such digital content 12 in ways not permitted by the content owner.

It should be appreciated that changes could be made to the embodiments described above without departing from the inventive concepts thereof. It should be understood, therefore, that this invention is not limited to the particular embodiments disclosed, but it is intended to cover modifications within the spirit and scope of the present invention as defined by the appended claims.

CLAIMS

1. A method for a device to interdependently validate:

a digital content package having a piece of digital content in an encrypted form; and

a corresponding digital license for rendering the digital content,

the method comprising:

deriving a first key from a source available to the device;

obtaining a first digital signature from the digital content package;

applying the first key to the first digital signature to validate the first digital signature and the digital content package;

deriving a second key based on the first digital signature;

obtaining a second digital signature from the license; and

applying the second key to the second digital signature to validate the second digital signature and the license.

2. The method of claim 1 wherein deriving the first key comprises:

obtaining a first encrypted key from the license;

applying a key available to the device to the first encrypted key to decrypt the first encrypted key;

obtaining a second encrypted key from the digital content; and

applying the decrypted first encrypted key to the second encrypted key to produce the first key.

3. The method of claim 2 wherein the encrypted digital content is decryptable according to a decryption key (KD), and wherein the first encrypted key is the decryption key (KD) encrypted with the device public key (PU-D) (i.e.,(PU-D (KD))).

4. The method of claim 2 wherein the device has a public key (PU-D) and a private key (PR-D), and wherein the key available to the device is (PR-D).

5. The method of claim 2 wherein the encrypted digital content is decryptable according to a decryption key (KD), wherein the digital content package is provided by a content provider having a public key (PU-C) and a private key (PR-C), and wherein the second encrypted key is the content provider public key (PU-C) encrypted with the decryption key (KD) (i.e., KD (PU-C)).

6. The method of claim 2 wherein the second encrypted key is the basis for the first digital signature.

7. The method of claim 1 wherein deriving the second key comprises:

obtaining a signed certificate from the license. the signed certificate having contents therein; and

applying the first key to the signature of the signed certificate to produce the contents of the certificate and also to validate the signature.

8. The method of claim 7 wherein the digital license is provided by a license provider having a public key (PU-L) and a private key (PR-L), and wherein the contents of the certificate is (PU-L).

9. The method of claim 8 wherein the digital content package is provided by a content provider having a public key (PU-C) and a private key (PR-C), and wherein the signed certificate is a certificate containing the license provider public key (PU-L) and signed by the content provider private key (PR-C) (i.e., (CERT (PU-L) S (PR-C))).

10. The method of claim 8 wherein the digital content package is provided by a content provider authorized by a root source to provide the package, wherein the root source has a public key (PU-R) and a private key (PR-R) and wherein the signed certificate is a certificate containing the license provider public key (PU-L) and signed by the root source private key (PR-R) (i.e., (CERT (PU-L) S (PR-R))).

11. The method of claim 1 wherein the digital content package is provided by a content provider having a public key (PU-C) and a private key (PR-C), and wherein the first key is (PU-C).

12. The method of claim 11 wherein the encrypted digital content is decryptable according to a decryption key (KD), and wherein the first digital signature is based on the content provider public key (PU-C) encrypted with the decryption key (KD) and

is signed by the content provider private key (PR-C) (i.e.. (KD (PU-C) S (PR-C))).

13. The method of claim 12 wherein deriving (PU-C) comprises:

deriving (KD) from a source available to the device;

applying (KD) to (KD (PU-C) S (PR-C)) to produce (PU-C).

14. The method of claim 13 wherein the device has a public key (PU-D) and a private key (PR-D), wherein the license has the decryption key (KD) encrypted with the device public key (PU-D) (i.e.,(PU-D (KD))). and wherein deriving (KD) comprises:

obtaining (PU-D (KD)) from the license;

applying (PR-D) to (PU-D (KD)) to produce (KD).

15 The method of claim 14 wherein the license has a license rights description specifying terms and conditions that must be satisfied before the digital content may be rendered, the license rights description being encrypted with the decryption key (KD) (i.e., (KD (DRL))), the method further comprising applying (KD) to (KD(DRL)) to obtain the license terms and conditions.

16. The method of claim 14 wherein the license has a license rights description specifying terms and conditions that must be satisfied before the digital content may be rendered, the method further comprising:

evaluating the license terms and conditions to determine whether the digital content is permitted to be rendered in the manner sought;

if so, applying (KD) to the encrypted digital content to decrypt such encrypted digital content; and

rendering the decrypted digital content.

17. The method of claim 11 wherein the encrypted digital content package is provided by a content provider authorized by a root source to provide the package, wherein the root source has a public key (PU-R) and a private key (PR-R) and wherein the first digital signature is a signed certificate containing the content provider public key (PU-C) and signed by the root source private key (PR-R) (i.e., (CERT (PU-C) S (PR-R))).

18. The method of claim 1 wherein the digital license is provided by a license provider having a public key (PU-L) and a private key (PR-L). and wherein the second key is (PU-L).

19. The method of claim 18 wherein the second digital signature is a digital signature encrypted with the license provider private key (i.e., (S (PR-L))).

20. The method of claim 19 wherein the digital content package is provided by a content provider having a public key (PU-C) and a private key (PR-C), wherein the

license has a certificate containing the license provider public key (PU-L) and signed by the content provider private key (PR-C) (i.e., (CERT (PU-L) S (PR-C))), and wherein deriving (PU-L) comprises:

deriving (PU-C) from a source available to the device;

obtaining (CERT (PU-L) S (PR-C)) from the license; and

applying (PU-C) to (CERT (PU-L) S (PR-C)) to validate (CERT (PU-L) S (PR-C)), to produce (PU-L) and also to validate the content provider.

21. The method of claim 20 wherein the encrypted digital content is decryptable according to a decryption key (KD), wherein the first digital signature is based on the content provider public key (PU-C) encrypted with the decryption key (KD) and is signed by the content provider private key (PR-C) (i.e., (KD (PU-C) S (PR-C))), and wherein deriving (PU-C) comprises:

deriving (KD) from a source available to the device;

applying (KD) to (KD (PU-C) S (PR-C)) to produce (PU-C).

22. The method of claim 21 wherein the device has a public key (PU-D) and a private key (PR-D), wherein the license has the decryption key (KD) encrypted with the device public key (PU-D) (i.e.,(PU-D (KD))), and wherein deriving (KD) comprises:

obtaining (PU-D (KD)) from the license;

applying (PR-D) to (PU-D (KD)) to produce (KD).

23. The method of claim 22 wherein the license has a license rights description specifying terms and conditions that must be satisfied before the digital content may be rendered, the license rights description being encrypted with the decryption key (KD) (i.e., (KD (DRL))), the method further comprising applying (KD) to (KD(DRL)) to obtain the license terms and conditions.

24. The method of claim 22 wherein the license has a license rights description specifying terms and conditions that must be satisfied before the digital content may be rendered, the method further comprising:

evaluating the license terms and conditions to determine whether the digital content is permitted to be rendered in the manner sought;

if so, applying (KD) to the encrypted digital content to decrypt such encrypted digital content; and

rendering the decrypted digital content.

25. A method for a device to interdependently validate a piece of digital content and a corresponding digital license for rendering the digital content. the digital content being encrypted, the encrypted digital content being decryptable according to a decryption key (KD) and being packaged in a digital content package. the digital content package being provided by a content provider having a public key (PU-C) and a private key (PR-C), the digital license being provided by a license provider having

a public key (PU-L) and a private key (PR-L). the device having a public key (PU-D) and a private key (PR-D), the digital content package comprising:

the encrypted digital content; and

the content provider public key (PU-C) encrypted with the decryption key (KD) and signed by the content provider private key (PR-C) (i.e., (KD (PU-C) S (PR-C)));

the digital license comprising:

the decryption key (KD) encrypted with the device public key (PU-D) (i.e.,(PU-D (KD)));

a digital signature from the license provider (without any attached certificate) based on (KD (DRL)) and (PU-D (KD)) and encrypted with the license provider private key (i.e., (S (PR-L))); and

a certificate containing the license provider public key (PU-L) and signed by the content provider private key (PR-C) (i.e., (CERT (PU-L) S (PR-C)));

the method comprising:

obtaining (PU-D (KD)) from the license;

applying (PR-D) to (PU-D (KD)) to produce (KD);

obtaining (KD (PU-C) S (PR-C)) from the digital content package;

applying (KD) to (KD (PU-C) S (PR-C)) to produce (PU-C);

applying (PU-C) to (S (PR-C)) to validate (KD (PU-C) S (PR-C)), thereby validating the digital content package;

obtaining (CERT (PU-L) S (PR-C)) from the license;

applying (PU-C) to (CERT (PU-L) S (PR-C)) to validate (CERT (PU-L) S (PR-C)), thereby validating the content provider, and also to obtain (PU-L);

obtaining (S (PR-L)) from the license; and

applying (PU-L) to (S (PR-L)). thereby validating the license.

26. The method of claim 25 wherein the digital content package further comprises a content / package ID identifying one of the digital content and the digital content package, and wherein the license further comprises the content / package ID of the corresponding digital content / digital content package, the method further comprising ensuring that the content / package ID of the license in fact corresponds to the content / package ID of the digital content / digital content package.

27. The method of claim 25 wherein the license further comprises a license rights description (DRL) specifying terms and conditions that must be satisfied before the digital content may be rendered, the method further comprising;

evaluating the license terms and conditions to determine whether the digital content is permitted to be rendered in the manner sought;

if so, applying (KD) to the encrypted digital content to decrypt such encrypted digital content; and

rendering the decrypted digital content.

28. The method of claim 27 wherein the license rights description is encrypted with the decryption key (KD) (i.e., (KD (DRL))), the method further comprising applying (KD) to (KD (DRL)) to obtain the license terms and conditions.

29. A computer-readable medium having computer-executable instructions for performing a method for a device to interdependently validate:

a digital content package having a piece of digital content in an encrypted form; and

a corresponding digital license for rendering the digital content, the method comprising:

deriving a first key from a source available to the device;

obtaining a first digital signature from the digital content package;

applying the first key to the first digital signature to validate the first digital signature and the digital content package;

deriving a second key based on the first digital signature;

obtaining a second digital signature from the license; and

applying the second key to the second digital signature to validate the second digital signature and the license.

30. The method of claim 28 wherein deriving the first key comprises:

obtaining a first encrypted key from the license;

applying a key available to the device to the first encrypted key to

decrypt the first encrypted key;

obtaining a second encrypted key from the digital content; and

applying the decrypted first encrypted key to the second encrypted key to produce the first key.

31. The method of claim 30 wherein the encrypted digital content is decryptable according to a decryption key (KD), and wherein the first encrypted key is the decryption key (KD) encrypted with the device public key (PU-D) (i.e.,(PU-D (KD))).

32. The method of claim 30 wherein the device has a public key (PU-D) and a private key (PR-D), and wherein the key available to the device is (PR-D).

33. The method of claim 30 wherein the encrypted digital content is decryptable according to a decryption key (KD), wherein the digital content package is provided by a content provider having a public key (PU-C) and a private key (PR-C), and wherein the second encrypted key is the content provider public key (PU-C) encrypted with the decryption key (KD) (i.e., KD (PU-C)).

34. The method of claim 30 wherein the second encrypted key is the basis for the first digital signature.

35. The method of claim 29 wherein deriving the second key comprises:

obtaining a signed certificate from the license. the signed certificate having contents therein; and

applying the first key to the signature of the signed certificate to produce the contents of the certificate and also to validate the signature.

36. The method of claim 35 wherein the digital license is provided by a license provider having a public key (PU-L) and a private key (PR-L), and wherein the contents of the certificate is (PU-L).

37. The method of claim 36 wherein the digital content package is provided by a content provider having a public key (PU-C) and a private key (PR-C), and wherein the signed certificate is a certificate containing the license provider public key (PU-L) and signed by the content provider private key (PR-C) (i.e.. (CERT (PU-L) S (PR-C))).

38. The method of claim 36 wherein the digital content package is provided by a content provider authorized by a root source to provide the package. wherein the root source has a public key (PU-R) and a private key (PR-R) and wherein the signed certificate is a certificate containing the license provider public key (PU-L) and signed by the root source private key (PR-R) (i.e.. (CERT (PU-L) S (PR-R))).

39. The method of claim 29 wherein the digital content package is provided by a content provider having a public key (PU-C) and a private key (PR-C). and wherein

the first key is (PU-C).

40. The method of claim 39 wherein the encrypted digital content is decryptable according to a decryption key (KD), and wherein the first digital signature is based on the content provider public key (PU-C) encrypted with the decryption key (KD) and is signed by the content provider private key (PR-C) (i.e., (KD (PU-C) S (PR-C))).

41. The method of claim 40 wherein deriving (PU-C) comprises:

deriving (KD) from a source available to the device;

applying (KD) to (KD (PU-C) S (PR-C)) to produce (PU-C).

42. The method of claim 41 wherein the device has a public key (PU-D) and a private key (PR-D), wherein the license has the decryption key (KD) encrypted with the device public key (PU-D) (i.e.,(PU-D (KD))), and wherein deriving (KD) comprises:

obtaining (PU-D (KD)) from the license;

applying (PR-D) to (PU-D (KD)) to produce (KD).

43. The method of claim 42 wherein the license has a license rights description specifying terms and conditions that must be satisfied before the digital content may be rendered, the license rights description being encrypted with the decryption key (KD) (i.e., (KD (DRL))), the method further comprising applying (KD) to (KD(DRL))

to obtain the license terms and conditions.

44. The method of claim 42 wherein the license has a license rights description specifying terms and conditions that must be satisfied before the digital content may be rendered, the method further comprising:

evaluating the license terms and conditions to determine whether the digital content is permitted to be rendered in the manner sought;

if so, applying (KD) to the encrypted digital content to decrypt such encrypted digital content; and

rendering the decrypted digital content.

45. The method of claim 39 wherein the encrypted digital content package is provided by a content provider authorized by a root source to provide the package, wherein the root source has a public key (PU-R) and a private key (PR-R) and wherein the first digital signature is a signed certificate containing the content provider public key (PU-C) and signed by the root source private key (PR-R) (i.e., (CERT (PU-C) S (PR-R))).

46. The method of claim 29 wherein the digital license is provided by a license provider having a public key (PU-L) and a private key (PR-L), and wherein the second key is (PU-L).

47. The method of claim 46 wherein the second digital signature is a digital signature encrypted with the license provider private key (i.e., (S (PR-L))).

48. The method of claim 47 wherein the digital content package is provided by a content provider having a public key (PU-C) and a private key (PR-C), wherein the license has a certificate containing the license provider public key (PU-L) and signed by the content provider private key (PR-C) (i.e., (CERT (PU-L) S (PR-C))), and wherein deriving (PU-L) comprises:

deriving (PU-C) from a source available to the device;

obtaining (CERT (PU-L) S (PR-C)) from the license; and

applying (PU-C) to (CERT (PU-L) S (PR-C)) to validate (CERT (PU-L) S (PR-C)), to produce (PU-L) and also to validate the content provider.

49. The method of claim 48 wherein the encrypted digital content is decryptable according to a decryption key (KD), wherein the first digital signature is based on the content provider public key (PU-C) encrypted with the decryption key (KD) and is signed by the content provider private key (PR-C) (i.e., (KD (PU-C) S (PR-C))), and wherein deriving (PU-C) comprises:

deriving (KD) from a source available to the device:

applying (KD) to (KD (PU-C) S (PR-C)) to produce (PU-C).

50. The method of claim 49 wherein the device has a public key (PU-D) and a

private key (PR-D), wherein the license has the decryption key (KD) encrypted with the device public key (PU-D) (i.e.,(PU-D (KD))), and wherein deriving (KD) comprises:

obtaining (PU-D (KD)) from the license;

applying (PR-D) to (PU-D (KD)) to produce (KD).

51. The method of claim 50 wherein the license has a license rights description specifying terms and conditions that must be satisfied before the digital content may be rendered, the license rights description being encrypted with the decryption key (KD) (i.e., (KD (DRL))), the method further comprising applying (KD) to (KD(DRL)) to obtain the license terms and conditions.

52. The method of claim 50 wherein the license has a license rights description specifying terms and conditions that must be satisfied before the digital content may be rendered, the method further comprising:

evaluating the license terms and conditions to determine whether the digital content is permitted to be rendered in the manner sought;

if so, applying (KD) to the encrypted digital content to decrypt such encrypted digital content; and

rendering the decrypted digital content.

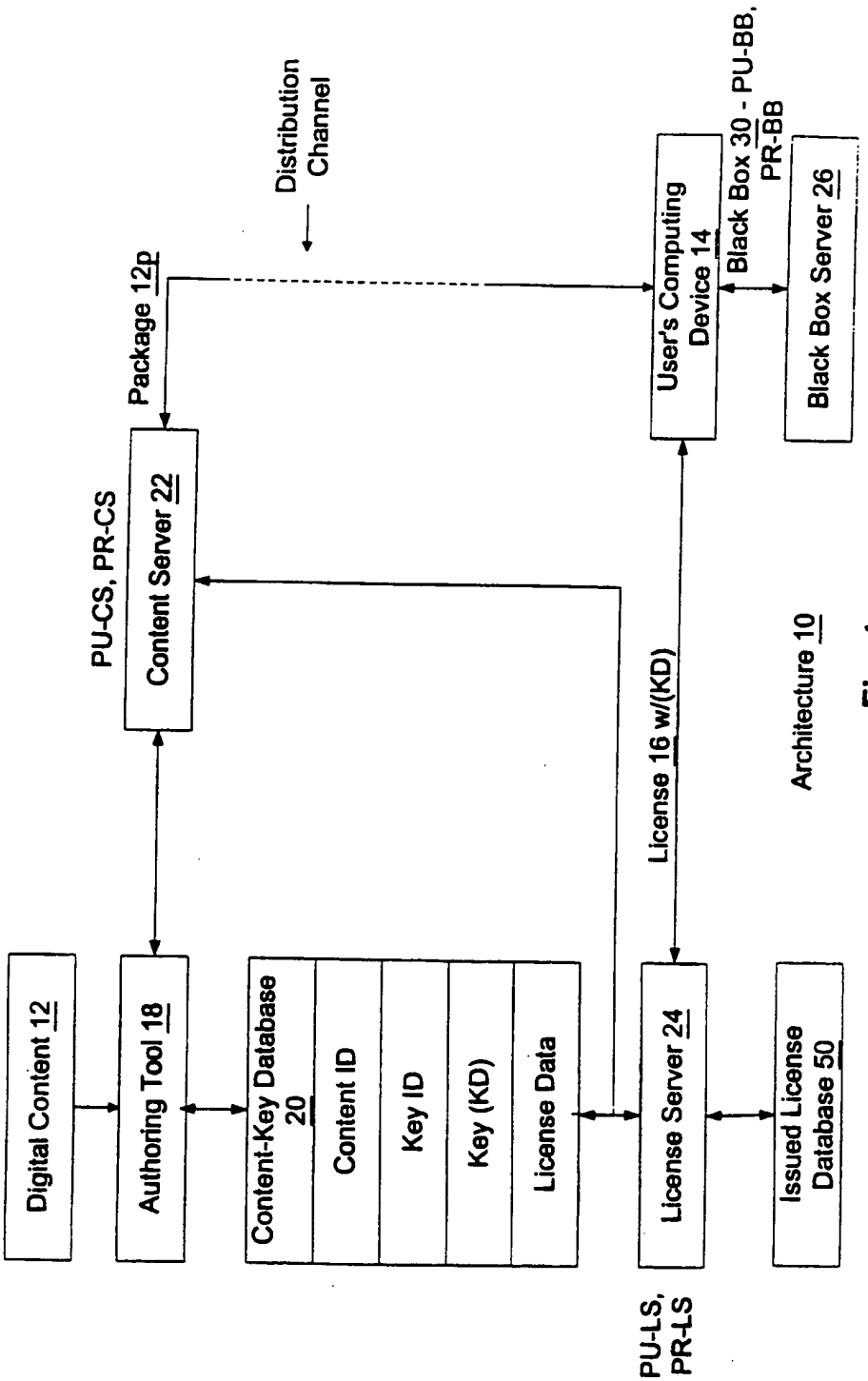

Distribution Channel
Package 12p
User's Computing Device 14
Black Box 30 - PU-BB, PR-BB
Black Box Server 26
PU-CS, PR-CS
Content Server 22
License 16 w/(KD)
Digital Content 12
Authoring Tool 18
Content-Key Database 20
Content ID
Key ID
Key (KD)
License Data
License Server 24
Issued License Database 50
PU-LS, PR-LS
Architecture 10

Fig. 1

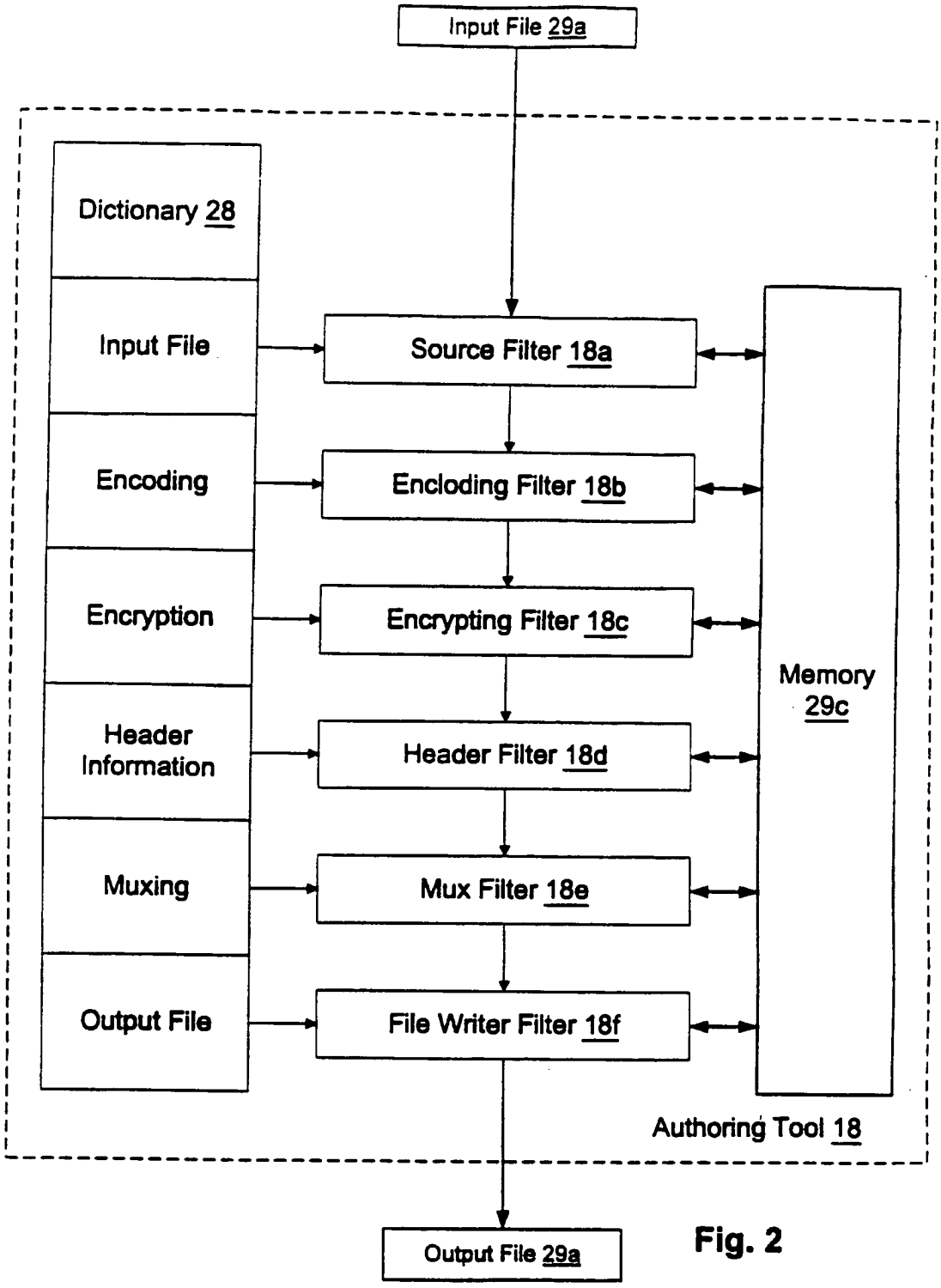

Input File 29a
Dictionary 28
Input File
Encoding
Encryption
Header Information
Muxing
Output File
Source Filter 18a
Encloding Filter 18b
Encrypting Filter 18c
Header Filter 18d
Mux Filter 18e
File Writer Filter 18f
Memory 29c
Authoring Tool 18
Output File 29a

Fig. 2

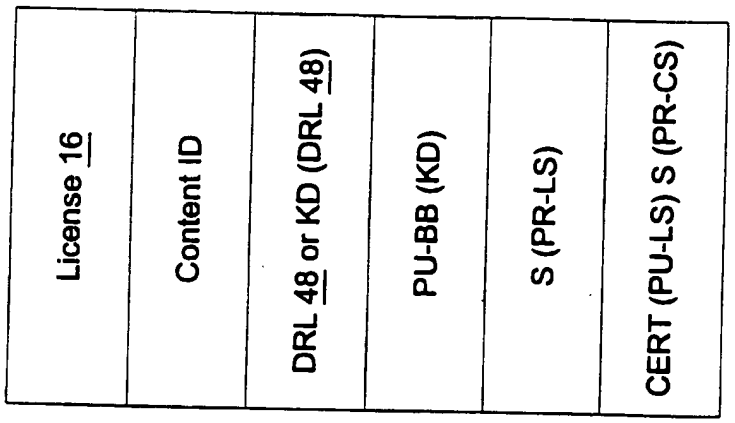

License 16
Content ID
DRL 48 or KD (DRL 48)
PU-BB (KD)
S (PR-LS)
CERT (PU-LS) S (PR-CS)

Fig. 8

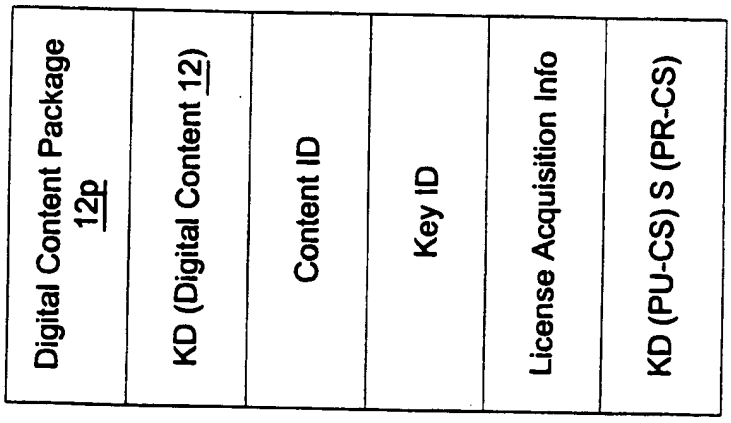

Digital Content Package 12p
KD (Digital Content 12)
Content ID
Key ID
License Acquisition Info
KD (PU-CS) S (PR-CS)

Fig. 3

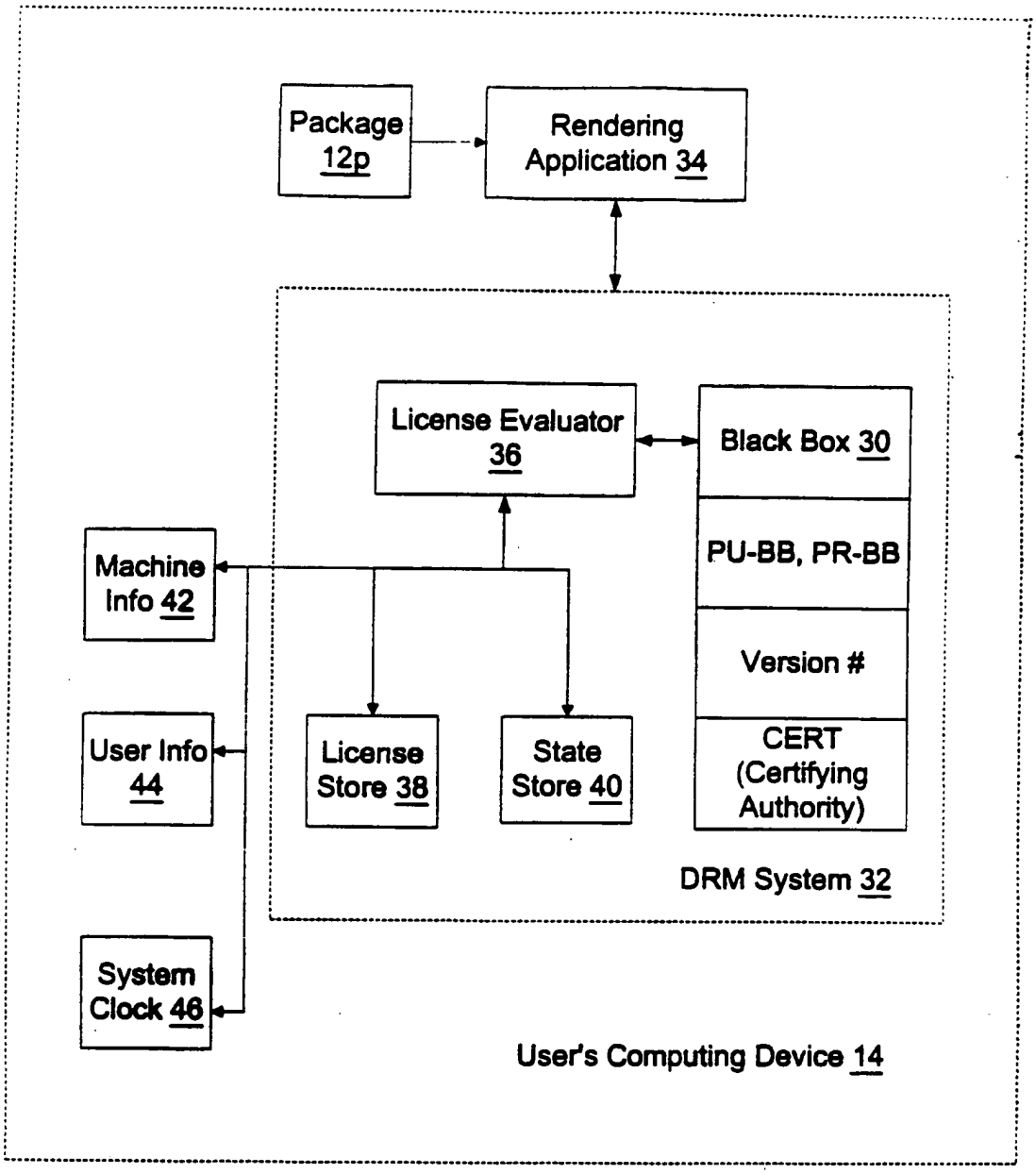

Package 12p
Rendering Application 34
License Evaluator 36
Black Box 30
PU-BB, PR-BB
Version #
CERT (Certifying Authority)
Machine Info 42
User Info 44
License Store 38
State Store 40
DRM System 32
System Clock 46
User's Computing Device 14

Fig. 4

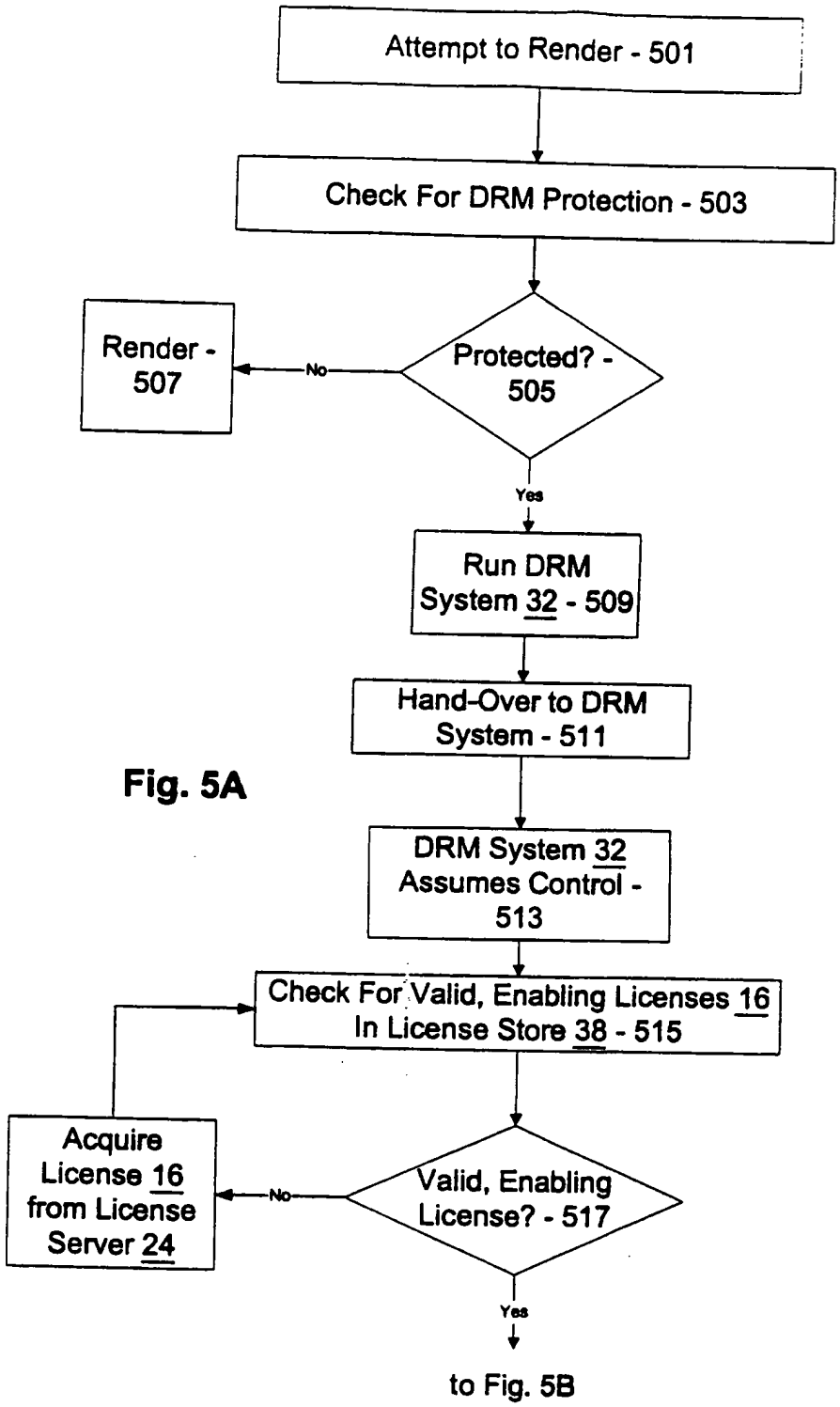

Attempt to Render - 501
Check For DRM Protection - 503
Protected? - 505
No
Render - 507
Yes
Run DRM System 32 - 509
Hand-Over to DRM System - 511
DRM System 32 Assumes Control - 513
Check For Valid, Enabling Licenses 16 In License Store 38 - 515
Valid, Enabling License? - 517
No
Acquire License 16 from License Server 24
Yes
to Fig. 5B

Fig. 5A

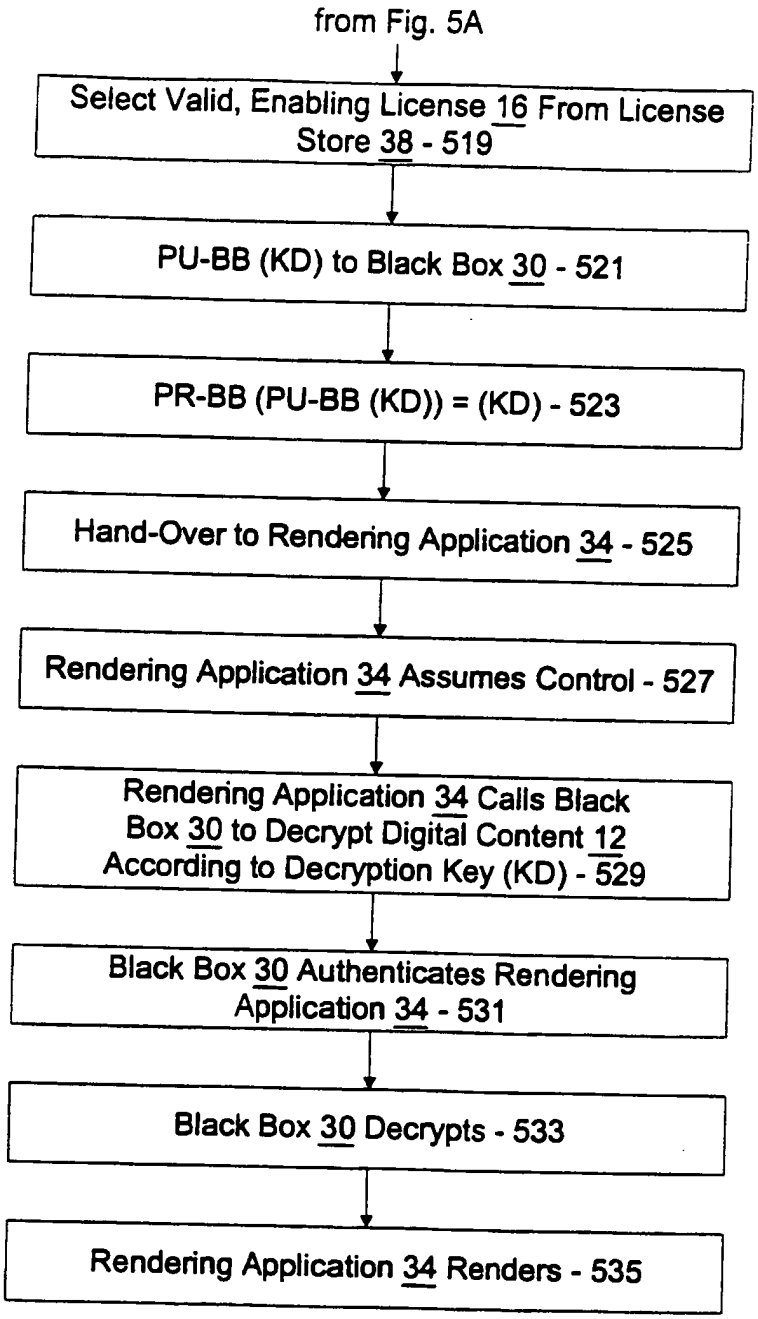

from Fig. 5A
Select Valid, Enabling License 16 From License Store 38 - 519
PU-BB (KD) to Black Box 30 - 521
PR-BB (PU-BB (KD)) = (KD) - 523
Hand-Over to Rendering Application 34 - 525
Rendering Application 34 Assumes Control - 527
Rendering Application 34 Calls Black Box 30 to Decrypt Digital Content 12 According to Decryption Key (KD) - 529
Black Box 30 Authenticates Rendering Application 34 - 531
Black Box 30 Decrypts - 533
Rendering Application 34 Renders - 535

Fig. 5B

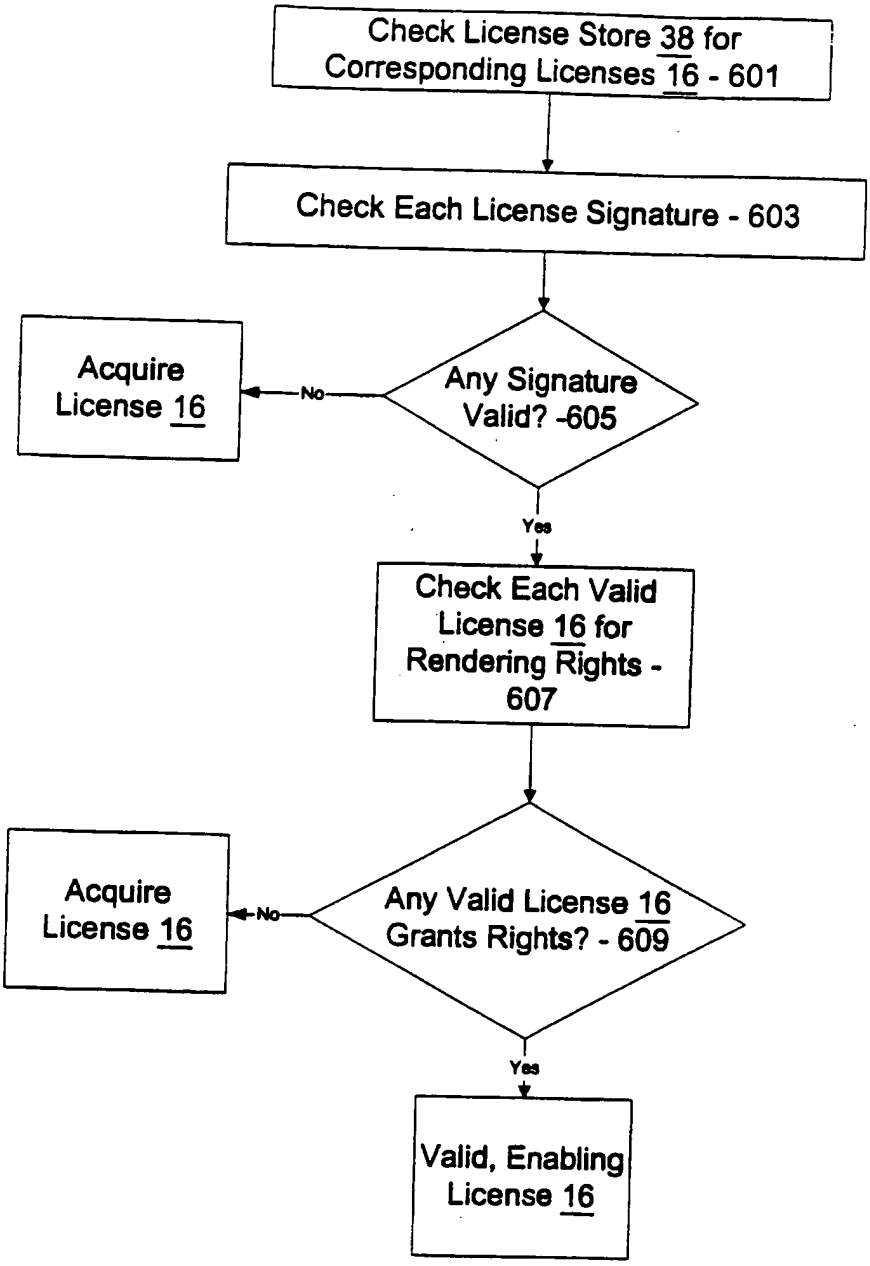

Check License Store 38 for Corresponding Licenses 16 - 601
Check Each License Signature - 603
Any Signature Valid? -605
No
Acquire License 16
Yes
Check Each Valid License 16 for Rendering Rights - 607
Any Valid License 16 Grants Rights? - 609
No
Acquire License 16
Yes
Valid, Enabling License 16

Fig. 6

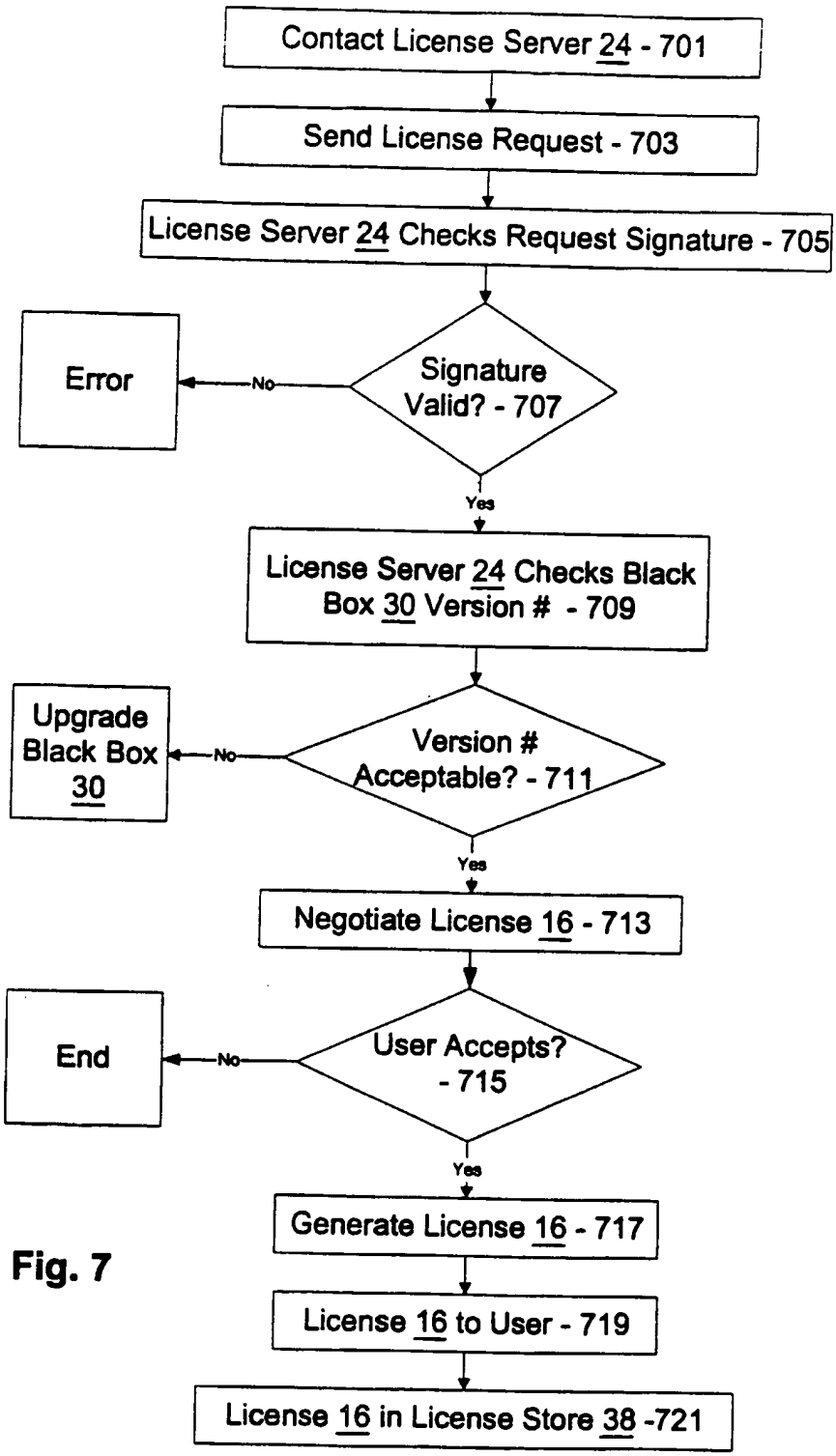

Contact License Server 24 - 701
Send License Request - 703
License Server 24 Checks Request Signature - 705
Signature Valid? - 707
No
Error
Yes
License Server 24 Checks Black Box 30 Version # - 709
Version # Acceptable? - 711
No
Upgrade Black Box 30
Yes
Negotiate License 16 - 713
User Accepts? - 715
No
End
Yes
Generate License 16 - 717
License 16 to User - 719
License 16 in License Store 38 -721

Fig. 7

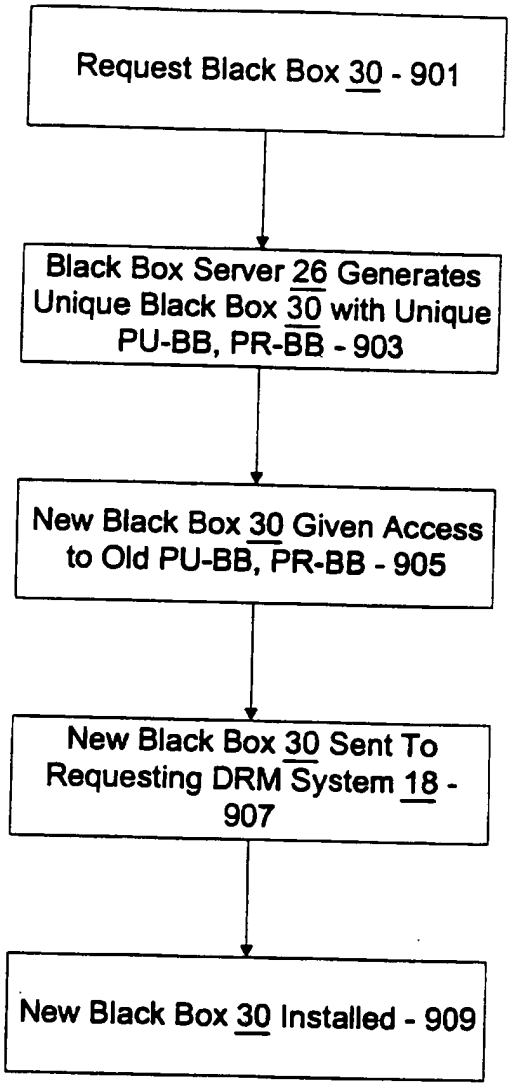

Request Black Box 30 - 901
Black Box Server 26 Generates Unique Black Box 30 with Unique PU-BB, PR-BB - 903
New Black Box 30 Given Access to Old PU-BB, PR-BB - 905
New Black Box 30 Sent To Requesting DRM System 18 - 907
New Black Box 30 Installed - 909

Fig. 9

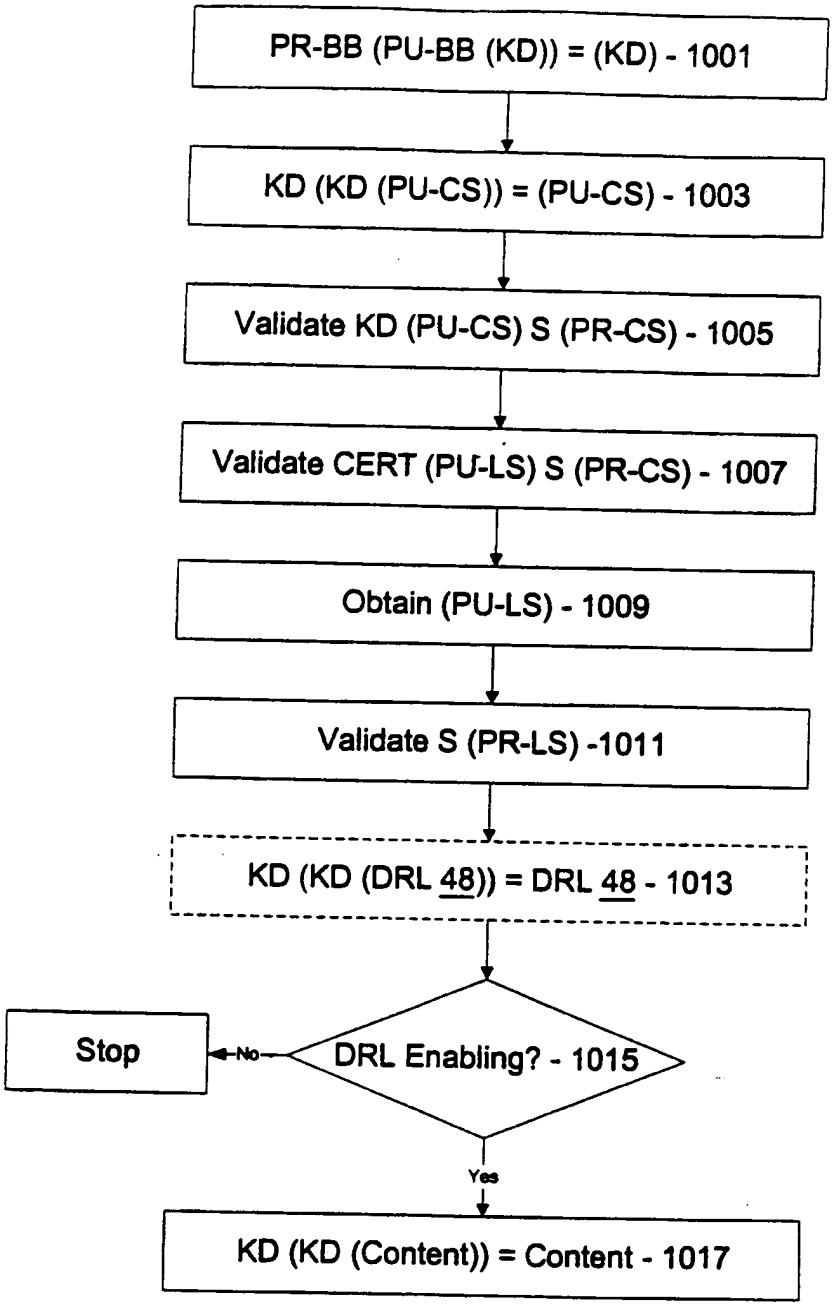

PR-BB (PU-BB (KD)) = (KD) - 1001
KD (KD (PU-CS)) = (PU-CS) - 1003
Validate KD (PU-CS) S (PR-CS) - 1005
Validate CERT (PU-LS) S (PR-CS) - 1007
Obtain (PU-LS) - 1009
Validate S (PR-LS) -1011
KD (KD (DRL 48)) = DRL 48 - 1013
DRL Enabling? - 1015
No
Stop
Yes
KD (KD (Content)) = Content - 1017

Fig. 10

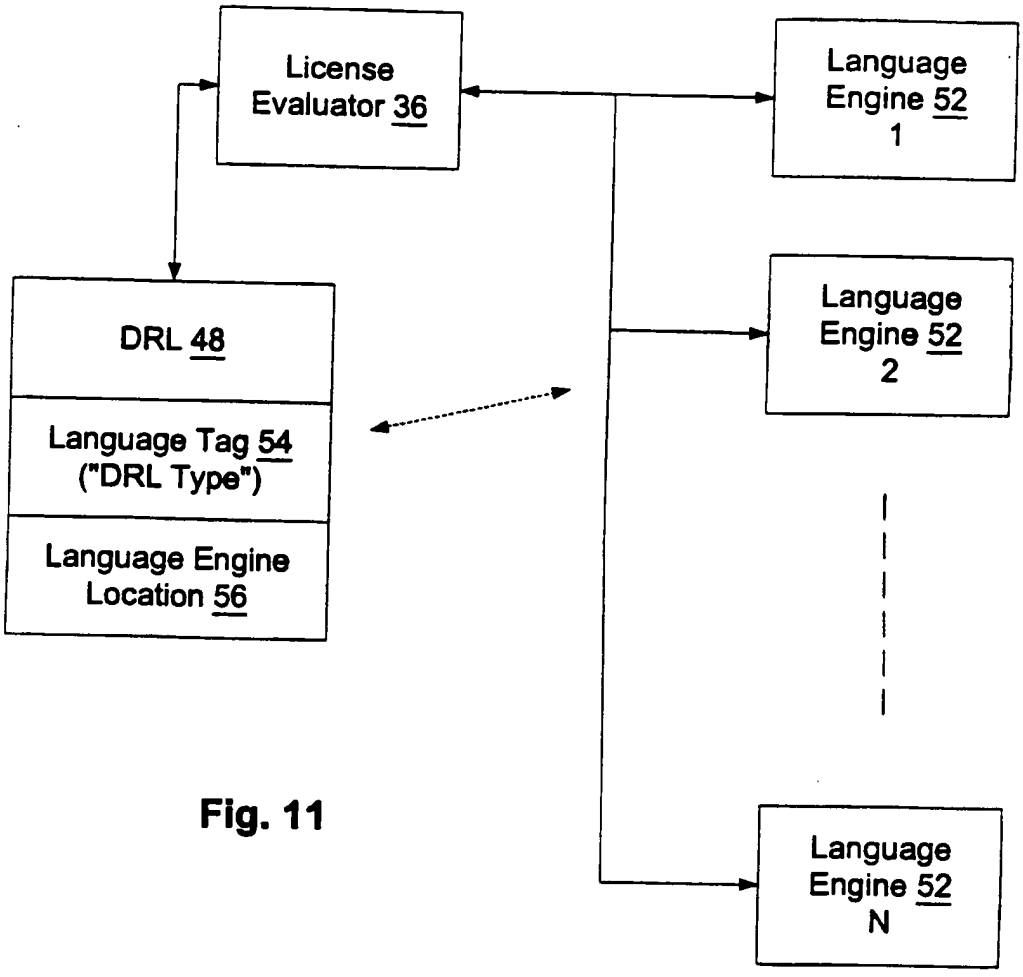

License Evaluator 36
Language Engine 52 1
Language Engine 52 2
Language Engine 52 N
DRL 48
Language Tag 54 ("DRL Type")
Language Engine Location 56

Fig. 11

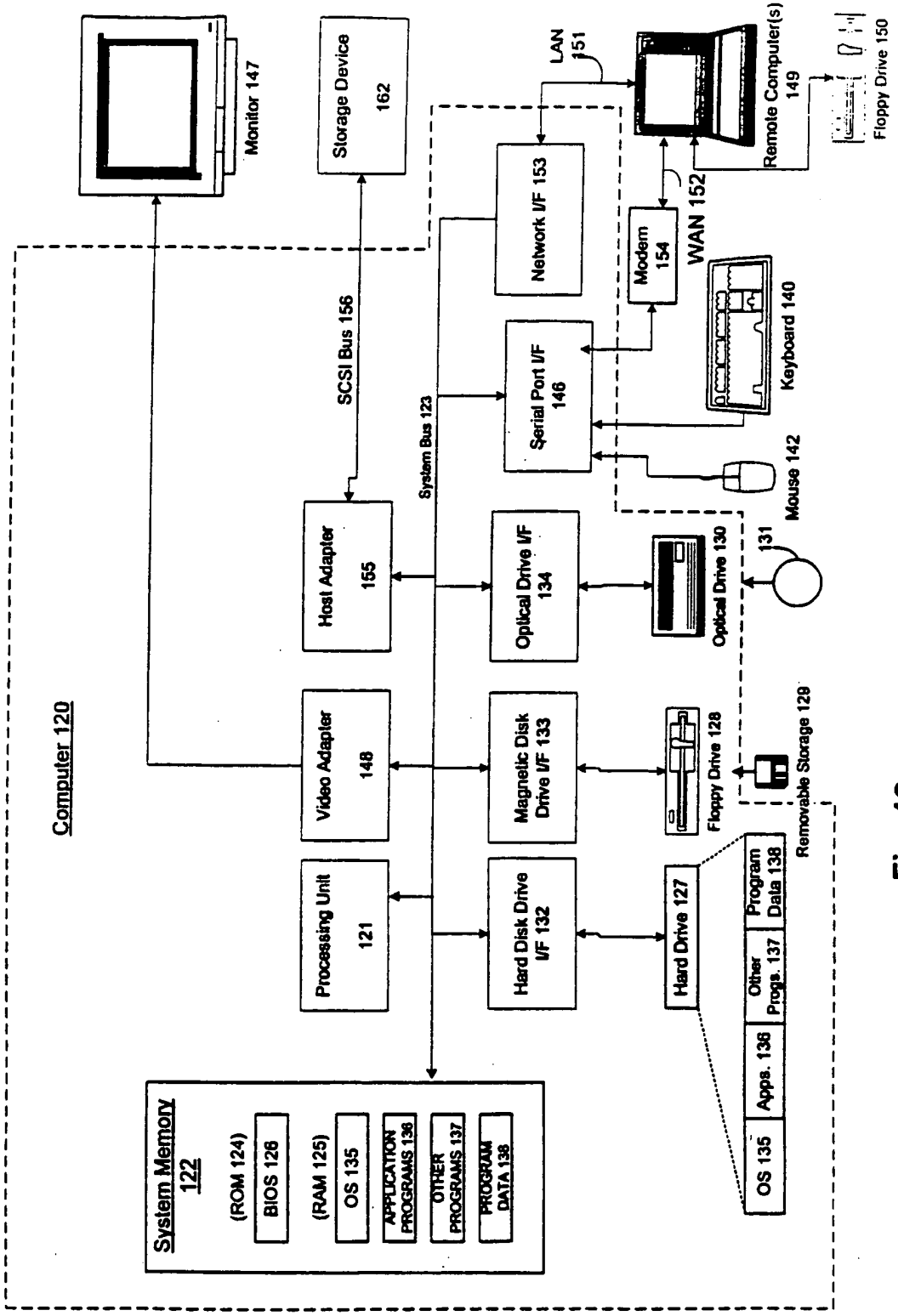

Computer 120
System Memory 122
(ROM 124)
BIOS 126
(RAM 125)
OS 135
APPLICATION PROGRAMS 136
OTHER PROGRAMS 137
PROGRAM DATA 138
Processing Unit 121
Video Adapter 148
Host Adapter 155
Hard Disk Drive I/F 132
Magnetic Disk Drive I/F 133
Optical Drive I/F 134
Serial Port I/F 146
Network I/F 153
Modem 154
WAN 152
LAN 151
SCSI Bus 156
System Bus 123
Monitor 147
Storage Device 162
Remote Computer(s) 149
Floppy Drive 150
Keyboard 140
Mouse 142
131
Optical Drive 130
Floppy Drive 128
Removable Storage 129
Hard Drive 127
OS 135
Apps. 136
Other Progs. 137
Program Data 138

Fig. 12

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
30 November 2000 (30.11.2000)

PCT

(10) International Publication Number
WO 00/72118 A1

(51) International Patent Classification[7]: G06F 1/00

(21) International Application Number: PCT/US00/10213

(22) International Filing Date: 13 April 2000 (13.04.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/316,717 21 May 1999 (21.05.1999) US

(71) Applicant: COMPAQ COMPUTERS INC. [US/US]; 10435 N. Tautau Avenue, Loc 200-16, Cupertino, CA 95014-3548 (US).

(72) Inventors: GLASSMAN, Steven, C.; 615 Palo Alto Avenue, Mountain View, CA 94041 (US). MANASSE, Mark, S.; 1270 Monterey Boulevard, San Francisco, CA 94127 (US).

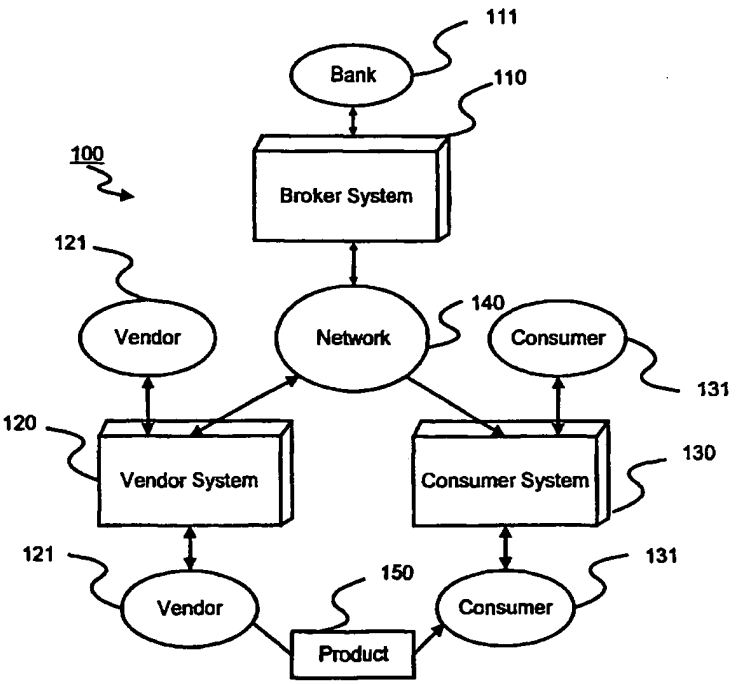
(74) Agents: GRANATELLI, Lawrence; Fenwick & West LLP, Two Palo Alto Square, Palo Alto, CA 94306 et al. (US).

(81) Designated States *(national)*: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU,

[Continued on next page]

(54) Title: METHOD AND SYSTEM FOR ENFORCING LICENSES ON AN OPEN NETWORK



(57) Abstract: An electronic commerce system and method enforces a license agreement for content on an open network (140) by restricting the number of consumers (131) that can concurrently access the content. A consumer (131) initially acquires vendor scrip, either from a broker or the vendor (121) itself. The consumer (131) presents the vendor scrip to the vendor (121) along with a request to access the content. In response, the vendor (121) gathers information about the consumer (131) to determine whether the consumer (131) belongs to the class allowed to access the content. The information may be gathered from the scrip or from other sources. If the consumer (131) belongs to the class, then the vendor (121) determines if a license to access the content is available. Generally, a license is available if the number of other consumers (131) having licenses to access the content is less than the maximum specified in the license agreement. If no licenses are available, the vendor (121) provides the consumer (131) with an estimate of when a license will be available. If a license is available, the vendor (121) directs the consumer (131) to obtain license scrip which allows the consumer (131) to access the content. The license scrip expires after a relatively brief period of time. When the consumer (131) uses the license scrip to access the content, the vendor (121) provides the consumer (131) with new license scrip having a later expiration time.

WO 00/72118 A1

MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:

— *With international search report.*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

METHOD AND SYSTEM FOR ENFORCING LICENSES ON AN OPEN NETWORK

BACKGROUND

FIELD OF THE INVENTION

This invention relates generally to an electronic commerce system and more particularly to a commerce system supporting restricted use of a resource, and even more particularly to a commerce system supporting N-user license agreements.

BACKGROUND OF THE INVENTION

It is common for a library, corporation, or other organization to purchase content that will be made available to members of the organization. Often, the content is subject to a license restriction limiting distribution of the content. For example, a corporation may license or purchase a magazine and then distribute the magazine to interested employees. Typically, the corporation is restricted by the licensing agreement or copyright law from photocopying the magazine. Accordingly, the corporation must either obtain multiple copies of the magazine or circulate the single copy through the organization.

Similarly, the content licensed or purchased by the organization may be in electronic form. For example, the corporation may license a CD-ROM holding an electronic version of the magazine. While the CD-ROM can be loaded onto a server accessible to employees of the corporation via a computer network, the content may be restricted by an N-user license that forbids the corporation from allowing more than N users to simultaneously access the CD-ROM. To implement the restriction, software executing on the server tracks the number of people currently accessing the CD-ROM and blocks usage that exceeds the scope of the license.

In existing systems, the license control is performed by a combination of a specialized lock server and a client program. The lock server validates users' requests for access to the content and maintains the status of active users. The client program interacts with the lock server to acquire a lock and to provide access to the content.

There are many existing implementations of lock servers. However, they all are subject to one or more of the following undesirable restrictions:

each content source has its own, separate, and proprietary lock server;

the user's system already has the content (protected from direct access) and the client program gets the lock to access the content;

acquiring a lock is a complicated action; and/or

the set of valid users is limited.

For these reasons, existing lock servers are undesirable on an open network.

A lock server providing an N-user license on an open network should also support the following requirements:

an unrestricted set of potential users;

no single administrative domain covers all users;

the users do not need to have a separate user application for each source of content;

access to the content can be easily restricted; and

the content exists on the server and not with the user.

Accordingly, there is a need for a way to provide restricted access to electronic content that works with a wide variety of possible access schemes. Preferably, the solution will allow enforcement of an N-user license for content located on an open network like the Internet.

SUMMARY OF THE INVENTION

The above needs are met by a method and system for electronic commerce that uses special scrip - called "license scrip" - to provide temporary licenses to consumers accessing content. Scrip is primarily used as a form of electronic currency, however it can be more generally considered as a one-time token representing a general value. When scrip is used as an electronic currency, its value is monetary. When scrip is used as a temporary license, its value is the permission to access specific content. This permission may be unlimited or it may be for only a relatively brief period of time, say a few minutes to a few hours.

Accessing content with license scrip is very much like buying regular content with monetary scrip. Instead of having a price specified in monetary terms. Each page of content has a price (which may be zero) given in terms of license scrip. A consumer obtains license scrip from the vendor, preferably exchanging regular vendor scrip for the license scrip.

The vendor uses the license scrip to enforce an N-user license agreement - granting up to N people simultaneous access to the content. The vendor tracks the number and identity of consumers currently having licenses to access the content (i.e., consumers currently possessing valid license scrip).

A consumer initially lacks the license scrip needed to access the content. Upon receiving an access request from the consumer, the vendor determines whether a license is available. If a license is not available, the vendor tells the consumer to try again later and, optionally, provides the consumer with an estimate of when a license will be available.

If a license is available, then the vendor directs the consumer to obtain license scrip. Normally, the consumer obtains license scrip by requesting it from the vendor, but the consumer may get the license by any acceptable means. After receiving a license scrip

request. the vendor verifies that the consumer belongs to a class entitled to have a license. For example, if licenses are available to residents of only a certain state, the vendor ensures that the consumer resides in the state before granting the consumer a license.

If a license is available, then the vendor provides the consumer with the license scrip and remembers the granted license. The license scrip is preferably set to expire after a brief time period, but the duration of the license may vary depending upon business or legal concerns. To access content covered by the license, the consumer provides the license scrip when requesting content from the vendor. Each time the consumer accesses the content, the vendor returns replacement license scrip having the same or a later expiration time. Accordingly, the consumer can access the content as long as their license remains valid. When the consumer has not accessed the content for a while, the license scrip expires and the consumer can no longer access the content without obtaining new license scrip.

BRIEF DESCRIPTION OF THE DRAWINGS

FIGURE 1 is a top-level block diagram illustrating a computerized system for conducting electronic commerce;

FIGURE 2 is a block diagram illustrating a computer system used in the system of FIG. 1;

FIGURE 3 is a flow diagram illustrating the operations of the system of FIG. 1;

FIGURE 4 is a block diagram illustrating the data fields of a piece of scrip used in the system of FIG. 1;

FIGURE 5 is a diagram illustrating transactions between a consumer and a vendor utilizing license scrip to enforce an N-user license agreement according to the present invention; and

FIGURE 6 is a flow chart illustrating steps for determining whether to grant a license to a consumer.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

A preferred embodiment of the present invention restricts access to electronic content through the use of an electronic commerce system. Accordingly, it is useful to describe the electronic commerce system before detailing how the system is utilized according to the present invention.

FIG. 1 shows a computerized system 100 for conducting electronic commerce. The system 100 includes a broker system 110, a vendor system 120, and a consumer system 130 interconnected by a communications network 140.

For clarity, the system 100 depicted in FIG. 1 shows only single broker, vendor, and consumer systems. In actual practice, any number of broker, vendor, and consumer systems can be interconnected by the network 140. The network 140 can be public or private, such as, for example, the Internet, an organization's intranet, a switched telephone system, a satellite linked network, or another form of network. The broker 111 using the broker system 110 can be a bank, a credit provider, an Internet service provider, a telephone company, or any institution the consumer trusts to sell electronic currency called "scrip."

The vendor system 120 is operated by a vendor 121. The vendor 121 provides products and/or content 150 of any type to consumers and, in one embodiment, provides content which is available by subscription. Each subscription page (i.e., page of data that is available for "purchase") has a price of zero but requires a special type of scrip, called "subscription scrip," before it can be accessed. Since the price of a page is zero, the consumer 131 can "purchase" an unlimited number of pages once the consumer 131 has the

proper subscription scrip 330. The subscription expires when the subscription scrip 330 expires.

A consumer 131 can use the consumer computer system 130 to electronically acquire the products or content 150 of the vendor 121. As used herein, "consumer" refers to an organization such as a library or corporation, a member of the organization, such as a librarian or an employee, or an individual, such as a person visiting a library or a home computer user. Of course, actions attributed to the organization are usually performed by a member of the organization.

A computer system 200 suitable for use as the broker, vendor, and consumer systems is shown in FIG. 2. The computer system 200 includes a central processing unit (CPU) 210, a memory 220, and an input/output interface 230 connected to each other by a communications bus 240. The CPU 210, at the direction of users 250, e.g. brokers, vendors, and/or consumers, executes software programs, or modules, for manipulating data. The programs and data can be stored in the memory 220 as a database (DB) 221. The DB 221 storing programs and data on the consumer computer system 130 is referred to as a "wallet." In a preferred embodiment of the present invention described herein, many of the operations attributed to the consumer are, in fact, performed automatically by the wallet 221.

The memory 220 can include volatile semiconductor memory as well as persistent storage media, such as disks. The I/O interface 230 is for communicating data with the network 140, the users 250, and other computer system peripheral equipment, such as printers, tapes, etc.

The computer system 200 is scaled in size to function as the broker, vendor, or consumer systems. For example, when scaled as the consumer computer system 130, the computer system 200 can be a small personal computer (PC), fixed or portable. The

configurations of the computer system 200 suitable for use by the broker 111 and the vendor 121 may include multiple processors and large database equipped with "fail-safe" features. The fail-safe features ensure that the database 221 is securely maintained for long periods of time.

FIG. 3 shows an operation of the electronic commerce system 100. The consumer 131 uses currency to purchase electronic broker scrip 320 generated by the broker 111. Here, purchasing means that upon a validation of the authenticity of the consumer 131 and the consumer's currency 310, the broker system 110 generates signals, in the form of data records. The signals are communicated, via the network 140, to the consumer system 130 for storage in the wallet 221 of the memory 220 of the consumer system 130.

The scrip is stamped by the generator of the scrip to carry information that is verifiable by the originator, and any other system that has an explicit agreement with the originator. In addition, each scrip is uniquely identifiable and valid at only a single recipient. After a single use, the recipient of the scrip can invalidate it, meaning that the signals of the data record are no longer accepted for processing by the recipient computer system.

In one embodiment, the consumer 131 exchanges the broker scrip 320 with the broker 111 for vendor scrip 330. To complete this transaction, the broker system 110 executes licensed software programs which generate scrip 330 for consumers as needed. Alternatively, the broker 111, in a similar transaction 303, exchanges currency 310 for bulk vendor scrip 330 which is then sold to consumers.

In another embodiment, the consumer 131 exchanges currency with the vendor 121 for regular vendor. In this latter embodiment, there is no need for a broker 111. In addition, the vendor scrip may be free, meaning that the consumer 131 does not need to exchange currency for the scrip.

The consumer 131, in a transaction 304, provides the scrip 330 to the vendor 121. The vendor 121 checks the stamp of the scrip 330 to verify its authenticity, and also checks to make sure the value of the scrip covers the requested content and has not expired. Approval of the transaction results in the delivery of the desired content 150 to the consumer 131. The vendor 121 can also return 304 modified scrip 330 to the consumer 131 as change.

FIG. 4 is a block diagram illustrating the data fields of a single piece of scrip 400. The scrip 400 is logically separated into seven data fields. The Vendor field 410 identifies the vendor for the scrip 400. The Value field 412 gives the value of the scrip 400. The scrip ID field 414 is the unique identifier of the scrip. The Customer ID field 416 is used by the broker 111 and vendor 121 to verify that the consumer has the right to spend the scrip. The Expires field 418 gives the expiration time for the scrip 400. The Props field 420 holds consumer properties, such as the consumer's age, state of residence, employer, etc. Finally, the Stamp field 422 holds a digital stamp and is used to detect tampering with the scrip 400.

The present invention uses "license" scrip, which can be thought of as special purpose scrip having a short period of validity. A consumer with license scrip has a license to view the content covered by the license until the scrip expires.

FIG. 5 is a diagram illustrating transactions between a consumer 510 and a vendor 512 utilizing license scrip to enforce an N-user license agreement according to the present invention. In the transactions of FIG. 5, the vendor 512, for example, can be a library located at a state university. Assume the library purchases a four user license for a CD-ROM and makes the CD-ROM available to other terminals in the library via a local area network and residents of the state via the Internet. To conform with the license, the library must ensure that no more than four consumers are simultaneously accessing the CD-ROM. In this

example, the library is the vendor 512 and the people who can access the CD-ROM, either in the library or elsewhere, are the consumers 510.

In another example, a newspaper publisher operates a web site. Assume that a corporation purchases a 20 user license allowing up to 20 people from the corporation to simultaneously access content on the web site. To police its license, the publisher tracks the users of its web site and block users who are not licensed or who have exceeded the scope of the applicable license. Accordingly, the newspaper publisher is the vendor 512 and the corporation and its employees are the consumers 510.

Although neither the illustrated transactions nor the above examples directly utilize a broker, there may be circumstances where it is desirable to use a broker 111 to perform one or more of the transactions described below. Those of ordinary skill in the art will understand that certain transactions attributed to the consumer or the vendor can be performed instead by a broker 111. For example, the library and/or newspaper may issue vendor and license scrip directly or rely on a third-party broker for this task.

Turning to FIG. 5, the consumer 510 initially requests 520 content from the vendor 512 without valid license scrip. In response, the vendor 512 checks to determine whether there is an available license (i.e., whether an additional consumer is allowed to view the content under the license). Preferably, the vendor 512 maintains a data structure associated with the licensed content that can be quickly scanned to determine whether a license is available. In one embodiment, this data structure is a simple N-entry array, with each entry holding fields for the expiration time and Customer ID of the consumer 510 having the license. As licenses are granted, the vendor 512 fills in the array until no more entries are available.

If no licenses are available, then the vendor 512 instructs 522 the consumer 510 to try again later. In one embodiment, the vendor 512 scans the data structure to determine when the first license may become available and provides the consumer 510 with that time as a suggestion of when to try to access the content again. If a license is available, then the vendor 512 instructs the consumer 510 to go and obtain license scrip.

In response, the consumer 510 attempts 524 to obtain license scrip from the vendor 512. The vendor 512 determines whether the consumer 510 is entitled to a license (i.e., entitled to view the content). FIG. 6 is a flow chart 600 illustrating steps for determining whether to grant license scrip to the consumer 510. When the vendor 512 receives the request from the consumer 510, the vender retrieves 610 information about the consumer. The vendor 514 may retrieve this information by asking the consumer 510 to provide it, from the scrip used to request the license scrip, from a "cookie" on the consumer's computer system, or from a table of information shared by the vendor 512 and the consumer 510 or a broker 111. Additionally, the wallet 221 on the consumer's computer system 130 may be configured to automatically provide information about the consumer 510 when requested by a vendor 512. Depending on the needs of the vendor 512 and the license agreement for the content, the information that may be gathered in this manner includes whether the consumer 510 is a member of an organization, the state of residence of the consumer, the consumer's age, or any other information that is relevant to determining whether to provide access to the consumer 510.

The vendor 512 uses this information to determine 612 whether the consumer belongs to a class that has access to the content held by the vendor 512. If the consumer does not belong to a class having access, for example, if the consumer is not a state resident, then the

vendor denies 614 access to the consumer 510. Preferably, the vendor 512 directs the consumer 510 to a web page explaining why access was denied.

If the consumer 510 belongs to a class having access, the vendor 512 scans the data structure identifying the current licensees of the content and determines 616 whether an additional license is available. Since there may be a delay between the time the consumer 510 is told to buy license scrip and when the wallet 221 tries to buy the scrip, it is possible that the available license may have been acquired by another consumer during that time. If no licenses are available, then the consumer 510 is told to try again later and optionally given a time when a license may be available.

If a license is available, then the vendor 512 grants 618 the license to the consumer 510. The vendor 512 provides 526 the consumer with license scrip that allows the consumer 510 to access the content. The license scrip preferably has a relatively short validity period, say a few minutes to an hour, and allows the consumer 510 full access to the licensed material for the duration of the scrip. The choice of expiration time for the scrip is a business or legal decision. Since the intention of the license scrip is to hold onto one license slot while the consumer 510 is actively using the content, the duration of the license should cover the time that the consumer 510 is expected to be active. In another embodiment, the duration of the scrip is determined, at least in part, by the type of content accessed by the consumer 510. In addition, the vendor 512 preferably records data about the granted license, including the Customer ID of the consumer 510 and the expiration time of the license in the appropriate data structure.

Each time the consumer 510 wishes to access 528 content held by the vendor 512, the consumer provides the license scrip to the vendor. If the scrip is expired or otherwise invalid. then the consumer's request for access is treated as a request without scrip as illustrated by

transaction 520. If the scrip is valid, then the vendor 512 allows the consumer 510 to access the content. In addition, the vendor 512 provides 530 the consumer 510 with replacement license scrip having an updated expiration time. Typically, the updated expiration time is later than the old expiration time, although it can be the same or earlier. In one embodiment, the vendor 512 grants the consumer 510 less additional time each time the vendor issues new license scrip to ensure that the consumer's license eventually expires and other consumers may eventually access the content. The vendor 512 also updates its data structure to reflect the new expiration date of the consumer's license.

Periodically, the vendor 514 preferably scans the data structure to determine whether any licenses have expired. If so, the entry is purged from the data structure, thereby freeing up a license for another consumer 510. Accordingly, the present invention uses license scrip to enforce an N-user license agreement.

It should be understood that FIG. 5 illustrates only one possible set of transactions. FIG. 3, in combination with FIG. 5, provides insight into other possible transactions. For example, a corporation could purchase an N-user license agreement from a broker 111 to access content on a vendor's system 120. The broker 111 can verify that the corporation is entitled to a license and then issue the license scrip from a special scrip series corresponding to the number of users covered by the license. The vendor 121 knows from the scrip series to restrict access from consumers using that license scrip.

Having described a preferred embodiment of the invention, it will now become apparent to those skilled in the art that other embodiments incorporating its concepts may be provided. It is felt therefore, that this invention should not be limited to the disclosed invention, but should be limited only by the spirit and scope of the appended claims.

CLAIMS

We claim:

1. A method of restricting simultaneous access to content, comprising the steps of:

receiving a request to access the content from a consumer;
determining whether the consumer is entitled to access the content; and
responsive to a positive determination, providing the consumer with license scrip allowing access to the content.

2. The method of claim 1, wherein the request to access the content is accompanied by license scrip having an expiration time and wherein the providing step provides the consumer with additional license scrip having an updated expiration time.

3. The method of claim 1, wherein the license scrip has an expiration time and further comprising the steps of:

receiving a second request to access the content from the consumer, the second request including the license scrip; and
responsive to the second request, providing the consumer with replacement license scrip having an updated expiration time.

4. The method of claim 1, wherein the step of determining whether the consumer is entitled to access the content comprises the steps of:

determining whether the consumer belongs to a class having access to the content; and
determining whether a license to access the content is available.

5. The method of claim 4, wherein the step of determining whether the consumer belongs to a class having access to the content comprises the step of:

determining information about the consumer from scrip utilized to request access to the content.

6. The method of claim 4, wherein the step of determining whether a license to access the content is available comprises the steps of:

determining a number of consumers that have licenses to access the content; and

determining a number of allowed licenses;

wherein a license to access the content is available if the number of consumers that have licenses to access the content is less than the number of allowed licenses.

7. The method of claim 4, further comprising the step of:

responsive to a determination that no licenses to access the content are available, providing the consumer with an estimate of when a license will be available.

8. A computer program product having computer-readable instructions embodied thereon for restricting access to content stored on a computer system, the computer-readable instructions comprising instructions for:

receiving a request to access the content stored on the computer system, the request accompanied by scrip;

determining whether the scrip authorizes access to the content;

responsive to a determination that the scrip does not authorize access to the content, determining whether scrip authorizing access to the content is available; and

responsive to a determination that scrip authorizing access to the content is available, providing the scrip.

9. The computer program product of claim 8, further comprising instructions for:

responsive to a determination that the scrip authorizes access to the content, providing replacement scrip having an updated expiration time.

10. The computer program product of claim 8, wherein the instructions for determining whether the scrip authorizes access to the content further comprise computer instructions for:

determining a type of the scrip accompanying the request; and

responsive to a determination that accompanying scrip is license scrip, determining whether the license scrip has expired, wherein unexpired license scrip authorizes access to the content.

11. The computer program product of claim 8, wherein the instructions for determining whether scrip authorizing access to the content is available comprise instructions for:

determining a maximum number of requesters that can be authorized to access the content;

determining whether a current number of requesters authorized to access the content is less than the maximum number of requesters; and

responsive to a determination that the current number of requesters authorized to access the content is less than the maximum number of requesters, determining that scrip authorizing access to the content is available.

12. The computer program product of claim 8, further comprising instructions for:

responsive to a determination that scrip authorizing access to the content is not available, calculating an estimate of when the scrip authorizing access will be available.

13. A computer system for limiting a number of users that can access content stored on a server associated with the computer system, the computer system comprising:

a module for receiving a request from a user to access the content stored on the server;

a module for determining the number of users currently having rights to access the content; and

a module for providing the user with license scrip if the number of users currently having rights to access the content is less than a number of users allowed to access the content, the license scrip granting the user the right to access the content.

14. The system of claim 13, wherein the module for determining the number of users currently having access rights to content comprises:

a module for scanning a data structure stored in a memory of the computer system, the data structure having one or more entries indicating the number of users having access rights to the content.

15. The computer system of claim 14, wherein the data structure indicates when users' rights to access the content expire, further comprising:

a module for purging the entries of users whose right to access the content has expired.

16. The system of claim 13, wherein only a privileged class can access the content, further comprising:

a module for determining whether the user is a member of the privileged class.

17. The system of claim 13, wherein the license scrip grants the user the right to access the content until an expiration time.

18. The system of claim 17, further comprising:

a module for receiving a second request from the user to access the content stored on the server accompanied by the license scrip; and

a module for providing the user with replacement license scrip having a later expiration time.

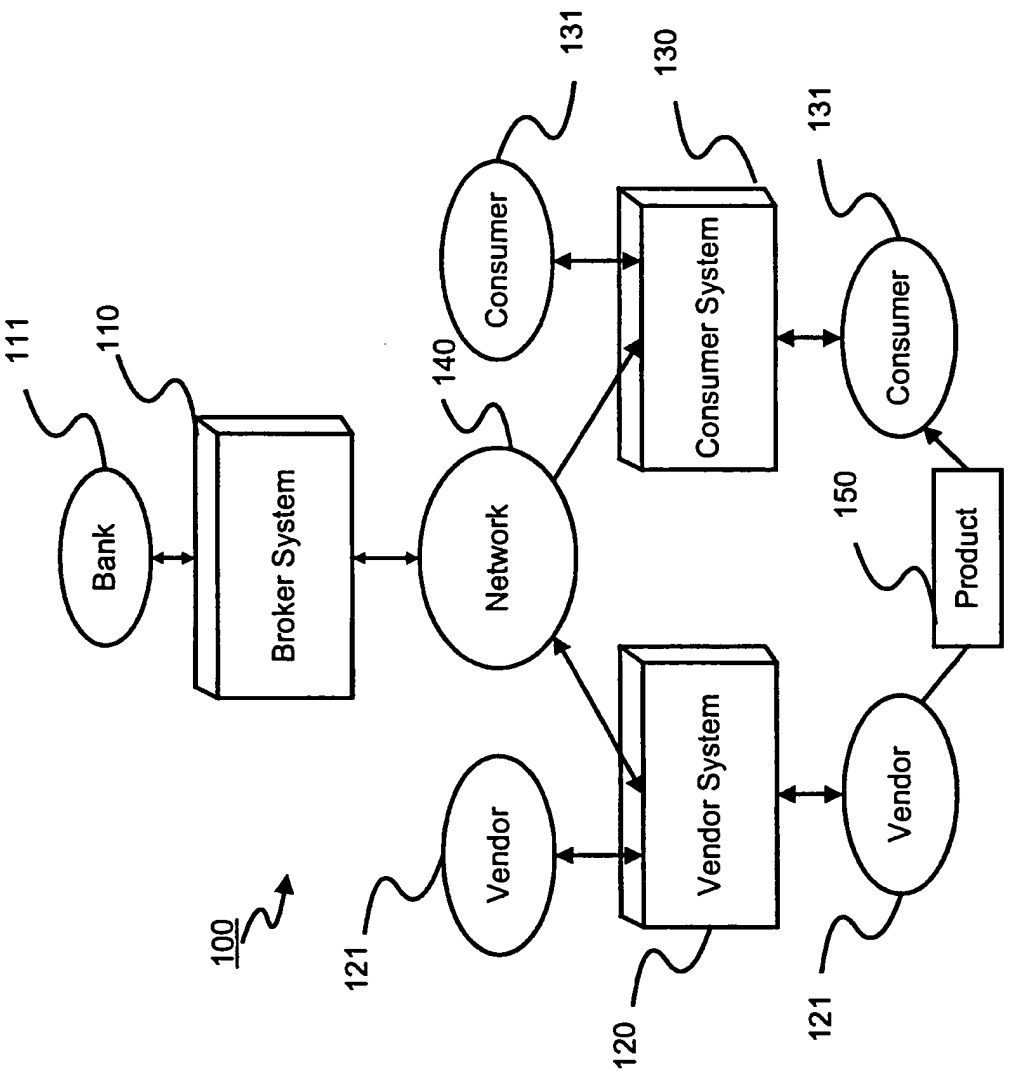


FIG. 1

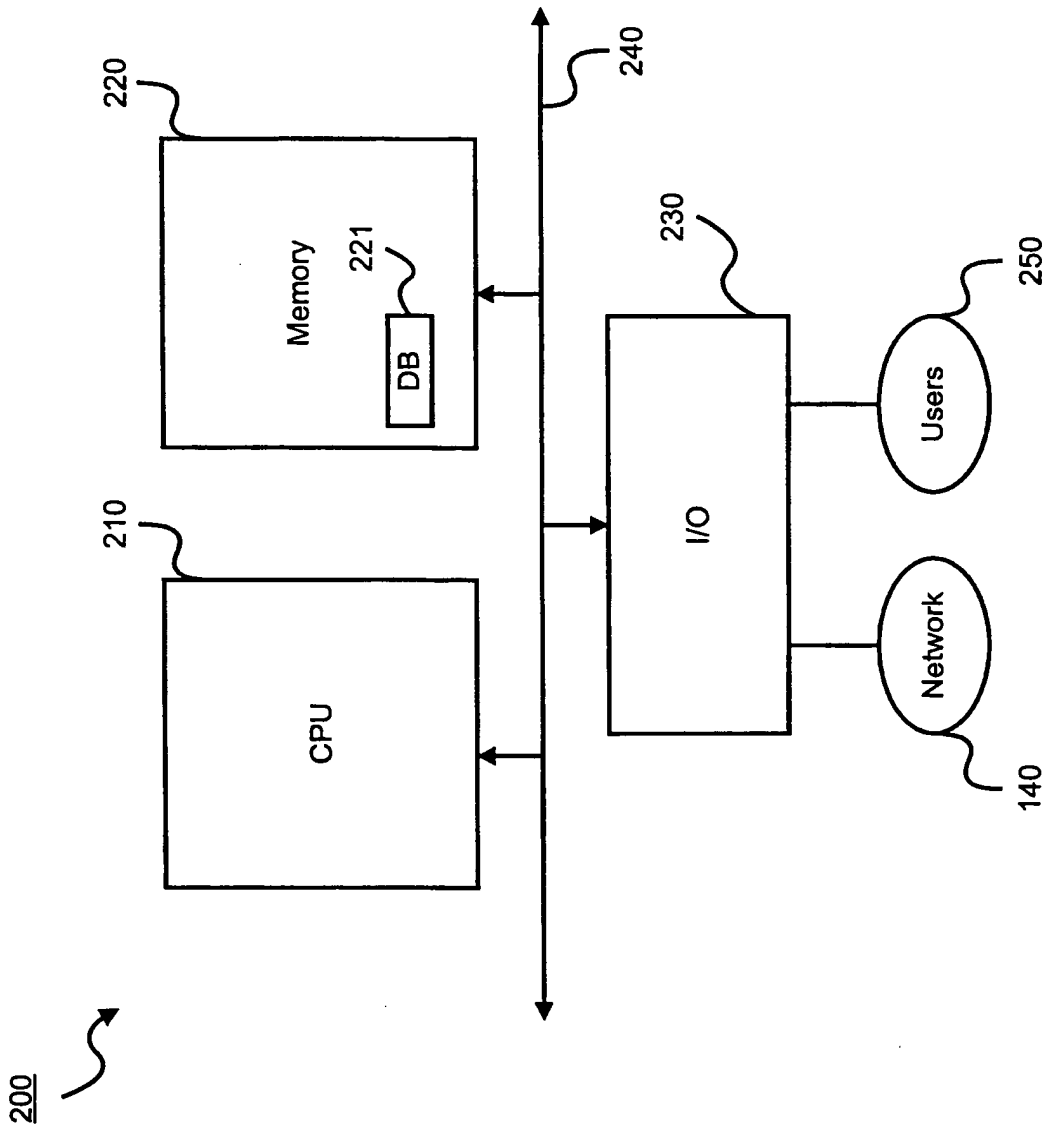


FIG. 2

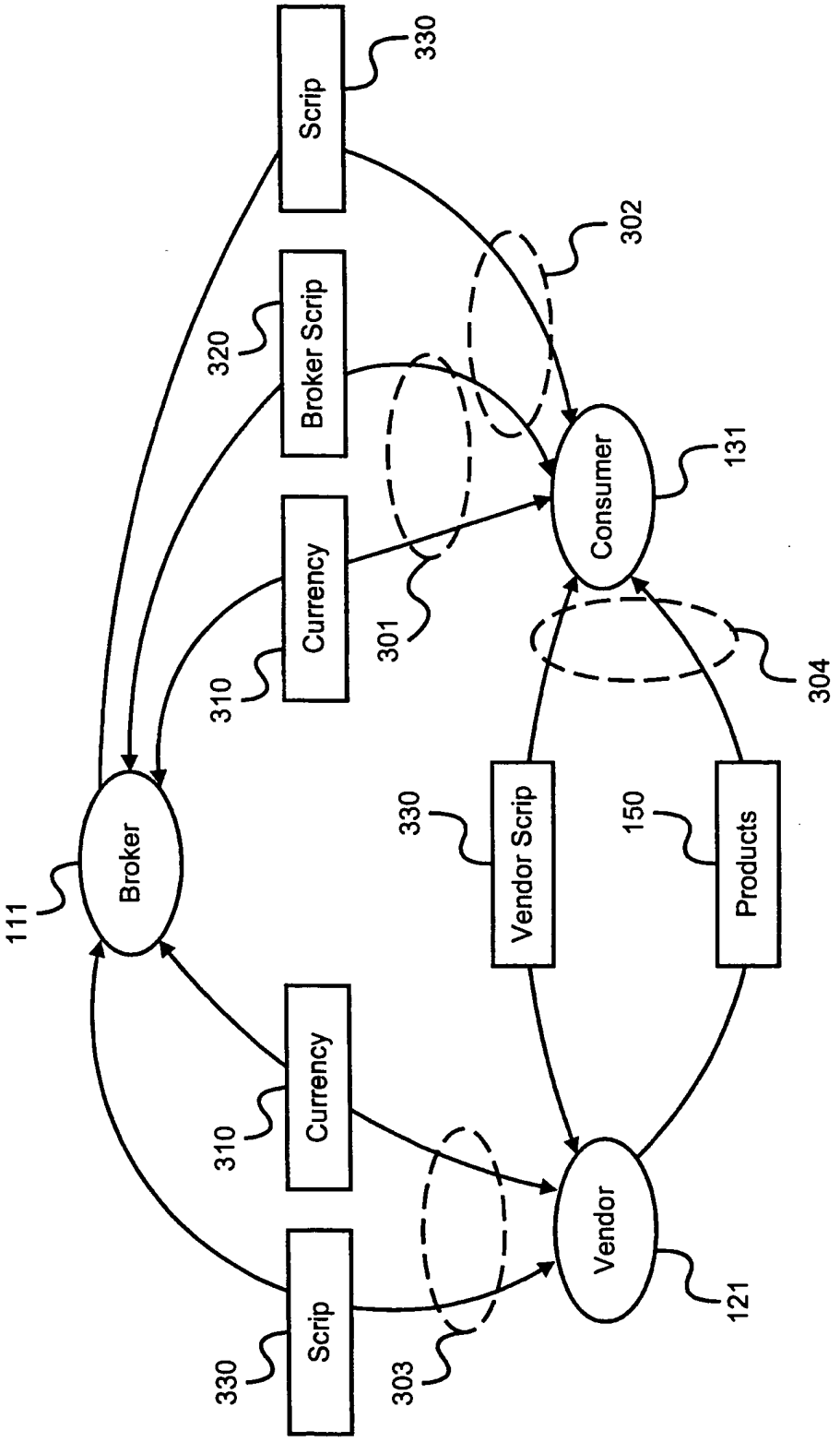


FIG. 3

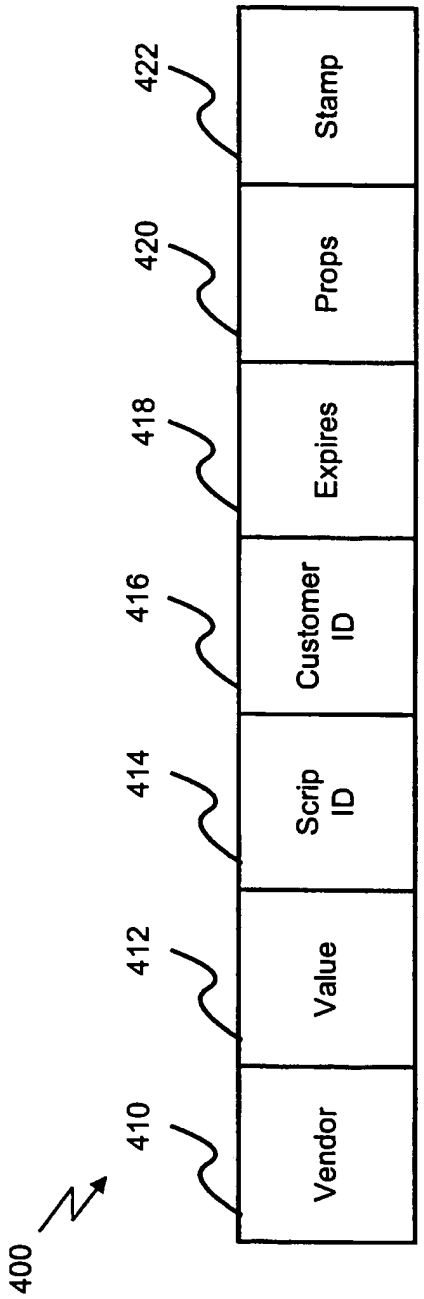


FIG. 4

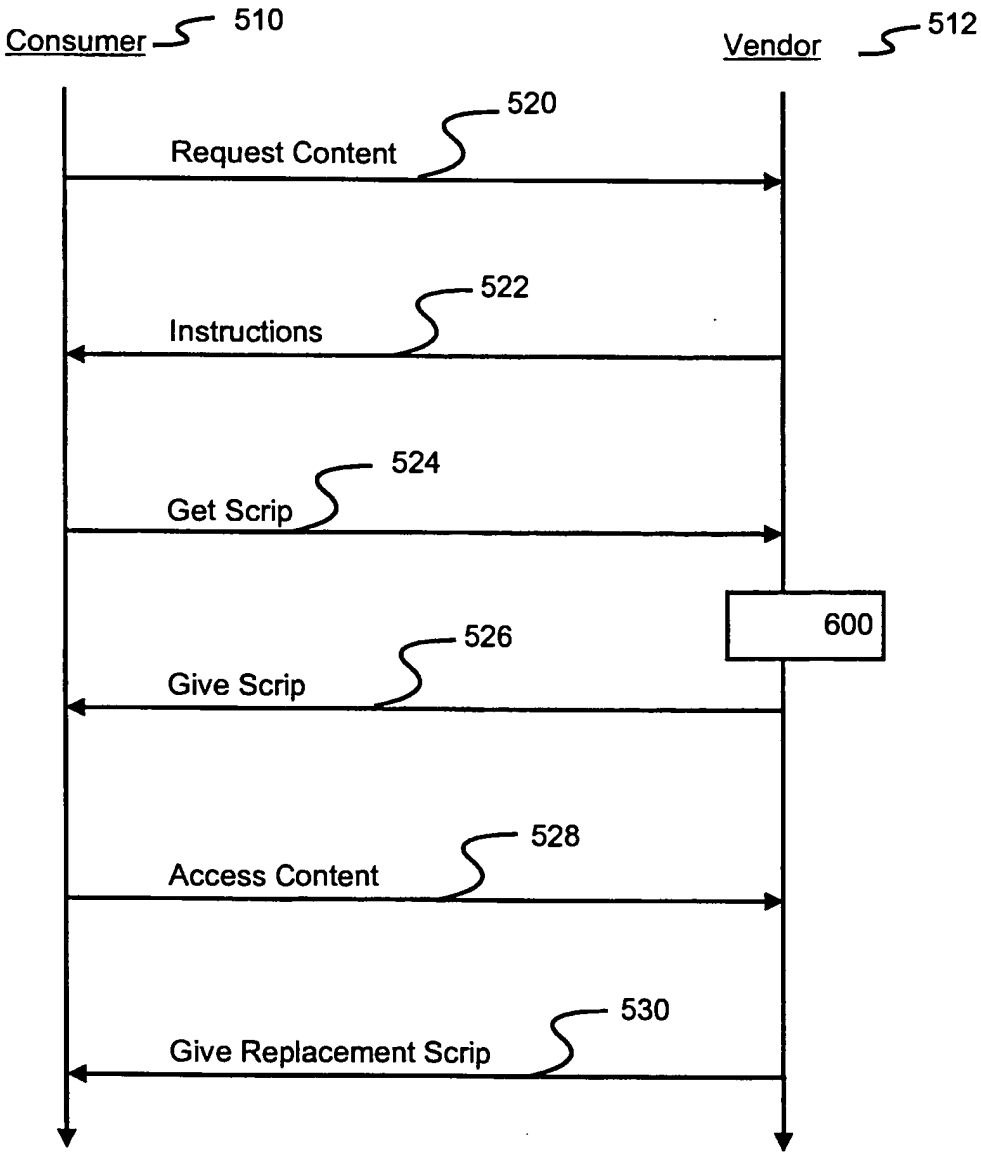
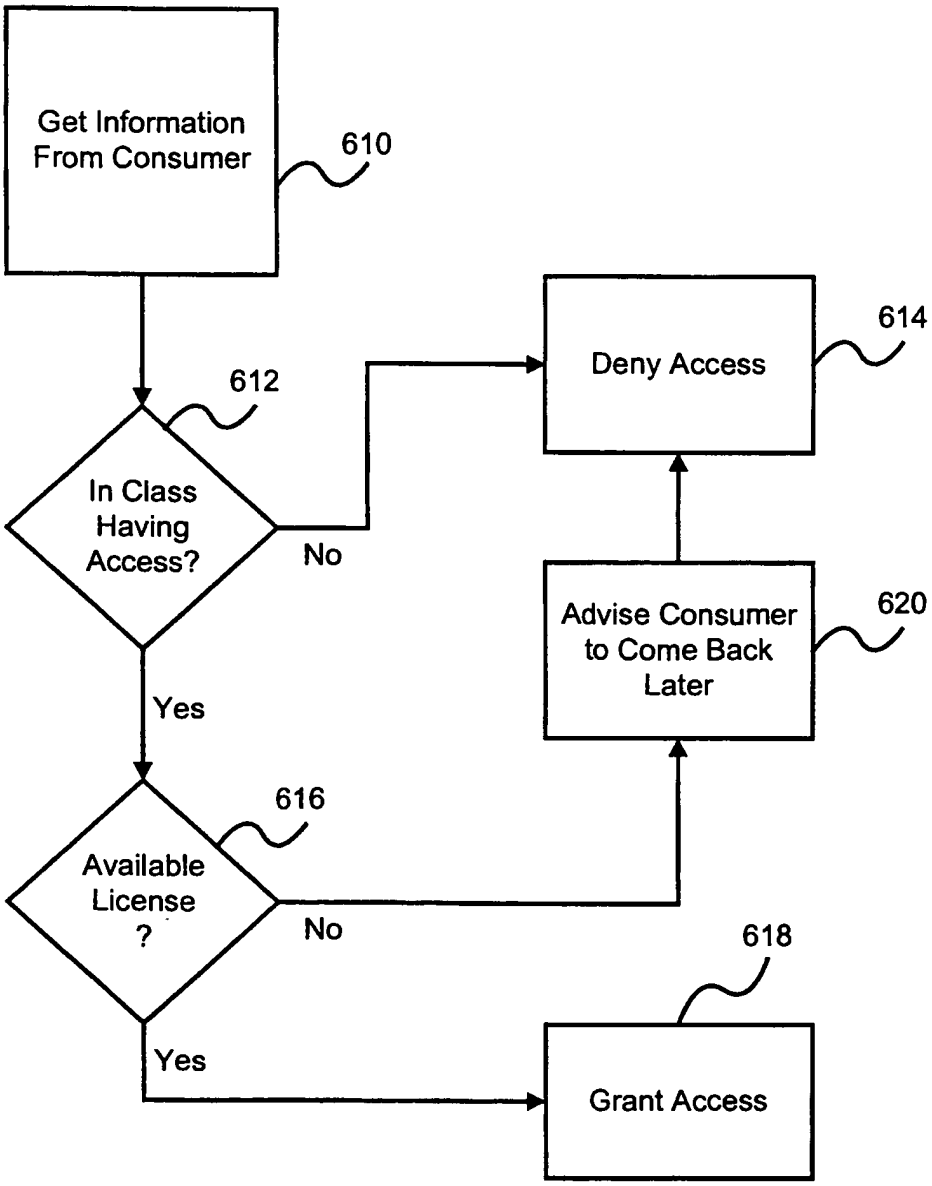


FIG. 5



INTERNATIONAL SEARCH REPORT

Interna al Application No
PCT/US 00/10213

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F1/00

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F G07F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal, WPI Data, PAJ

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	WO 98 58306 A (OYLER SCOTT ;GUTHRIE JOHN (US); TECHWAVE INC (US); KRISHNAN GANAPA) 23 December 1998 (1998-12-23)	1,4-6,8, 10,11, 13,14, 16,17
Y		18
A		2,3,7,9, 12,15
	abstract page 6, line 4 -page 8, line 10 page 10, line 8 -page 16, line 17 page 28, line 7 -page 30, line 23 page 39, line 2 - line 11 figures 1-4	
	— -/—	

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
2 August 2000	09/08/2000

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Jacobs, P

2

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Intern nal Application No
PCT/US 00/10213

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	WO 97 03423 A (DIGITAL EQUIPMENT CORP) 30 January 1997 (1997-01-30)	1-4,8-10
Y		18
A		5,13,16, 17
	page 4, line 15 -page 11, line 33 figures 1-5	
X	US 5 905 860 A (BRINGHURST ADAM L ET AL) 18 May 1999 (1999-05-18)	1,4-6, 13,14, 16,17
A		2,3, 7-12,15, 18
	abstract column 2, line 40 -column 16, line 3	
X	GB 2 316 503 A (ICL PERSONAL SYSTEMS OY) 25 February 1998 (1998-02-25)	1,4-6, 13,14, 16,17
A		2,3, 7-12,15, 18
	abstract page 6 -page 21 figures 1-4 claim 1	

2

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

...rmation on patent family members

Intern... ...al Application No
PCT/US 00/10213

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 9858306 A	23-12-1998	US 6073124 A	06-06-2000
		AU 8150598 A	04-01-1999
WO 9703423 A	30-01-1997	US 5802497 A	01-09-1998
		BR 9606450 A	30-09-1997
		EP 0796480 A	24-09-1997
		IL 117195 A	20-06-1999
		JP 2984731 B	29-11-1999
		JP 9510814 T	28-10-1997
US 5905860 A	18-05-1999	US 5758069 A	26-05-1998
GB 2316503 A	25-02-1998	NONE	

Form PCT/ISA/210 (patent family annex) (July 1992)

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
7 December 2000 (07.12.2000)

PCT

(10) International Publication Number
WO 00/73922 A2

(51) International Patent Classification[7]: G06F 17/00

(21) International Application Number: PCT/US00/11078

(22) International Filing Date: 25 April 2000 (25.04.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/323,635 1 June 1999 (01.06.1999) US

(71) Applicant: ENTERA, INC. [US/US]; 40971 Encyclopedia Circle, Fremont, CA 94538 (US).

(72) Inventor: SCHARBER, John, M.; 1616 Placer Circle, Livermore, CA 94550 (US).

(74) Agents: FAHMI, Tarek, N. et al.; Blakely, Sokoloff, Taylor & Zafman LLP, 7th floor, 12400 Wilshire Boulevard, Los Angeles, CA 90025 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *Without international search report and to be republished upon receipt of that report.*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

WO 00/73922 A2

(54) Title: CONTENT DELIVERY SYSTEM

(57) Abstract: Disclosed is a network content delivery system configured to: select a first content routing technique for processing a first set of network content; and select a second content routing technique for processing a second set of network content, wherein the first and second content routing techniques are selected based on one or more content routing variables. Also disclosed is a content delivery system comprising: a network node for storing network content; a first transmission medium communicatively coupled to the network node for transmitting a first set of network content to the network node; and a second transmission medium communicatively coupled to the network node for transmitting a second set of network content to the network node, wherein the first and second sets of network content are selected based on one or more routing variables.

CONTENT DELIVERY SYSTEM
1

BACKGROUND OF THE INVENTION

Field of the Invention

This invention relates to the transmission and storage of digital information across a network. More particularly, the invention relates to an improved system and method for caching and/or delivering various types of digital content using a plurality of network protocols.

Description of the Related Art

The World Wide Web (hereinafter "the Web") is a network paradigm which links documents known as "Web pages" locally or remotely across multiple network nodes (i.e., Web servers). A single Web page may have links (a.k.a., "hyperlinks") which point to numerous other Web pages. When a user points and clicks on a link using a cursor control device such as a mouse, the user can jump from the initial page to another page, regardless of where the Web pages are actually located. For example, the initial Web page might be stored on a Web server in New York and the second page (accessed via the hyperlink in the first page) might be stored on a Web server in California.

The underlying principles of the Web were developed 1989 at the European Center for Nuclear Research (CERN) in Geneva. By 1994 there were approximately 500 Web servers on the Internet. Today there are more than a million, with new sites starting up at an extraordinary rate. In sum, the Web has become the center of Internet activity and is the primary reason for the explosive growth of the Internet over the past several years.

In addition to providing a simple point-and-click interface to vast amounts of information on the Internet, the Web is quickly turning into a content delivery system. Well known Internet browsers such as Netscape Navigator™ and Microsoft Internet Explorer™ frequently provide plug-in software which allow additional features to be incorporated into the browser program. These include, for example, support for audio and video streaming, telephony, and videoconferencing.

The unparalleled increase in Web usage combined with the incorporation of high bandwidth applications (i.e., audio and video) into browser programs has created serious

performance/bandwidth problems for most Internet Service Providers (hereinafter "ISPs"). Moreover, the network traffic resulting from non-Web-based Internet services such as Internet News (commonly known as "Usenet" News) has increased on the same scale as the increase in Web traffic, thereby further adding to the bandwidth problems experienced by most ISPs.

These issues will be described in more detail with respect to **Figure 1** which illustrates an ISP 100 with a link 160 to a larger network 150 (e.g., the Internet) through which a plurality of clients 130, 120 can access a plurality of Web servers 140-144 and/or News servers 146-148. Maintaining a link 160 to the Internet 150 with enough bandwidth to handle the continually increasing traffic requirements of its clients 120, 130 represents a significant cost for ISP. At the same time, ISP 110 must absorb this cost in order to provide an adequate user experience for its clients 120, 130.

One system which is currently implemented to reduce network traffic across link 160 is a proxy server 210 with a Web cache 220, illustrated in **Figure 2**. When client 120 initially clicks on a hyperlink and requests a Web page (shown as address "www.isp.com/page.html") stored on Web server 144, client 120 will use proxy server 210 as a "proxy agent." This means that proxy server 210 will make the request for the Web page on behalf of client 120 as shown. Once the page has been retrieved and forwarded to client 120, proxy server will store a copy of the Web page locally in Web cache 220. Thus, when client 120 or another client – e.g., client 130 – makes a subsequent request for the same Web page, proxy server 210 will immediately transfer the Web page from its Web cache 210 to client 130. As a result, the speed with which client 130 receives the requested page is substantially increased, and at the same time, no additional bandwidth is consumed across Internet link 160.

While the foregoing proxy server configuration alleviates some of the network traffic across Internet link 160, several problems remain. One problem is that prior Web cache configurations do not have sufficient intelligence to deal with certain types of Web pages (or other Web-based information). For example, numerous Web pages and associated content can only be viewed by a client who pays a subscriber fee. As such, only those clients which provide proper authentication should be permitted to download the information. Today, proxy servers such as proxy server 210 will simply not cache a Web document which requires authentication.

In addition, Web caches do not address the increasing bandwidth problem associated with non-Web based Internet information. In particular, little has been done to alleviate the increasing bandwidth problems created by Usenet news streams. In fact, ISPs today must set aside a substantial amount of bandwidth to provide a continual Usenet news feed to its clients. Moreover, no mechanism is currently available for caching other data transmissions such as the streaming of digital audio and video. The term "streaming" implies a one-way transmission from a server to a client which provides for uninterrupted sound or video. When receiving a streaming transmission, the client will buffer a few seconds of audio or video information before it starts sending the information to a pair of speakers and/or a monitor, thus compensating for momentary delays in packet delivery across the network.

Accordingly, what is needed is a content delivery system which will reduce the bandwidth requirements for ISP 110 while still providing clients 120, 130 with an adequate user experience. What is also needed is a system which will work seamlessly with different types of Web-based and non-Web-based information and which can be implemented on currently available hardware and software platforms. What is also needed is an intelligent content delivery system which is capable of caching all types of Web-based information, including information which requires the authentication of a client before it can be accessed. What is also needed is a content delivery system which is easily adaptable so that it can be easily reconfigured to handle the caching of new Internet information and protocols. Finally, what is needed is a data replication system which runs on a distributed database engine, thereby incorporating well known distributed database procedures for maintaining cache coherency.

SUMMARY OF THE INVENTION

Disclosed is a network content delivery system configured to: select a first content routing technique for processing a first set of network content; and select a second content routing technique for processing a second set of network content, wherein the first and second content routing techniques are selected based on one or more content routing variables.

Also disclosed is a content delivery system comprising: a network node for storing network content; a first transmission medium communicatively coupled to the network node for transmitting a first set of network content to the network node; and a second transmission medium communicatively coupled to the network node for transmitting a second set of

network content to the network node, wherein the first and second sets of network content are selected based on one or more routing variables.

BRIEF DESCRIPTION OF THE DRAWINGS

A better understanding of the present invention can be obtained from the following detailed description in conjunction with the following drawings, in which:

FIG. 1 illustrates generally a network over which an ISP and a plurality of servers communicate.

FIG. 2 illustrates an ISP implementing a proxy server Web cache.

FIG. 3 illustrates one embodiment of the underlying architecture of an Internet content delivery system node.

FIG. 4 illustrates a plurality of Internet content delivery system nodes communicating across a network.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

One embodiment of the present system is a computer comprising a processor and a memory with which software implementing the functionality of the internet content delivery system described herein is executed. Such a computer system stores and communicates (internally or with other computer systems over a network) code and data using machine readable media, such as magnetic disks, random access memory, read only memory, carrier waves, signals, etc. In addition, while one embodiment is described in which the parts of the present invention are implemented in software, alternative embodiments can implement one or more of these parts using any combination of software, firmware and/or hardware.

The underlying architecture of one embodiment of the present internet content delivery system (hereinafter "ICDS") is illustrated in **Figure 3**. A single ICDS node 300 is shown including a cache 330, an ICDS application programming interface (hereinafter "API") 360 which includes a distributed database engine 361, and a plurality of software modules 310-326

which interface with the ICDS API 360. ICDS node 300 may communicate over a network 340 (e.g., the Internet) over communication link 370 and may also interface with a plurality of clients 350-351 and/or other ICDS nodes (e.g., through link 380).

As is known in the art, an API such as ICDS API 360 is comprised of a plurality of subroutines which can be invoked by application software (i.e., software written to operate in conjunction with the particular API). Thus, in **Figure 3** each of the plurality of software modules 310-326 may be uniquely tailored to meet the specific needs of a particular ISP. The modules interface with API 360 by making calls to the API's set of predefined subroutines. Another significant feature of ICDS API 360 is that it is platform-independent. Accordingly, it can be implemented on numerous hardware platforms including those that are Intel-based, Macintosh-based and Sun Microsystems-based.

In one embodiment, a portion of API 360's subroutines and a set of prefabricated modules can be marketed together as a Software Development Kit (hereinafter "SDK"). This will allow ISPs, corporations and/or end-users to customize the type of internet content delivery/caching which they require. In addition, because modules 310-326 may be dynamically linked, they may be loaded and unloaded without having to reboot the hardware platform on which cache 330 is executed.

I. Distributed Content Processing

As illustrated, ICDS node 300 includes a plurality of network protocol modules 310-319 which interface with API 360. These modules provide caching support on ICDS node 300 for numerous different Internet protocols including, but not limited to, Web protocols such as the Hypertext Transfer Protocol (hereinafter "HTTP") 310, Usenet news protocols such as the Network News Transport Protocol (hereinafter "NNRP") 312, directory access protocols such as the Lightweight Directory Access Protocol (hereinafter "LDAP") 314, data streaming protocols such as the Real Time Streaming Protocol (hereinafter "RTSP") 316, and protocols used to perform Wide Area Load Balancing (hereinafter "WALB") 318. Because the underlying architecture of the present ICDS system includes an open API, new protocol modules (e.g., module 319) can be seamlessly added to the system as needed.

One embodiment of the ICDS system includes a plurality of standardized service definitions through which individual service modules 320-326 may be configured to interface

with the ICDS API 360. These service modules provide the underlying functionality of ICDS node 300 and may include a data services module 320, an access services module 322, a transaction services module 323, a commercial services module 324, a directory services module 325, and a resource services module 326. The functionality of each of these modules will be described in more detail below.

In one embodiment of the ICDS system, the ICDS API includes a distributed relational database engine 361. As a result, a plurality of ICDS nodes 410-440 can be distributed across ISP 400's internal network and still maintain a coherent, up-to-date storage of Internet content. For example, if a particular data object is updated at two nodes simultaneously, the underlying distributed database system may be configured to resolve any conflicts between the two modifications using a predefined set of distributed database algorithms. Accordingly, the present system provides built in caching support for dynamically changing Internet content (e.g., Web pages which are modified on a regular basis). Such a result was not attainable with the same level of efficiency in prior art caching systems such as proxy server 210 of **Figure 2** (which are executed on, e.g., standard flat file systems such as UNIX or NFS file servers).

Data Services

Data services modules such as module 320 running on each ICDS node 410-440 provide support for data replication and distribution across ISP 400's internal network 480. This includes caching support for any data protocol included in the set of protocol modules 310-319 shown in **Figure 3** as well as for any future protocol which may be added as a module to the ICDS API 360. Because the ICDS API 360 provides a set of standardized service definitions for data services module 320, an ISP using a plurality of ICDS nodes 300 as illustrated in **Figure 4** can replicate data across its network without an extensive knowledge of distributed database technology. In other words, the ISP can configure its plurality of nodes by invoking the standardized service definitions associated with data services module 320 and leave the distributed database functionality to the distributed database engine 361.

Generally, three different types of data replication may be implemented by the present system: dynamic replication, database replication (or "actual" replication), and index replication. Using dynamic replication, if client 472 requests content from internal ICDS server 460 or from a server across network 490, the content will be delivered to client 472 and replicated in ICDS node 430. If client 473 (or any other client) subsequently requests the

same content, it will be transmitted directly from ICDS node 430 rather than from its original source (i.e., a second request to server 460 or a server across network 490 will not be required). Accordingly, bandwidth across ISP 400's internal network and across Internet link 405 is conserved.

The dynamic replication mechanism just described works well for replicating static content but not for replicating dynamically changing content. For example, if the replicated content is a magazine article then caching a copy locally works well because it is static information – i.e., there is no chance that the local copy will become stale (out of date). However, if the replicated content is a Web page which contains continually changing information such as a page containing stock market quotes, then dynamic replication may not be appropriate. No built in mechanism is available for proxy cache server 210 to store an up-to-date copy of the information locally.

The present ICDS system, however, may use database replication to maintain up-to-date content at each ICDS node 410-440. Because the present system includes a distributed database engine 361, when a particular piece of content is changed at one node (e.g., ICDS server 460) a store procedure may be defined to update all copies of the information across the network. This may be in the form of a relational database query. Thus, the present system may be configured to use dynamic replication for static content but to use database replication for time-sensitive, dynamically changing content.

The third type of database replication is known as index replication. Using index replication a master index of content is replicated at one or more ICDS nodes 410-440 across the network 480. Once again, this implementation is simplified by the fact that the underlying ICDS node engine is a distributed database engine. Certain types of information distributions are particularly suitable for using index replication. For example, news overview information (i.e., the list of news articles in a particular newsgroup) is particularly suited to index replication. Instead of replicating each individual article, only the news overview information needs to be replicated at various nodes 410-440 across the network 480. When a client 473 wants to view a particular article, only then will the article be retrieved and cached locally (e.g., on ICDS node 430).

ICDS node 430 is capable of caching and delivering various types of Internet data using any of the foregoing replication techniques. While prior art proxy servers such as proxy server 210 may only be used for caching Web pages, ICDS node 430 is capable of caching various other types of internet content (e.g., news content) as a result of the protocol modules 310-319 interfacing with ICDS API 360. Moreover, as stated above, ICDS node 430 (in conjunction with nodes 410, 420 and 440) may be configured to cache dynamic as well as static Web-based content using various distributed database algorithms.

One specific example of a data service provided by one embodiment of the present system is Wide Area Load Balancing (hereinafter "WALB") using layer 7 switching as described in the co-pending U.S. Patent Application entitled "WIDE AREA LOAD BALANCING" (Serial No. _____), which is assigned to the assignee of the present application and which is incorporated herein by reference. The present system may also perform dynamic protocol selection, dynamic query resolution, and/or heuristic adaptation for replicating content across a network as set forth in the co-pending U.S. Patent Application entitled Dynamic Protocol Selection and "QUERY RESOLUTION FOR CACHE SERVERS" (Serial No. ____), which is assigned to the assignee of the present application and which is incorporated herein by reference. Finally, the present system also may include network news (e.g., Usenet news) services set forth in the co-pending U.S. Patent Applications entitled "HYBRID NEWS SERVER" (Serial No. ___), and "SELF-MODERATED VIRTUAL COMMUNITIES" (Serial No. ___), each assigned to the assignee of the present application and each incorporated herein by reference.

Access Services

As stated above, prior art proxy server cache systems such as proxy server 210 are only capable of caching static, publicly available Web pages. A substantial amount of Web-based and non-Web-based content, however, requires some level of authentication before a user will be permitted to download it. Thus, client 472 (in Figure 4) may pay a service fee to obtain access to content on a particular web site (e.g., from server 460 or from another server over network 490). As a result, when he attempts to access content on the site he will initially be prompted to enter a user name and password. Once the user transmits this information to the Web server, he will then be permitted to download Web server content as per his service agreement.

A problem that arises, however, is that prior art cache systems such as proxy server 210 are not permitted to cache the requested content. This is because proxy server 210 has no way of authenticating subsequent users who may attempt to download the content. Thus, documents which require authentication are simply uncacheable using current network cache systems.

The present ICDS system, however, includes user authentication support embedded in access services module 322. Thus, when client 473, for example, attempts to access a Web page or other information which requires authentication, ICDS node will determine whether the requested content is stored locally. If it is, then ICDS node 430 may communicate with the authentication server (e.g., server 460 or any server that is capable of authenticating client 473's request) to determine whether client 473 should be granted access to the content. This may be accomplished using standardized authentication service definitions embedded in access services module 322. Using these definitions, ICDS node 430 will not only know what authentication server to use, it will also know what authentication *protocol* to use when it communicates to the authentication server. As a result of providing local access services module 322 for authentication, network information which requires authentication can now be cached locally in ICDS node 430, thereby conserving additional bandwidth across network link 405 and/or ISP network 480.

One particular embodiment of the present system replicates Remote Authentication Dial In User Service (hereinafter "RADIUS") information across network 480. RADIUS is an application-level protocol used by numerous ISP's to provide user authentication and profile services. This is achieved by setting up a central RADIUS server with a database of users, which provides both authentication services (i.e., verification of user name and password) and profile services detailing the type of service provided to the user (for example, SLIP, PPP, telnet, rlogin).

Users connect to one or more Network Access Servers (hereinafter "NASs") which operate as a RADIUS clients and communicate with the central RADIUS server. The NAS client passes the necessary user information to the central RADIUS server, and then acts on the response which is returned. RADIUS servers receive user connection requests, authenticate users, and then return all configuration information necessary for the client to deliver service to the user.

One problem associated with the RADIUS protocol is that it does not provide any built in facilities for replication of RADIUS information. Accordingly, on large ISP's such as America Online ("AOL"), which may have tens of millions of users, RADIUS servers are hard hit, potentially handling thousands of logon requests a minute. This may create severe performance/bandwidth problems during high traffic periods. In response, some ISP's have taken a brute-force approach to distributing RADIUS information by simply copying the information to additional servers across the network without any built in mechanism to keep the RADIUS data coherent and up-to-date.

One embodiment of the present ICDS system provides an efficient, dynamic mechanism for distributing RADIUS information. Specifically, a RADIUS module is configured to interface with ICDS API 360 in this embodiment (similar to the way in which protocol modules 310-319 interface with the ICDS API 360). RADIUS information can them be seamlessly distributed across the system using distributed database engine 361. For example, the RADIUS module in conjunction with access services module 322 on ICDS node 430 may maintain radius information for local users. [Exactly how will this work? I assume that access services module will be used but there will be a separate RADIUS protocol module to support the protocol??] Thus, when client 472 first logs in to the system, ICDS node 430 may communicate with a second ICDS node (e.g., central ICDS server 460) which contains the necessary RADIUS authentication and user profile information. Client 472 will input a user name and password and will then be permitted access to the network as per his service agreement with ISP 400.

Unlike previous RADIUS systems, however, ICDS node 430 in the present embodiment may locally cache client 472's RADIUS information so that the next time client 472 attempts to login to the network, the information will be readily available (i.e., no access to a second ICDS node will be necessary). ICDS node 430 may be configured to save client 472's RADIUS information locally for a predetermined period of time. For example, the information may be deleted if client 472 has not logged in to local ICDS node 430 for over a month.

Thus, if client 472 represents a user who frequently travels across the country and logs in to ISP 400's network 480 from various different ICDS nodes, the present system provides a quick, effective mechanism for dynamically replicating client 472's user information into

those geographical locations from which he most commonly accesses ISP 400. This reduces the load which would otherwise be borne by a central RADIUS server and also improves client 472's user experience significantly (i.e., by providing him with a quick login).

Database replication can also be used to update RADIUS information distributed across multiple ICDS nodes 410-440. This may be done using known store procedures defined in relational database 361. For example, if client 472 cancels his service agreement with ISP 400, he should not be able to continually log in to local ICDS node 430 using the RADIUS information which has been cached locally. Thus, under the present ICDS system, ISP 400 may simply issue a relational database query such as [let's add another update query here using database terminology as an example] to immediately update ICDS node 430's radius information.

One of ordinary skill in the art will readily recognize from the preceding discussion that alternative embodiments of the structures and methods illustrated herein may be employed without departing from the principles of the invention. Throughout the foregoing description, specific embodiments of the ICDS system were described using the RADIUS protocol in order to provide a thorough understanding of the operation of the ICDS system. It will be appreciated by one having ordinary skill in the art, however, that the present invention may be practiced without such specific details. For example, the ICDS system may also distribute authentication and user profile information in the form of the Lightweight Directory Access Protocol ("LDAP"). In other instances, well known software and hardware configurations/techniques have not been described in detail in order to avoid obscuring the subject matter of the present invention.

Access services module 322 may also provide local encryption/decryption and watermarking of internet content. Audio or video content delivery systems, for example, commonly use encryption of content to protect the rights of the underlying copyright holder. When a user requests a particular piece of content some delivery systems encrypt the content using a unique client encryption key. Only a client who possesses the encryption key (presumably the client who paid for the content) will be permitted to play the content back. Other systems provide for the "watermarking" of content (rather than encrypting) so that the rightful owner of the content may be identified. This simply entails embedding a unique "tag"

which identifies the source of the content and/or the owner of the content (i.e., the one who paid for it).

Prior art caching systems such as proxy server 210 are not capable of dealing with encrypted or watermarked content because the encryption/watermarking functionality was not provided locally (i.e., proxy server 210 was not "smart" enough). In one embodiment of the present ICDS system, however, access services module 322 of ICDS nodes 410-440 includes a local encryption module and/or a local watermarking module. For example, if client 473 requests specific content such as a copyrighted music content stored on a music server (e.g., ICDS server 460), the initial request for the content will be made from ICDS node 430 on behalf of client 473. ICDS node 430 will retrieve the requested content and cache it locally. If the requested content requires encryption, ICDS node 430 will use its local encryption module to encrypt the requested content using a unique user encryption key for client 473.

If a second client – e.g., client 472 – requests the same content, the copy stored locally on ICDS module 430 can be used. ICDS module 430 will simply encrypt the content using a *different* encryption key for user 472. Thus, frequently requested multimedia content (which, as is known in the art, can occupy a substantial amount of storage space) may be cached locally at ICDS node 430 notwithstanding the fact that the content requires both user authentication and encryption.

The same functionality may be provided for watermarking of content. ICDS node will use a watermarking module (which may comprise a component of access services module 322) to individually watermark multimedia information requested by individual clients, thereby protecting the rights of the copyright holder of the underlying multimedia content. This information can then be regularly communicated back to a central database repository.

As is known in the art, multimedia files can be extremely large and, accordingly, take substantially more time to communicate across a network than do, for example, generic Web pages. As such, the ability to locally cache multimedia files significantly reduces traffic across network 480, and also significantly improves the user experience for local users when downloading multimedia information.

Transaction Services

In addition to replicating data services and access services information across a network, the present ICDS system also provides for the replication of transaction services. Transaction services includes maintaining information on client payments for use of ISP 400's services as well as information relating to the client's online access profile (i.e., recording of the times when the user is online).

When a client logs in to an ISP today, the client's online information is maintained on a single central server. The central server maintains records of when and for how long the client logged in to the network and may also include information about what the client did while he was online. As was the case with maintaining a central RADIUS server, maintaining a central transaction server for all users of a large ISP is inefficient and cumbersome. The present system solves the performance and bandwidth problems associated with such a configuration by storing transaction information locally via transaction module 323 and algorithms build around distributed database engine 361.

Thus, if client 472 only logs on to ISP 400's network 480 via ICDS node 430, all of his transaction and billing information will be stored locally. The information may then be communicated across network 480 to a central billing server at predetermined periods of time (e.g., once a month). [We didn't go into great detail on transaction services and the rest; please add information as you feel appropriate]

Commercial Services

Commercial services module 324 provides a significantly improved local caching capability for add rotation and accounting. An add rotation system operating in conjunction with a typical proxy cache server will now be described with respect to **Figure 2.** When client 120 downloads a web page from Web server 142 the Web page may contain an ad tag or an add tag may automatically be inserted. The add tag will identify add server 170 and will indicate that an add should be inserted into the requested Web page from add server 170. Add server will then identify a specific add to insert into the downloaded Web page from add content server. The Web page plus the inserted add will then be forwarded to proxy server 210 and on to client 120.

Add server 170 will keep an accounting of how many different users have downloaded Web pages with adds inserted as described above. However, one problem with accounting on this system is that proxy server 210 requests Web pages *on behalf of* its clients. Accordingly, once the requested Web pages has been cached locally in Web cache 220, add server will only receive requests from proxy server 210 for any subsequent requests for the Web page. This will result in an inaccurate accounting of how many unique clients actually requested the Web page (and how many adds were viewed by unique users).

Because one embodiment of the present system provides built in caching support for ad rotation services, an accurate accounting of the number of hits that a particular ad receives may be maintained. More specifically, one embodiment of the present ICDS system solves this problem by providing a commercial services module that monitors and records how often individual clients request Web pages containing particular adds from add content server 171. This information than then be communicated to a central server (e.g., ICDS server 460) at predetermined intervals for generating add rotation usage reports.

Directory Services

Directory services provide the ability to cache locally a directory of information across network 480 or 490. That is, the question here is not whether the particular information is available but where exactly over networks 480 or 490 it is located. It should be noted that there may be some overlap between the directory services concept and the index replication concept described above with respect to data services. [I'm still not 100% sure what this is – please elaborate with an example]

II. Content Routing

The term "content routing" refers to the ability to select among various techniques/protocols for maintaining a coherent set of content across a network. The selection of a particular technique may be based on several routing variables including, but not limited to, the type of content involved (i.e., FTP, HTTP . . . etc), the size of the content involved (i.e., small files such as HTTP vs. large files such as audio/video streaming), the location of the content on network 480 and/or network 490, the importance of a particular piece of content (i.e., how important it is that the content be kept up-to-date across the entire network), the particular user requesting the content and the terms of his subscription agreement (i.e., some users may be willing to pay more to be insured that they receive only the most up-to-date

content without having to wait), the frequency with which the content is accessed (e.g., 5%-10% of content on the Internet represents 90% of all the *requested* content), and the underlying costs and bandwidth constraints associated with maintaining up-to-date, coherent content across a particular network (e.g., network 480).

Three content routing techniques which may be selected (based on one or more of the foregoing variables) to maintain coherent content across the plurality of nodes illustrated in **Figure 4** are content revalidation, content notification, and content synchronization.

Content Revalidation

When content validation is selected, the original content source will be checked only when the content is requested locally. For example, client 473 may request an installation program for a new Web browser (e.g., the latest version of Microsoft's™ Internet Explorer™). The file may then be transmitted form ICDS server 460 to client 473 and a copy of the file cached locally on ICDS node 430. Consequently, if client 472 requests the same program, for example, two weeks later, ICDS node 430 may be configured to check ICDS server 460 to ensure that it contains the most recent copy of the file before passing it on to client 472 (i.e., ICDS node 430 "revalidates" the copy it has locally).

ICDS node 430 may also be configured to revalidate a piece of content only if has been stored locally for a predetermined amount of time (e.g., 1 week). The particular length of time selected may be based on one or more of the variables discussed above. Moreover, in one embodiment, the age/revision of a particular piece of content is determined based on tags (e.g., HTML metatags) inserted in the particular content/file.

Revalidation may work more efficiently with certain types of content than with others. For example, revalidation may be an appropriate mechanism for maintaining up-to-date copies of larger files which do not change very frequently (i.e., such as the program installation files described above). However, revalidation may not work as efficiently for caching smaller and/or continually changing files (e.g., small HTML files) because the step of revalidating may be just as time consuming as making a direct request to ICDS server 460 for the file itself. If the file in question is relatively small and/or is changing on a minute-by-minute basis (e.g., an HTML file containing stock quotes) then one or more other content routing techniques may be more appropriate.

Of course, other routing variables may influence the decision on which technique to use, including the issue of how strong the data transmission connection is between ICDS node 430 and ICDS server 460 (i.e., how reliable it is, how much bandwidth is available . . . etc) and the necessity that the underlying information cached locally (at ICDS node 430) be accurate. The important thing to remember is that ICDS node 430 – because of its underlying open API architecture – may be configured based on the unique preferences of a particular client.

Content Notification

Content notification is a mechanism wherein the central repository for a particular piece of content maintains a list of nodes, or "subscribers," which cache a copy of the content locally. For example, in **Figure 4**, a plurality of agents may run on ICDS server 460 which maintain a list of content subscribers (e.g., ICDS node 430, ICDS node 420 . . . etc) for specific types of content (e.g., HTML, data streaming files, FTP files . . . etc). In one embodiment of the system, a different agent may be executed for each protocol supported by ICDS server 460 and/or ICDS nodes 410-440.

When a particular piece of content is modified on ICDS server 460, a notification of the modification may be sent to all subscriber nodes (i.e., nodes which subscribe to that particular content). Upon receiving the notification, the subscriber node – e.g., ICDS node 430 – may then invalidate the copy of the content which it is storing locally. Accordingly, the next time the content is requested by a client (e.g., client 472), ICDS node 430 will retrieve the up-to-date copy of the content from ICDS server 460. The new copy will then be maintained locally on ICDS node 430 until ICDS node 430 receives a second notification from an agent running on ICDS server 460 indicating that a new copy exists.

Alternatively, each time content is modified on ICDS server 460 the modified content may be sent to all subscriber nodes along with the notification. In this manner a local, up-to-date copy of the content is always ensured. In one embodiment of the system, notification and/or transmittal of the updated content by the various system agents is done after a predetermined period of time has elapsed (e.g., update twice a day). The time period may be selected based on the importance of having an up-to-date copy across all nodes on the network 480, 490.

As was the case with content revalidation, the different varieties of content notification may work more efficiently in some situations than in others. Accordingly, content notification may be selected as a protocol (or not selected) based on one or more of the routing variables recited at the beginning of this section (i.e., the "content routing" section). For example, content notification may be an appropriate technique for content which is frequently requested at the various nodes across networks 480 and 490 (e.g., for the 5-10% of the content which is requested 90% of the time), but may be a less practical technique for larger amount of content which is requested infrequently. As another example, large files which change frequently may not be well suited for content notification (i.e., particularly the type of content notification where the actual file is sent to all subscribers along with the notification) due to bandwidth constraints across networks 480 and/or 490 (i.e., the continuous transmission of large, frequently changing files may create too much additional network traffic).

Content Synchronization

Content synchronization is a technique for maintaining an exact copy of a particular type of content on all nodes on which it is stored. Using content synchronization, as soon as a particular piece of content is modified at, for example, ICDS node 430, it will immediately be updated at all other nodes across networks 480 and/or 490. If the same piece of data was concurrently modified at one of its other storage locations (e.g., ICDS node 410) then the changes may be backed off in order to maintain data coherency. Alternatively, an attempt may be made to reconcile the two separate modifications if it is possible to do so (using, e.g., various data coherency techniques).

Once again, as with content notification and content revalidation, content synchronization is more suitable for some situations than it is for others. For example, content synchronization is particularly useful for information which can be modified from several different network nodes (by contrast, the typical content notification paradigm assumes that the content is modified at one central node). Moreover, content synchronization may be useful for maintaining content across a network which it is particularly important to keep current. For example, if network 480 is an automatic teller machine (hereinafter "ATM") network, then when a user withdraws cash from a first node (e.g., ICDS node 440), his account will be instantly updated on all nodes (e.g., ICDS node 410, 420, 460, and 430) to reflect the withdrawal. Accordingly, the user would not be able to go to a different node in a different part of the country and withdraw more than what he actually has in his account.

As another example, a user's account status on a network (i.e., whether he is a current subscriber and/or what his network privileges are) may be maintained using content synchronization. If, for example, a user of network 480 were arrested for breaking the law over network 480 (e.g., distributing child pornography), it would be important to disable his user account on all network nodes on which this information might be cached. Accordingly, using content synchronization, once his account was disabled at one node on network 480 this change would automatically be reflected across all nodes on the network.

As previously stated, the choice of which content routing technique to use for a particular type of content may be based on any of the variables set forth above. In one embodiment, the frequency with which content is accessed across the networks 480 and/or 490 may be an important factor in deciding which protocol to use. For example, the top 1% accessed content may be selected for content synchronization, the top 2%-10% accessed content may be selected for content notification, and the remaining content across networks 480, 490 may be selected for content revaildation.

III. Content Delivery Medium Selection

In addition to the content routing flexibility provided by the content delivery system as set forth above, one embodiment of the system allows content delivery nodes such as ICDS node 430 to select from a plurality of different transmission media. For example, ICDS node 430 may receive content from ICDS server 460 via a plurality of communication media, including, but not limited to, satellite transmission, wireless RF transmission, and terrestrial transmission (e.g., fiber).

Moreover, as with the selection of a particular content routing technique, the selection of a particular transmission medium may be based on any of the variables set forth above (see, e.g., routing variables listed under counter routing heading; page 24, line 18 through page 25, line 9). Moreover, the choice of a particular transmission medium may be dynamically adjustable based on performance of that medium. For example, ICDS node 430 may be configured to receive all of its content over terrestrial network 480 as long as network 480 is transmitting content at or above a threshold bandwidth. When transmissions over network 480 dip below the threshold bandwidth, ICDS node 480 may then begin receiving certain content via satellite broadcast or wireless communication.

In addition, a transmission medium may be selected for transmitting specific content based on how frequently that content is accessed. For example, the top 10% frequently accessed content may be continually pushed out to ICDS node 430 via satellite broadcast while the remaining content may be retrieved by (i.e., "pulled" to) ICDS node 430 over network 480 upon request by clients (e.g., client 473). Accordingly, those employing ICDS nodes such as node 430 can run a cost-benefit analysis to determine the most cost effective way to implement their system by taking in to consideration, for example, the needs of their users, the importance of the content involved and the expense of maintaining multiple transmission connections into ICDS node 430 (e.g., the cost associated with maintaining an ongoing satellite connection).

In one embodiment of the system, tags (e.g., HTML metatags) may be inserted into particular types of content to identify a specific transmission path/medium for delivering that content to ICDS node 480. The tags in this embodiment may identify to various nodes (and/or routers) across networks 480 and/or 490 how the particular content should be routed across the networks (e.g., from node 410 to node 420 via terrestrial network 480; from node 420 to node 430 via wireless transmission).

One of ordinary skill in the art will readily recognize from the preceding discussion that alternative embodiments of the structures and methods illustrated herein may be employed without departing from the principles of the invention. Throughout this detailed description, numerous specific details are set forth such as specific network protocols (i.e., RADIUS) and networks (i.e., the Internet) in order to provide a thorough understanding of the present invention. It will be appreciated by one having ordinary skill in the art, however, that the present invention may be practiced without such specific details. In other instances, well known software and hardware configurations/techniques have not been described in detail in order to avoid obscuring the subject matter of the present invention. The invention should, therefore, be measured in terms of the claims which follow.

CLAIMS

What is claimed is:

1. A network content delivery system configured to:

select a first content routing technique for processing a first set of network content; and

select a second content routing technique for processing a second set of network content, wherein said first and second content routing techniques are selected based on one or more content routing variables.

2. The network content delivery system as claimed in Claim 1 wherein one of said selected content routing techniques is a content revalidation technique.

3. The network content delivery system as claimed in Claim 1 wherein one of said selected content routing techniques is a content notification technique.

4. The network content delivery system as claimed in Claim 1 wherein one of said selected content routing techniques is a content synchronization technique.

5. The network content delivery system as claimed in Claim 1 wherein one of said content routing variables is the frequency with which said network content is accessed by users.

6. The network content delivery system as claimed in Claim 1 wherein one of said content routing variables is the size of said network content.

7. The network content delivery system as claimed in Claim 1 wherein one of said content routing variables is the frequency with which said network content is modified.

8. The network content delivery system as claimed in Claim 1 wherein one of said content routing variables is the type of network content (e.g., HTML, Usenet News).

9. The network content delivery system as claimed in Claim 1 wherein one of said content routing variables is identity of the user requesting said network content.

10. The network content delivery system as claimed in Claim 2 wherein said content revalidation technique is selected based on the size of said network content.

11. The network content delivery system as claimed in Claim 2 wherein said content revalidation technique is selected based on the frequency with which said network content is accessed.

12. The network content delivery system as claimed in Claim 3 wherein said content notification technique is selected based on the size of said network content.

13. The network content delivery system as claimed in Claim 3 wherein said content notification technique is selected based on the frequency with which said network content is accessed.

14. The network content delivery system as claimed in Claim 4 wherein said content synchronization technique is selected based on the size of said network content.

15. The network content delivery system as claimed in Claim 4 wherein said content synchronization technique is selected based on the frequency with which said network content is accessed.

16. The network content delivery system as claimed in Claim 1 including the additional step of selecting a first transmission medium for a first group of network content based on one or more of said content routing variables.

17. The network content delivery system as claimed in Claim 16 including the additional step of selecting a second transmission medium for a second group of network content based on one or more of said content routing variables.

18. The network content delivery system as claimed in Claim 1 including an application programming interface for interfacing with a plurality of network protocol and service modules.

19. A content delivery system comprising:

a network node for storing network content;

a first transmission medium communicatively coupled to said network node for transmitting a first set of network content to said network node; and

a second transmission medium communicatively coupled to said network node for transmitting a second set of network content to said network node,

wherein said first and second sets of network content are selected based on one or more routing variables.

20. The content delivery system as claimed in Claim 19 wherein said first transmission medium is a satellite transmission.

21. The content delivery system as claimed in Claim 19 wherein said first transmission medium is a wireless radio frequency transmission.

22. The content delivery system as claimed in Claim 19 wherein said first transmission medium is terrestrial-based transmission.

23. The content delivery system as claimed in Claim 19 wherein said network node monitors transmission bandwidth of said first transmission medium and reallocates content

from said first set to said second set if said first transmission medium drops below a predetermined threshold value.

24. The content delivery system as claimed in Claim 23 wherein said first transmission medium is terrestrial and said second transmission medium is non-terrestrial.

25. The content delivery system as claimed in Claim 19 wherein content is included in said first set based on the frequency with which said content is accessed.

26. The network content delivery system as claimed in Claim 19 including an application programming interface for interfacing with a plurality of network protocol and service modules.

27. An article of manufacture including a sequence of instructions stored on a computer-readable media which, when executed by a network node, cause the network node to perform the acts of:

establishing a plurality of groups of network content to be cached on said network node based on one or more content routing variables;

selecting a first content routing technique for maintaining data coherency in a first group of said plurality; and

selecting a second content routing technique for maintaining data coherency in a second group of said plurality.

28. The article of manufacture as claimed in claim 27 wherein said first content routing technique is content revalidation.

29. The article of manufacture as claimed in Claim 28 wherein said second content routing technique is content notification.

30. The article of manufacture as claimed in Claim 28 wherein said second content routing technique is content synchronization.

31. The article of manufacture as claimed in Claim 28 wherein said content routing variable used to select said content for said first group is the frequency with which said content is accessed.

32. The article of manufacture as claimed in Claim 29 wherein said content routing variable used to select said content for said first group is the frequency with which said content is accessed.

33. The article of manufacture as claimed in Claim 30 wherein said content routing variable used to select said content for said first group is the frequency with which said content is accessed.

34. A network node comprising:

an application programming interface ("API"), said API including a distributed relational database engine;

a plurality of protocol modules for interfacing with said API, said protocol modules configured to allow said system to communicate over a network using a plurality of network protocols;

a cache memory for caching data communicated to said cache memory using said plurality of protocol modules; and

a data services module for maintaining coherency between said data stored in said cache memory and data stored at other nodes across said network.

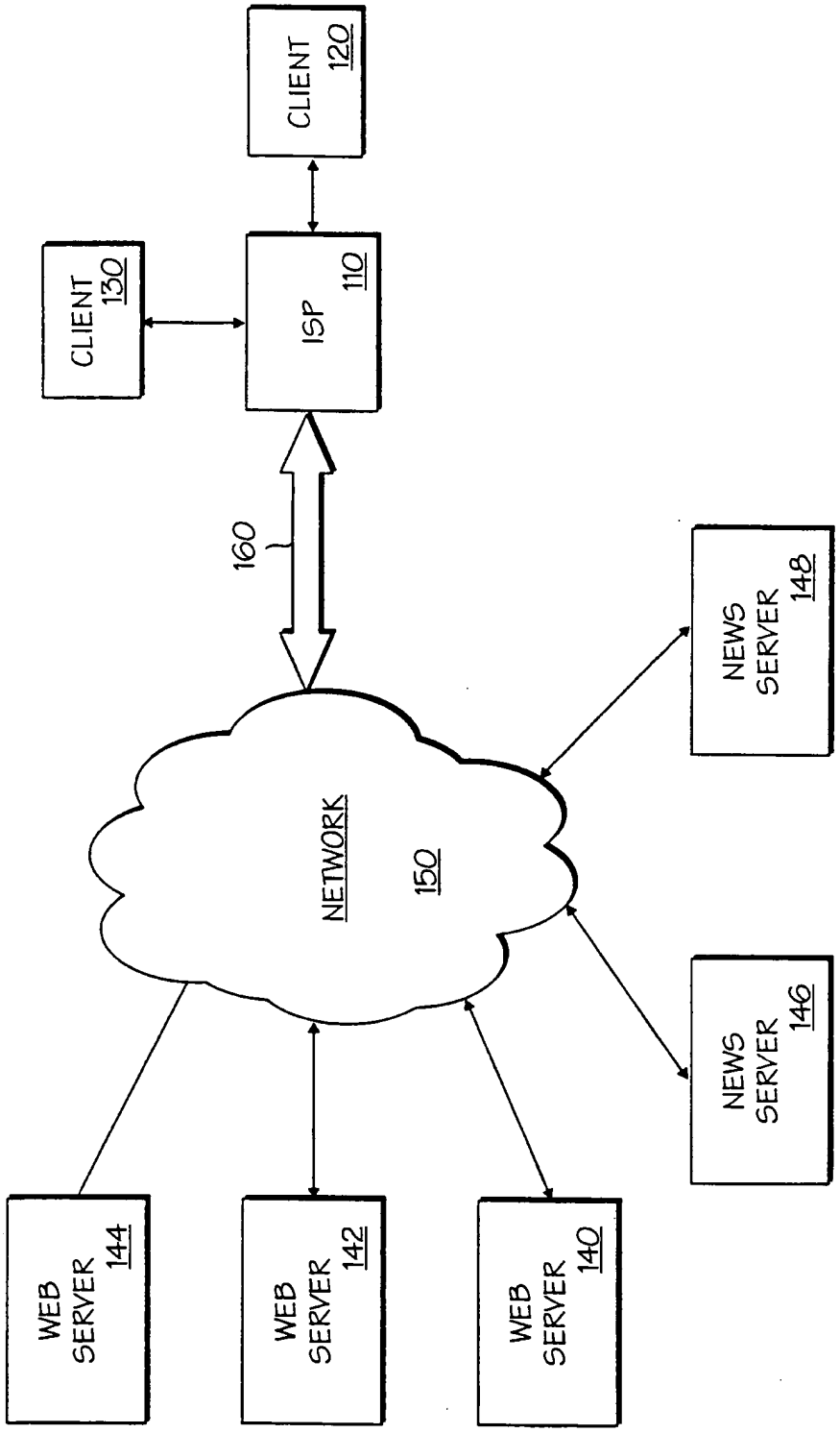

CLIENT 120
CLIENT 130
ISP 110
160
NEWS SERVER 148
NETWORK 150
NEWS SERVER 146
WEB SERVER 144
WEB SERVER 142
WEB SERVER 140

FIG. 1

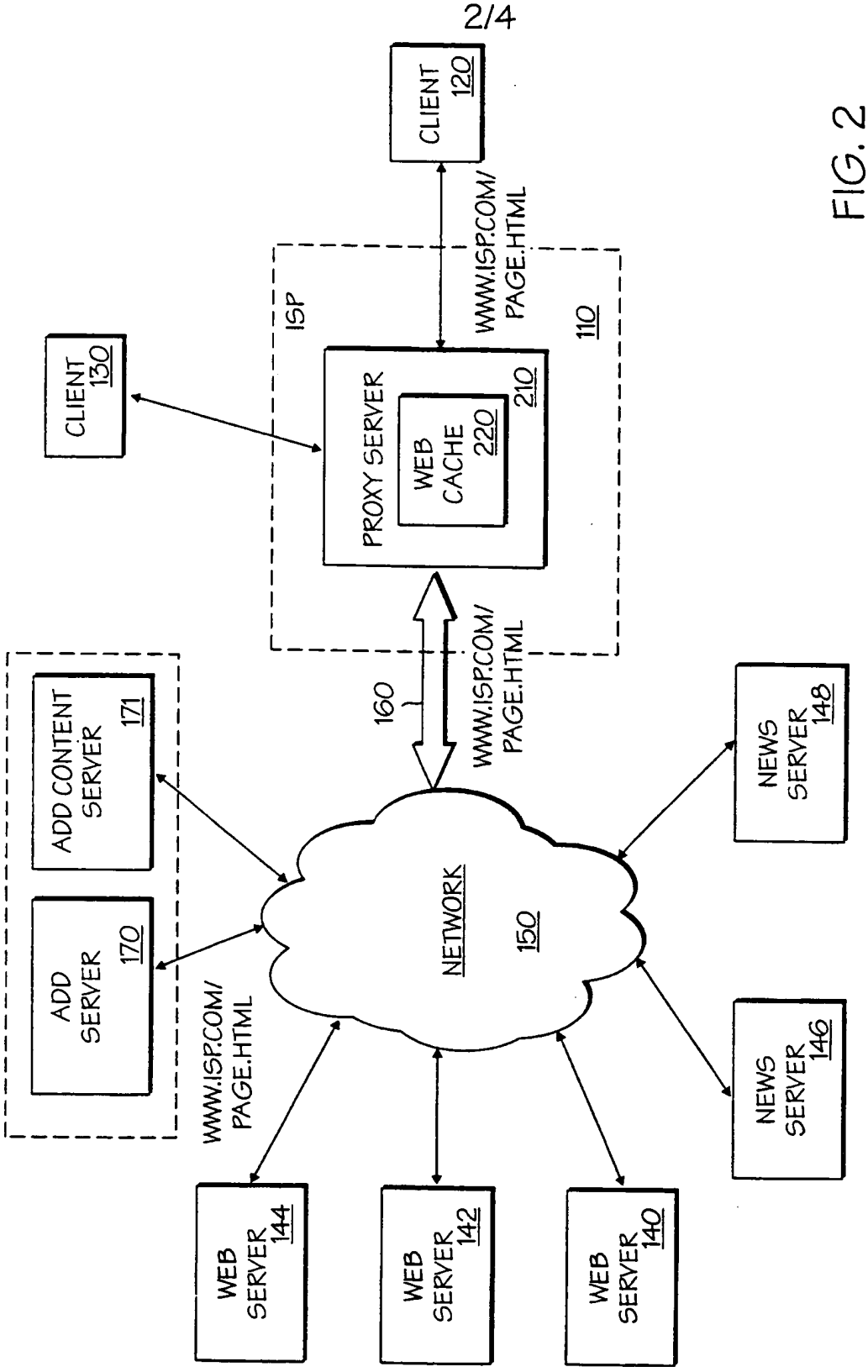

CLIENT 120
WWW.ISP.COM/ PAGE.HTML
ISP
110
CLIENT 130
PROXY SERVER 210
WEB CACHE 220
160
WWW.ISP.COM/ PAGE.HTML
ADD CONTENT SERVER 171
ADD SERVER 170
WWW.ISP.COM/ PAGE.HTML
NETWORK 150
NEWS SERVER 148
NEWS SERVER 146
WEB SERVER 144
WEB SERVER 142
WEB SERVER 140

FIG. 2

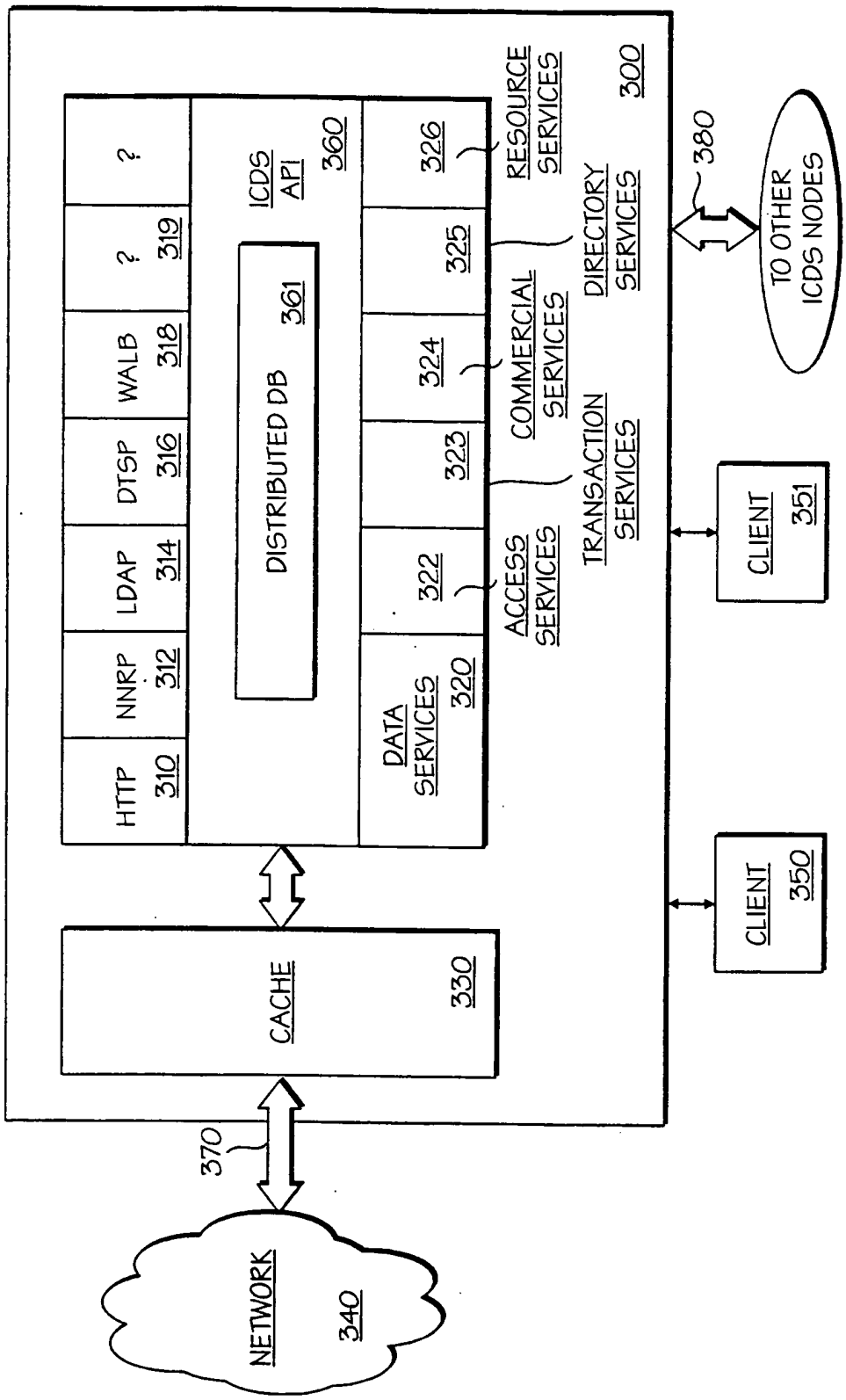

HTTP 310
NNRP 312
LDAP 314
DTSP 316
WALB 318
? 319
?
ICDS API 360
DISTRIBUTED DB 361
DATA SERVICES 320
322
323
324
325
326
ACCESS SERVICES
TRANSACTION SERVICES
COMMERCIAL SERVICES
DIRECTORY SERVICES
RESOURCE SERVICES
300
CACHE 330
370
NETWORK 340
CLIENT 350
CLIENT 351
380
TO OTHER ICDS NODES

FIG. 3

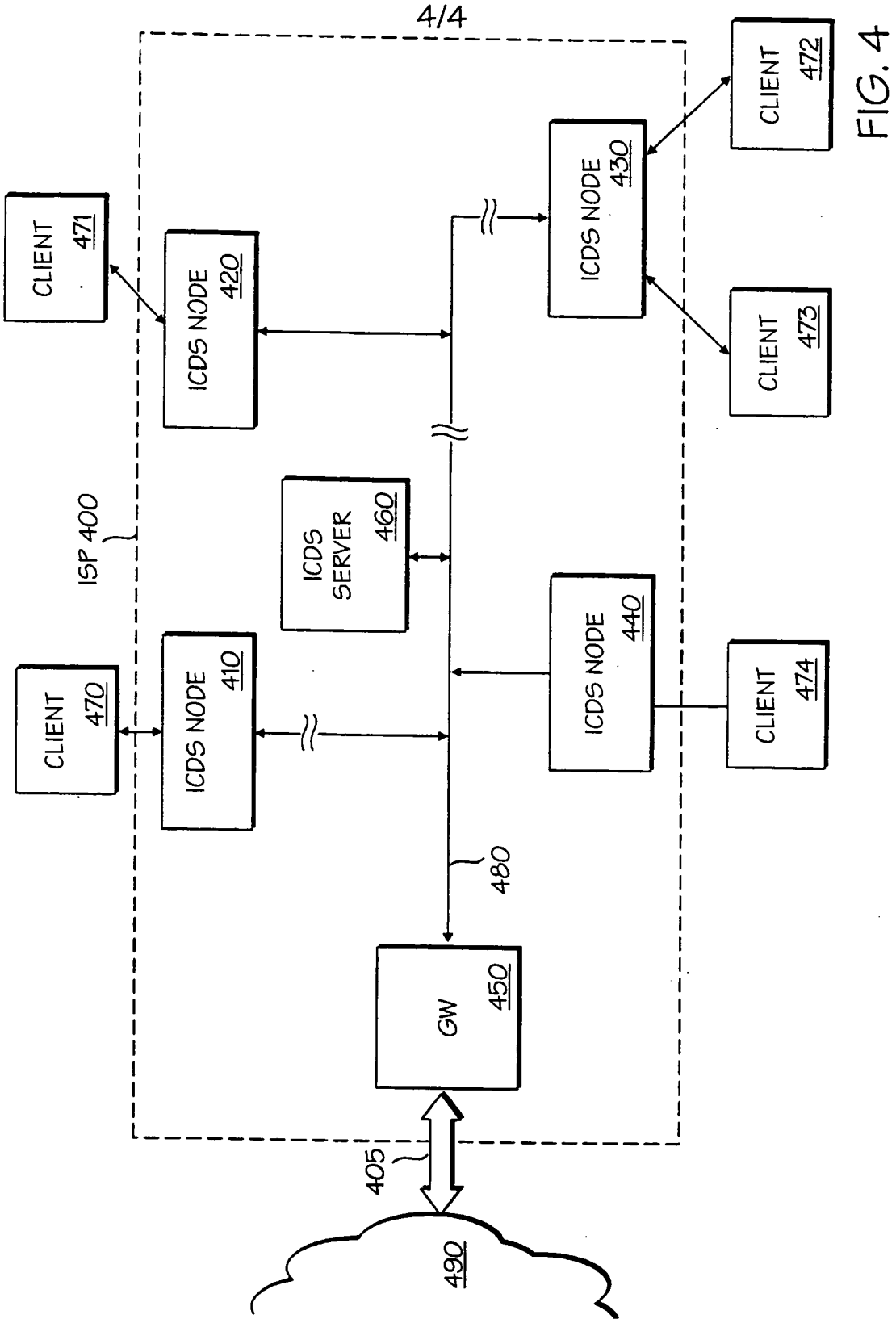

ISP 400
CLIENT 471
ICDS NODE 420
ICDS NODE 430
CLIENT 472
CLIENT 473
ICDS SERVER 460
CLIENT 470
ICDS NODE 410
ICDS NODE 440
CLIENT 474
480
GW 450
405
490

FIG. 4

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
25 May 2001 (25.05.2001)

PCT

(10) International Publication Number
WO 01/37209 A1

(51) International Patent Classification[7]: G06K 9/36, 9/46

(21) International Application Number: PCT/US00/30825

(22) International Filing Date:
8 November 2000 (08.11.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/438,666 12 November 1999 (12.11.1999) US

(71) Applicant: TERALOGIC, INC. [US/US]; 1240 Villa Street, Mountain View, CA 94041 (US).

(72) Inventors: CASTOR, Jon, S.; 2160 Stockbridge Avenue, Woodside, CA 94062 (US). CHUI, Charles, K.; 340 Olive Street, Menlo Park, CA 94025 (US).

(74) Agents: WILLIAMS, Gary, S. et al.; Pennie & Edmonds LLP, 1155 Avenue of the Americas, New York, NY 10036 (US).

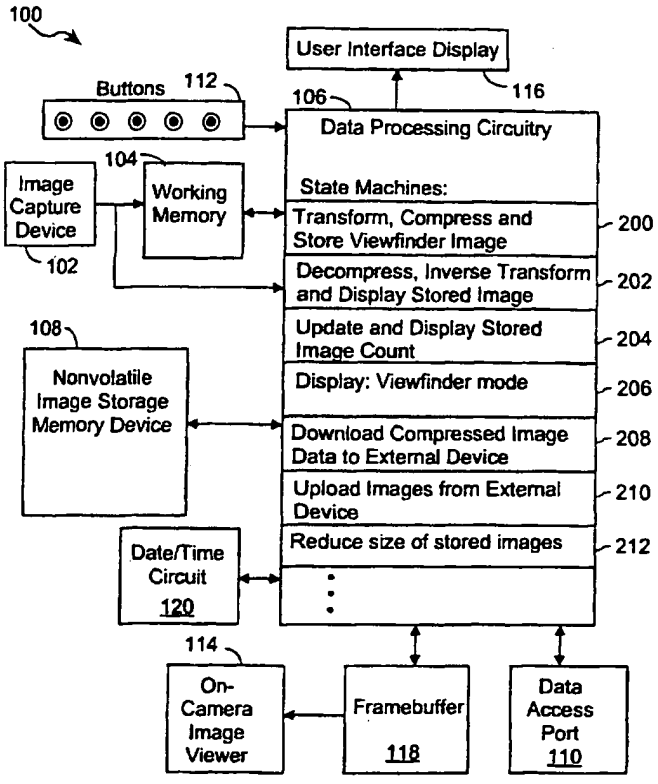
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *With international search report.*

[Continued on next page]

(54) Title: PICTURE AND VIDEO STORAGE MANAGEMENT SYSTEM AND METHOD



(57) Abstract: An image processing system (100) stores image files in a memory device (108) at a number of incremental quality levels. Each image file has an associated image quality (that is fidelity or resolution) level corresponding to a quality level at which the corresponding image has been encoded. The images are initially encoded by applying a predefined transform, such as a DCT transform or wavelet-like transform (200), to image data received from an image capture mechanism (102) and then applying a data compression method to the transform data (200). The image is regenerated by successively applying a data decompression method and an inverse transform to an image file (202). Image file size reduction circuitry (212) and one or more state machines are used to lower the quality level of a specified one of the image files, including circuitry for extracting a subset of the data in the specified image file and forming a lower quality version of the specified image file that occupies less space in the memory device than was previously occupied by the specified image data structure. As a result, the amount of space occupied by image files in the memory device can be reduced so as to make room for the storage of additional image files or to allow more rapid transmission in a restricted bandwidth environment.

WO 01/37209 A1

— *Before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments.*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

PICTURE AND VIDEO STORAGE MANAGEMENT SYSTEM AND METHOD

The present invention relates generally to the storage of still and video images in an image processing device or system, such as a digital camera or digital video camera or a computer based image storage system, and particularly to a system and method for storing images at different image quality levels and for enabling users to dynamically compress high quality images into lower quality images in order to free up storage for the storage of additional images.

BACKGROUND OF THE INVENTION

Digital cameras typically include either permanent internal storage, and/or removable storage, for storing images. For instance, many digital cameras use removable flash memory cards for storing images. Each flash memory card has a fixed amount of memory, and when the memory card is full it can be removed from the camera and replaced by another flash memory card. In addition, digital cameras typically have a built-in viewer that enables the user to review the images stored on the flash memory card (and/or in the internal memory) and to delete user specified ones of the stored images. Deleting stored images obviously creates room for storing additional images.

When a digital camera user is "in the field" he/she generally has a limited amount of image storage on hand. If all the available image storage is full, the user has the choice of either not taking any additional pictures, or of deleting pictures from the image storage devices on hand to make room for new images. While this is actually one level better than the situation with film cameras, in which the user is simply out of luck when all the available film has been used, it is the premise of the present invention that the current image storage limitations of digital cameras are caused, in part, by failure to fully exploit the advantages of having images stored in digital format.

Similar storage vs. image quality considerations also apply to digitally encoded video frames. In particular, for any given amount of storage space, such as in a digital video camera, the goal

is to retain the best image quality for the amount of storage required for a given number of video frames. Current devices allow the user to select image quality prior to capturing a digital video image, but do not enable the user to effectively manage the storage space in the video camera with respect to video sequences already taken, other than by deletion.

It is an object of the present invention to provide a digital camera or digital video camera, or other constrained storage device, that can store images at a plurality of image quality (i.e., fidelity or resolution) levels and furthermore can reduce images initially stored at a first image quality level to a lower image quality level so as to reduce the amount of storage occupied by those images.

It is also an object of the present invention to provide space efficient and computationally efficient image and video handling mechanisms for other applications, including network connected image and video libraries, Internet web browsers executed by client computers coupled to server computers, cable television set top boxes having video storage capabilities, and so on.

SUMMARY OF THE INVENTION

In summary, the present invention is an image processing device or system, such as a digital camera or digital video camera or a computer based image storage system, that can store images at a number of different image quality levels. The image processing device includes a memory device and image management logic. The memory device stores image files that each represent a respective image, each image file having an associated image quality level corresponding to a quality level at which the corresponding image has been encoded. The image management logic includes data processing circuitry and state machines for storing and processing image data received from an image capture mechanism. More specifically, the image management logic includes image processing circuitry and one or more state machines for applying a predefined transform, such as a wavelet-like transform, to image data received from the image capture mechanism to generate transform image data and for applying a data compression method to the transform image data so as to generate an image file having an associated image quality level.

The image management logic also includes image reconstruction circuitry and one or more state machines for successively applying a data decompression method and an inverse transform to a specified one of the image files so as to generate a reconstructed image suitable for display on an image viewer.

Further, the image management logic includes image file size reduction circuitry and one or more state machines for reducing the size of an image file while minimizing the reduction in image quality level. This circuitry extracts a subset of the data in the specified image file and forms a lower quality version of the specified image file that occupies less space in the memory device than was previously occupied by the specified image data structure. As a result, the amount of space occupied by image files in the memory device can be reduced so as to make room for the storage of additional image files or to allow more rapid transmission in a restricted bandwidth environment.

In a preferred embodiment, the image transform data in an image file is organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes. To generate a lower quality image file, the image size reduction circuitry extracts a portion of the image file that excludes the image transform data for at least one bit plane and replaces the image file with an image file containing the extracted portion. Further, the image data is also organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers. The image size reduction circuitry can also generate a lower quality image file by extracting a portion of the image file that excludes the image transform data for at least one transform layer and replaces the image file with an image file containing the extracted portion.

BRIEF DESCRIPTION OF THE DRAWINGS

Additional objects and features of the invention will be more readily apparent from the following detailed description and appended claims when taken in conjunction with the drawings, in which:

Fig. 1 is a block diagram of a digital camera in accordance with an embodiment of the present invention.

Fig. 2 depicts an image data array divided into a set of smaller analysis arrays for purposes of encoding and data compression.

Fig. 3 schematically depicts the process of transforming a raw image data array into a transform image array and compressing the transform image array into a compressed image file.

Figs. 4A and 4B depict image storage data structures.

Fig. 5 is a conceptual flow chart depicting changes in the state of a digital camera as various operations are performed.

Fig. 6 is a conceptual flow chart depicting changes in the state of a video image sequence as the video image sequence is encoded and compressed.

Figs. 7, 8 and 9 show data structures used in one particular embodiment for video image sequence encoding and compression.

Fig. 10 is a conceptual diagram of an Internet server and client devices that utilize the image or and video image compressing and management features of the present invention.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

In some image processing systems, an image can be stored at a number of discrete resolution levels, typically with each resolution level differing from its "neighbors" by a resolution factor of four. In other words, if the highest resolution representation (at resolution level 1) of the image contains X amount of information, the second resolution level representation contains (for example) X/4 amount of information, the third resolution level representation contains X/16 amount of information, and so on. Thus, an image's "resolution" typically means the

amount of image information, and in the context of digital image processing systems is often expressed in terms of the number of distinct pixel elements. The number of resolution levels and the particular amount of information reduction from one level to the next may vary considerably from one system to another. Further, the present invention would be equally applicable to systems having a continuous range of resolution levels.

Another concept concerning image quality is "fidelity." The fidelity of an image can be compromised even if its resolution, in terms of the number of pixels in the image, is unchanged. An image's fidelity can be reduced by reducing the number of bits used to represent the image data. For instance, if the transform coefficients used to represent an image are represented at full fidelity using 12 bits, and then the number of bits used to represent transform coefficients is reduced to 11, the fidelity of the resulting image will be reduced. In other words, the quality of the image reconstructed from the lower fidelity data will be a little less sharp than an image reconstructed from higher fidelity data.

In this document, the terms "image quality" and "quality level" will be used in the general sense of image quality, encompassing both image "resolution" and image "fidelity." Thus, a reduction in an image's "image quality" from a top image quality level to a next highest image quality level might be accomplished either by reducing the image's resolution or by reducing its fidelity, or both. In other embodiments the terms "image quality" and "quality level" may be used to refer to other aspects of image quality as well.

Digital Camera Architecture

Referring to Fig. 1, there is shown an embodiment of a digital camera system 100 in accordance with the present invention. The digital camera system 100 includes an image capture device 102, such as a CCD or CMOS sensor array or any other mechanism suitable for capturing an image as an array of digitally encoded information. Thus the image capture device is assumed to include analog to digital conversion (ADC) circuitry for converting analog image information into digital values.

A working memory 104, typically random access memory, receives digitally encoded image information from the image capture device 102. More generally, it is used to store a digitally

encoded image while the image is being transformed and compressed and otherwise processed by the camera's data processing circuitry 106. Memory 104 may be integrated on the same integrated circuit as other devices, or may be implemented using separate circuit(s).

The data processing circuitry 106 in one embodiment consists of hardwired logic and a set of state machines for performing a set of predefined image processing operations. In alternate embodiments the data processing circuitry 106 could be implemented in part or entirely using a fast general purpose microprocessor and a set of software procedures. However, at least using the technology available in 1999, it would be difficult to process and store full resolution images (e.g., full color images having 1280 x 840 pixels) fast enough to enable the camera to be able to take, say, twenty pictures per second, which is a requirement for some commercial cameras, as well as digital video cameras. In the future, general purpose microprocessors or general purpose image data microprocessors (e.g., single instruction multiple data (SIMD) processors) may be able to provide the fast image processing needed by digital cameras, in which case the data processing circuit 106 could be implemented using such a general purpose microprocessor or perhaps a hybrid processor system.

Each image, after it has been processed by the data processing circuitry 106, is typically stored as an "image file" in a nonvolatile memory storage device 108, typically implemented using "flash" (i.e., EEPROM) memory technology. The nonvolatile memory storage device 108 is preferably implemented as a removable memory card. This allows the camera's user to remove one memory card, plug in another, and then take additional pictures. However, in some implementations, the nonvolatile memory storage device 108 may not be removable, in which case the camera will typically have a data access port 110 to enable the camera to transfer image files to and from other devices, such as general purpose, desktop computers, computer systems and devices used to warehouse libraries of images, computer systems and devices used to store and distribute image files, and so on. Digital cameras with removable nonvolatile memory 108 may also include a data access port 110.

While the amount of storage in the nonvolatile image memory 108 will vary from one implementation to the next, such devices will typically have sufficient capacity to store 10 to 50 high quality images. Once the nonvolatile image memory 108 is full, if the file size reduction methodology of the present invention is not used, the only way the camera can be

used to take additional pictures is either by deleting images from the nonvolatile image memory 108 or by replacing the nonvolatile image memory 108 with another one. If neither of these options are feasible (e.g., because the user has filled all the memory cards he/she has on hand with images that he/she does not wish to delete), then no further pictures can be taken until a new memory device 108 is inserted or the stored images are transferred to an external device (if available).

The digital camera 100 includes a set of buttons 112 for giving commands to the camera. In addition to the image capture button, there will typically be several other buttons to enable the use to select the quality level of the next picture to be taken, to scroll through the images in memory for viewing on the camera's image viewer 114, to delete images from the nonvolatile image memory 108, and to invoke all the camera's other functions. Such other functions might include enabling the use of a flash light source, and transferring image files to and from a computer. In accordance with the present invention, the user selectable functions, selected by using the buttons 112, further include reducing the size of one or more of the image files stored in the nonvolatile image memory 108 so as to make room for the storage of additional images. The buttons in one embodiment are electromechanical contact switches, but in other embodiments at least some of the buttons may be implemented as touch screen buttons on a user interface display 116, or on the image viewer 114.

The user interface display 116 is typically implemented either (A) as an LCD display device separate from the image viewer 114, or (B) as images displayed on the image viewer 114. Menus, user prompts, and information about the images stored in the nonvolatile image memory 108 may be displayed on the user interface display 116, regardless of how that display is implemented.

After an image has been captured, processed and stored in nonvolatile image memory 108, the associated image file may be retrieved from the memory 108 for viewing on the image viewer. More specifically, the image file is converted from its transformed, compressed form back into a data array suitable for storage in a framebuffer 118. The image data in the framebuffer is displayed on the image viewer 114. A date/time circuit 120 is used to keep track of the current date and time, and each stored image is typically date stamped with the date and time that the image was taken.

Image Data Structures

Referring to Fig. 2, in one embodiment the nonvolatile image memory 108 stores a directory 130 that lists all the image files 132 stored in the memory 108. Preferably, the directory 130 contains information for each stored image file 132, such as the date and time the image was taken, the quality level of the image and the file's location in the memory 108.

To understand the image data structure stored in each image file, it is helpful to first understand how an image file is encoded. Referring to Fig. 3, a raw image data array 140, obtained from the digital camera's image capture mechanism 102 (Fig. 1), is treated as a set of non-overlapping "analysis arrays" 142 of a fixed size, such as 32 x 32, or 64 x 64 (or more generally $2^n \times 2^n$, for some integer value of n). A sufficient number of subarrays are used to cover the entire data array that is to be encoded, even if some of the subarrays overhang the edges of the data array. The overhanging portions of the subarrays are filled with zero data values during the data encoding process. In a preferred embodiment, the origin of the data array is the top left corner, the first coordinate used to identify data array positions is the "Y" axis or vertical coordinate, and the second coordinate used is the "X" axis or horizontal coordinate. Thus, a position of 0,64 indicates a pixel at the top vertical position of the array, 64 pixel positions over to the right from the array origin, while a position of 32,0 indicates a pixel on the left edge of the array, 32 pixel positions vertically down from the array origin.

An appropriate transform is applied to each of the analysis arrays 142, and then the resulting transform coefficients for each analysis array are quantized (e.g., divided by an appropriate value to generate integer valued, quantized transform coefficients) so as to generate a transformed image array 144. In one embodiment the transform applied to the raw image data is a wavelet-like transform. In other embodiments a DCT transform could be used (which is the type of transform used in current JPEG image encoding systems), or other types of wavelet or wavelet-like transforms could be used.

In this document, the terms "wavelet" and "wavelet-like" are used interchangeably. Wavelet-like transforms generally have spatial frequence characteristics similar to those of conventional wavelet transforms, and are losslessly reversible, but have shorter filters that are more computationally efficient.

In one embodiment the transformed image array 144 is generated by successive applications of a wavelet-like decomposition transform. A first application of the wavelet-like decomposition transform to an initial two dimensional array of "raw" image data generates four sets of coefficients, labeled LL, HL1, LH1 and HH1. Each succeeding application of the wavelet-like decomposition transform is applied only to the LL set of coefficients generated by the previous wavelet transformation step and generates four new sets of coefficients, labeled LL, HLx, LHx and HHx, where x represents the wavelet transform "layer" or iteration. After the last wavelet-like decomposition transform iteration only one LL set remains. The total number of coefficients generated is equal to the number of data samples in the original data array. The different sets of coefficients generated by each transform iteration are sometimes called layers. The number of wavelet transform layers generated for an image is typically a function of the resolution of the initial image. Performing five to seven wavelet transformation layers is fairly typical, but more or less may be used depending on such considerations as the size of the analysis arrays, the subject matter of the image, the data processing resources available for image compression, and the like.

For the purposes of explaining the operation of the image encoding and decoding operations of the present invention, the specific type of image transform used and the specific type of data quantization used to transform a raw image file 140 into a transformed image array 142 are not relevant and therefore are not further described herein. However, a preferred embodiment of the wavelet transform and data quantization methods are described in U.S. Patent No. 5,909518, "System and Method for Performing Wavelet and Inverse Wavelet Like Transformations of Digital Data," which is hereby incorporated by reference as background information.

Each transformed image array 144 is compressed and encoded using a sparse data encoding technique. In one embodiment, the method of compressing and encoding the analysis arrays is the method described in detail in U.S. patent application 08/858,035, filed May 16, 1997, entitled "System and Method for Scalable Coding of Sparse Data Sets," now U.S. Patent No. 5,949,911, which is hereby incorporated by reference as background information. The encoded image data for all the analysis arrays of the image are combined and stored as an image file 132.

Referring to Fig. 4A, the image file 132 includes header data 160 and a sequence of data structures 162, each representing one analysis array. The header data 160 indicates the size of the image file and the image file's quality level. The header data also includes a list of analysis array size values indicating the length of each of the analysis array data structures 162, thereby enabling fast indexing into the image data. Storing size values for the analysis arrays enables the camera's data processing circuitry 106 (Fig. 1) to locate the beginning of any analysis array data structure 162 without having to decode the contents of the earlier analysis arrays in the image file 132.

As shown in Fig. 4B, the encoded data 162 representing any one analysis array is stored in "bit layer order". For each analysis array, the encoding procedure determines the most significant non-zero bit in the data to be encoded, which is herein called the y^{th} bit. The value of y is determined by computing the maximum number of bits required to encode the absolute value of any data value in the analysis array. In particular, y is equal to $int(\log_2 V) + 1$, where V is the largest absolute value of any element in the analysis array, and "int()" represents the integer portion of a specified value.

The encoded data 162 representing one analysis array includes (A) header data 170 indicating the maximum number of bits required to encode the absolute value of any data value in the analysis array, and (B) a sequence of data structures 172, each representing one bit plane of the elements in the analysis array. The x^{th} bit plane of the analysis array is the x^{th} bit of the absolute value of each of the elements in the analysis array. A sparse data encoding technique is used so that it takes very little data to represent a bit plane that contains mostly zero values. Typically, higher frequency portions of the transformed, quantized image data will contain more zero values than non-zero values, and further most of the non-zero values will have relatively small absolute value. Therefore, the higher level bit planes of many analysis arrays will be populated with very few non-zero bit values.

In an alternate embodiment, the data structure shown in Fig. 4A is modified slightly. In particular, to facilitate fast extraction of lower-resolution image data from an image file, the boundaries of the analysis arrays are adjusted, if necessary, so as to coincide precisely with the boundaries between the wavelet transform regions shown in Fig. 3 (e.g., the boundary between HL2 and HL1). If the size of the initial image array is not equal to an integer number of

analysis arrays (i.e., if either the height or width of the image array is not an integer multiple of 2^n, where the size of each analysis array is 2^n x 2^n for an integer value of n), at least some of the boundaries between wavelet transform regions will fall in the middle of the analysis regions. For example, for a 800 x 600 pixel image, the LL region might have a size of 50 x 38. If the wavelet transform coefficients are encoded in units of analysis regions of size 32 x 32, the LL region will be encoded in four analysis regions, three of which would normally contain data for neighboring wavelet transform regions. In this alternate embodiment, each analysis array that overlaps a border between wavelet transform regions is replaced by two or four analysis regions (depending on whether the analysis array overlaps one or two region boundaries), with zero values being stored in the appropriate locations so that each analysis array contains data from only one wavelet transform region. The analysis arrays are still stored in "origin sorted order" in the image file 132, with the "origin" now being defined as the coordinate of the coefficient closest to the upper left corner of the analysis array that has not been overwritten with zero values.

In another alternate embodiment, a different transform than the wavelet-like transform could be used, but the resulting image data would still be stored in bit plane order. For instance, a DCT transform could be used.

In some embodiments of the present invention, the raw image array received from the digital camera's image capture mechanism may first be divided into "analysis arrays" and then transformed and quantized. Further, the analysis arrays may each be a thin horizontal strip of the image array. That is, each analysis array may extend the full width of the image array, but have a height of only a few (e.g., 4 to 16) image elements. In yet another embodiment, the image array might not be divided into analysis arrays at all.

Generally, in all embodiments described above, the compressed encoded image data is stored in bit plane order. The reason that bit plane ordered storage is favored is that it makes gradual fidelity reduction very easy: to reduce the fidelity of an image file by a minimum amount, the data for the lowest level bit plane in the file is discarded and the remaining image data is retained, resulting in a smaller file with one bit plane less fidelity.

However, in yet other alternate embodiments, each image file could be organized in "quality level support order" with the data for each analysis array being arranged so that each successive data structure 172 stores the image information needed to increase image quality by one predefined image quality level. Thus, the information in some data structures 172 might represent two, three or more bit planes of information. In this embodiment, an image can be reduced by one quality level by deleting the last data structure 172 from every analysis array data structure 172 in the image file.

Digital Camera State Machines

For the purposes of this explanation, it will be assumed that the digital camera 100 has four predefined image quality levels: High, Very Good +, Very Good -, and Good. It will be further assumed that image files stored at High quality typically occupy about twice as much space as image files stored at Good quality. In other embodiments, more or fewer image quality levels could be used, and the ratio of image file sizes from highest to lowest quality could be larger or smaller than 2:1. For instance, if the camera is capable of taking very high resolution images, such as 2000 x 2000 pixels or even 4000 x 4000 pixels, and at very high fidelity, then it would make sense to provide a large number of quality levels with a ratio of image file sizes from highest to lowest quality of perhaps as high as 64:1.

It is noted that an image file's quality cannot be increased once it has been lowered, unless the original image file or an alternate source thereof remains available, because the information needed to restore the image's quality has been lost.

Referring back to Fig. 1, the digital camera 100 preferably includes data processing circuitry 106 for performing a predefined set of primitive operations, such as performing the multiply and addition operations required to apply a transform to a certain amount of image data, as well as a set of state machines 200-212 for controlling the data processing circuitry so as to perform a set of predefined image handling operations. In one embodiment, the state machines in the digital camera are as follows.

- One or more state machines 200 for transforming, compressing and storing an image received from the camera's image capture mechanism. This image is sometimes called the "viewfinder" image, since the image being processed is generally the one seen on the camera's

image viewer 114. This set of state machines 200 are the ones that initially generate each image file stored in the nonvolatile image memory 108. Prior to taking the picture, the user specifies the quality level of the image to be stored, using the camera's buttons 112. It should noted that in most digital cameras the viewfinder is capable of displaying only a very small and low fidelity version of the captured image, and thus the image displayed in the camera's viewfinder is typically a much lower quality rendition of the captured image than the quality of the "viewfinder" image stored in the image file.

- One or more state machines 202 for decompressing, inverse transforming and displaying a stored image file on the camera's image viewer. The reconstructed image generated by decompressing, inverse transforming and dequantizing the image data is stored in camera's framebuffer 118 so that it can be viewed on the image viewer 114.
- One or more state machines 204 for updating and displaying a count of the number of images stored in the nonvolatile image memory 108. The image count is preferably displayed on the user interface display 116. This set of state machines 204 will also typically indicate what percentage of the nonvolatile image memory 108 remains unoccupied by image files, or some other indication of the camera's ability to store additional images. If the camera does not have a separate interface display 116, this memory status information may be shown on the image viewer 114, for instance superimposed on the image shown in the image viewer 114 or shown in a region of the viewer 114 separate from the main viewer image.
- One or more state machines 206 for implementing a "viewfinder" mode for the camera in which the image currently "seen" by the image capture mechanism 102 is displayed on the image viewer 114 to that the user can see the image that would be stored if the image capture button is pressed. These state machines transfer the image received from the image capture device 102, possibly after appropriate remedial processing steps are performed to improve the raw image data, to the camera's framebuffer 118.
- One or more state machines 208 for downloading images from the nonvolatile image memory 108 to an external device, such as a general purpose computer.
- One or more state machines 210 for uploading images from an external device, such as a general purpose computer, into the nonvolatile image memory 108. This enables the camera to be used as an image viewing device, and also as a mechanism for transferring image files on memory cards.
- One or more state machines 212 for reducing the size of image files in the nonvolatile image memory 108. This will be described in more detail next.

In the context of the present invention, an image file's quality level can be reduced in one of two ways: 1) by deleting from the image file all the analysis arrays associated with one or more transform layers, or 2) by deleting from the image file one or more bit planes of data. In either method, the state machines 212 extract the data structures of the image file that correspond to the new, lower image quality level selected by the user, and then replaces the original image file with one that stores the extracted data structures. Alternately, the original image file is updated by deleting a portion of its contents, thereby freeing some of the memory previously occupied by the file. A feature of the present invention is that the image quality level of an image file can be lowered without having to reconstruct the image and then re-encode it, which would be costly in terms of the computational resources used. Rather, the data structures within the image file are pre-arranged so that the image data in the file does not need to be read, analyzed or reconstructed. The image quality level of an image file is lowered simply by keeping an easily determined subset of the data in the image file and deleting the remainder of the data in the image file, or equivalently by extracting and storing in a new image file a determined subset of the data in the image file and deleting the original image file.

For the purposes of this document, it should be noted that the term "deleting" when applied to a data structure in an image file does not necessarily mean that the information in the data structure is replaced with null values. Rather, what this means is that the image file is replaced with another image file that does not contain the "deleted" data structure. Thus, various data structures in an image file may be deleted simply by copying all the other data structures in the image file into a new image file, and updating all required bookkeeping information in the image file and the image directory for the modified file. The "deleted" data structures may actually remain in memory unchanged until they are overwritten with new information. Alternately, data in an image file may in some implementations be deleted solely by updating the bookkeeping information for the file, without moving any of the image data.

In one embodiment of the present invention, the digital camera lowers the image quality of an image from High quality to "Very Good +" by deleting the two lowest bit planes of the image. Similarly, lowering the image's quality to "Very Good -" is accomplished by deleting two more bit planes of the image, and then lowering the image's quality to Good is accomplished by deleting yet another two bit planes of the image. More generally, each quality level

transition is represented by deleting a certain percentage of the bit planes of the highest quality level representation of the image.

In an alternate embodiment, the transition from High quality to the next highest quality level is accomplished by deleting the analysis arrays for a first transform layer (e.g., the analysis arrays for the HL1, HH1 and LH1 regions of the transformed image in Fig. 3). Subsequent quality level transitions to lower quality levels are accomplished by deleting appropriate numbers of bit planes.

In one embodiment of the present invention the digital camera provides two image file size reduction modes. In a first size reduction mode, the user selects one image (or a specified group of images), uses the camera's buttons to indicate what lower quality level the image is to be stored at, and then the state machine 212 generates a smaller image file and stores the resulting image file in the camera's nonvolatile image memory 108. In the second size reduction mode, the user commands the camera to reduce the size of all image files that are currently stored at quality level A to quality level B. For instance, in this second size reduction mode the user might command the camera to convert all "High" quality image files to "Very Good +" image quality files. This latter size reduction mode is particularly useful for "clearing space" in memory 108 to enable additional pictures to be stored in the memory 108.

In another embodiment, the camera or other device may include one or more automatic image file size reduction modes. For instance, in one such mode the camera could be set to record all pictures at a particular quality level. When the camera's memory is sufficiently filed with images files so that there is insufficient room to store onc more image at the current quality level setting, the camera automatically reduces the size of enough of the stored image files so as to create room for one more image at the current quality level. In some embodiments, the quality level setting of the device for future images might be automatically reduced to match the quality level of the highest quality image stored in the camera's memory. In this way, the camera takes and stores the maximum quality images for the space available, and this maximization will occur flexibly and "on-the-fly."

Camera Operation and
Image File Size Reduction

Referring to Fig. 5, the status of a digital camera is represented by status information displayed on the camera's user interface display 116. For example, before the camera's image memory 108 is filled, the camera might indicate to the user that it is currently storing twenty-one pictures at High quality and has enough memory to store three more pictures. The indication of how many more pictures can be stored in the camera's image memory 108 (Fig. 1) depends on the camera's current picture quality setting, which determines the quality of the next picture to be taken.

After the camera has stored three more pictures, the camera's image memory 108 is full (i.e., it has insufficient room to store another picture at the camera's current picture quality setting), and the camera indicates to the user that it is currently storing twenty-four pictures at High quality and has enough memory to store zero more pictures. For the purposes of this example, we will assume that the user wants to take at least ten more pictures, despite the fact that he/she has no more memory cards. To make this possible, the user utilizes the image size reduction feature of the camera.

In this example, the user commands the camera to reduce all "High" quality image files down one quality level to the "Very Good +" quality level. The camera accomplishes this by running the size reduction state machine 212 and then updating the status information displayed on the camera's user interface display 116. In this example, the twenty-four images are now shown to be stored in image files having the "Very Good +" quality level, and the camera has room for seven new images at the High quality image level.

In this example, the user next commands the camera to perform a second size reduction so as to compress all "Very Good +" quality image files down one quality level to the "Very Good -" quality level. The camera accomplishes this by running the size reduction state machine 212 and then updating the status information displayed on the camera's user interface display 116. In this example, the twenty-four images are now shown to be stored in image files having the "Very Good -" quality level, and the camera has room for twelve new images at the High quality image level.

Alternately, if the user had, before capturing the images, switched the quality level for new images to "Very Good +" quality, a single image size reduction step might have been sufficient to create room for at least ten additional pictures.

In another example, the digital camera may be configured to have an automatic image file size reduction mode that is activated only when the camera's memory is full and the user nevertheless presses the image capture button on the camera. In this mode of operation, the camera's image processing circuitry reduces the size of previously stored image files as little as possible so as to make room for an additional image file. If the user continues to take more pictures in this mode, the quality of the stored images will eventually degrade to some user defined or predefined setting for the lowest allowed quality level, at which point the camera will not store any additional image files until the permitted quality level is lowered further or at least some of the previously stored image files are transferred to another device or otherwise deleted.

The image management system and method of the present invention can also be implemented in computer systems and computer controlled systems, using a data processor that executes procedures for carrying of the image processing steps discussed above. The present invention can also be implemented as a computer program product (e.g., a CD-ROM or data signal conveyed on a carrier signal) containing image processing procedures suitable for use by a computer system.

Video Image Management System

Referring to Fig. 6, there is shown a conceptual data flow diagram for a video image management system for storing video images at a plurality of image quality levels. The basic structure of the video image management system is the same as shown in Fig. 1. However, when the camera is a digital video camera, successive images F_i are automatically generated at a predefined rate, such as eight, sixteen, twenty-four or thirty frames per second. In a preferred embodiment, the sequence of video images is processed N frames at a time, where N is an integer greater than three, and is preferably equal to four, eight or sixteen; generally N will be determined by the availability of memory and computational resources in the particular system in which the invention is being implemented. That is, each set of N (e.g., sixteen) successive

images (i.e., frames) are processed as a set, as follows. For each set of sixteen frames F_{16x} to F_{16x+15}, all the frames except the first one are replaced with differential frames. Thus, when N=16, fifteen differential frames F_{i+1}-F_i are generated. Then, following the data processing method shown in Fig. 3 and discussed above, the first frame and the fifteen differential frames are each divided into analysis arrays, a wavelet-like or other transform is applied to the analysis arrays, and then the resulting transform coefficients are encoded.

In alternate embodiments, other methodologies could be used for initially transforming and encoding each set of success frames. For instance, a frame by frame decision might be made, based on a measurement of frame similarity or dissimilarity, as to whether or not to replace the frame with a differential frame before applying the transform and encoding steps.

In all embodiments, the image file (or files) representing the set of frames is stored so as to facilitate the generation of smaller image files with minimal computational resources. In particular, the data in the image file(s) is preferably stored using distinct data structures for each bit plane (see Fig. 4B). Furthermore, as explained above with reference to Fig. 4A, the analysis arrays may be adjusted prior to the transform step so that the boundaries between analysis arrays correspond to the boundaries between transform layer coefficients. By so arranging the data stored in the video image files, the generation of smaller, lower quality level video image files is made much easier.

Continuing to refer to Fig. 6, after the video image files for a video frame sequence have been generated at a particular initial quality level, the user of the device (or the device operating in a particular automatic mode) may decide to reduce the size of the video image files while retaining as much image quality as possible. By way of example, in a first video image file size reduction step, the HH1 transform coefficients for the last eight frames of each sixteen frame sequence are deleted. In a second reduction step, the HH1 transform coefficients are deleted for all frames other than the first frame of each sixteen frame sequence. In a third reduction step, the Z (e.g., four) least significant bit planes of the video image files are deleted. In a fourth reduction step, the HL1 and LH1 coefficients are deleted for the last eight frames of each sixteen frame sequence. In a fifth reduction step, the HL1 and LH1 coefficients are deleted for all frames other than the first frame of each sixteen frame sequence. These reduction steps are only examples of the type of file size reduction steps that could be

performed. For instance, in other embodiments, bit planes might be deleted in earlier reduction steps and transform layers (or portions of transform layers) deleted only in later reduction steps. In general, each video image size reduction causes a corresponding decrease in image quality.

Referring to Figs. 7, 8 and 9, in another embodiment, video image sequences are compressed and encoded by performing a time domain, one dimensional wavelet transformation on a set of video frames. In particular, the video frames are divided into groups of N frames, where N is an integer greater than 3, and for every x,y pixel position in the video image, a one dimensional K level wavelet transform is performed on the pixels for a sequence of N+1 frames. For instance, the K level wavelet transform is performed on the 1,1 pixels for the last frame of the previous group and the current group of N frames, as well as the 1, 2 pixels, the 1,3 pixels and so on.

In order to avoid artifacts from the separate encoding of each group of N frames, the frame immediately preceding the current group is included in K level wavelet transform. The wavelet transform uses a short transform filter that extends only one position to the left (backwards in time) of the position for which a coefficient is being generated and extends in the right hand (forward in time) direction only to the right hand edge of the set of N frames.

Furthermore, as shown in Fig. 8, the "right edge" coefficients are saved for each of the first through K-1 level wavelet transforms for use when processing the next group of N frames. In a preferred embodiment, only the rightmost edge coefficient is saved for each of the first through K-1 level transforms; in other embodiments two or more right edge coefficients may be saved and used when processing the next block of N frames. When the second level transform is performed on a block of N frames, the saved layer 1 right edge coefficients for the previous set of N frames are used (i.e., included in the computation of the leftmost computed coefficient(s) for layer 2). By saving the rightmost edge coefficients for each of the 1 through K-1 layers, artifacts that would caused (during regeneration of the video image sequence) by the discontinuities between the last frame of one block and the first frame of the next block are avoided, resulting in a smoother and more visually pleasing reconstructed video image sequence. The wavelet-like transformation and data compression of a video sequence is shown in pseudocode form in Table 1.

Table 1
Pseudocode for Wavelet-Like Transform and
Compression of One Block of Video Frames

```
Repeat for each block of video frames:
{
For each row y (of the images)
        {
        For each column x (of the images)
                {
                Save rightmost edge value for use when processing next block of video frames;

                Apply level 1 wavelet-like transform to time-ordered sequence of pixel values
                at position x,y, including saved edge value from prior block to generate level 1
                L and H coefficients;

                Save rightmost edge L coefficient for use when processing next block of video
                frames;

                Apply level 2 wavelet-like transform to level 1 L coefficients for position x,y,
                including saved level 1 edge value from prior block to generate level 2 L and H
                coefficients;

                Save rightmost edge level 2 L coefficient for use when processing next block of
                video frames;

                ...

                Apply level k-1 wavelet-like transform to level k-2 L coefficients for position
                x,y, including saved level k-2 edge value from prior block to generate level k-1
                L and H coefficients;

                Save rightmost edge level k-1 L coefficient for use when processing next block
                of video frames;

                Apply level k wavelet-like transform to level k-1 L coefficients for position x,y,
                including saved level k-1 edge value from prior block to generate level k L and
                H coefficients;
                }
        }
Quantize coefficients
Encode coefficients
Store coefficients in image data structure(s), creating image file for current block of video
frames
}
```

A more detailed explanation of saving edge coefficient from one block of image data for use while performing a wavelet or wavelet like transforms on a neighboring block of image data is provided in U.S. patent application serial no. 09/358,876, filed 07-22-99, "Memory Saving Wavelet-Like Image Transform System and Method for Digital Camera And Other Memory Conservative Applications," which is hereby incorporated by reference as background information.

Once the wavelet-like transform of each block of video data has been completed, all other aspects of processing the transformed video data are as described above. That is, the transformed data is quantized, stored in image data structures and subject to reductions in image quality, using the same techniques as those applied to still images and video image sequences as described above.

The video image management system and method of the present invention can also be implemented in computer systems and computer controlled systems, using a data processor that executes procedures for carrying of the video frame processing steps discussed above. The present invention can also be implemented as a computer program product (e.g., a CD-ROM or data signal conveyed on a carrier signal) containing image and/or video frame processing procedures suitable for use by a computer system.

Alternate Embodiments

The state machines of the embodiments described above can be replaced by software procedures that are executed by a general purpose (programmable) data processor or a programmable image data processor, especially if speed of operation is not a concern.

Numerous other aspects of the described embodiments may change over time as technology improvements are used to upgrade various parts of the digital camera. For instance, the memory technology used to store image files might change from flash memory to another type of memory, or a camera might respond to voice commands, enabling the use of fewer buttons.

Referring to Fig. 10, the present invention can also be used in a variety of image processing systems other than digital cameras and digital video cameras, including cable television set top

boxes, computer systems and devices used to warehouse libraries of images, computer systems and devices used to store and distribute image files, and so on. For example, an Internet server 300 can store images and/or video sequences in the wavelet transform compressed data structures of the present invention. Copies of those compressed data structures are transferred to the memory 306 of client computers or other client devices 302, using HTTP or any other suitable protocol via the Internet 304 or other communications network. When appropriate, an image or video sequence is reduced in size so as to fit in the memory available in the client computer or other device (client device). Furthermore, once an image or video sequence has been stored in the memory 306 of a client device, the techniques of the present invention can be used to manage the storage of the image, for instance through gradual reduction of image quality so as to make room for the storage of additional images or video sequences. In the embodiment shown in Fig. 10, the memory 306 of the client computer will have stored therein:

- an operating system 310;
- a browser or other image viewer application 312 for viewing documents and images;
- image files 314;
- image transform procedures 316, such as wavelet or wavelet-like transform procedures for converting a raw image array into wavelet transform coefficients, procedures for compressing and encoding the wavelet transform coefficients, as well as other transform procedures for handling images received in other image formats, such JPEG transform procedures for converting JPEG files into reconstructed image data that is then used as the raw image data by a wavelet or wavelet-like transform procedure;
- an image compression, quality reduction procedure 318 for implementing the image data structure size and quality reduction features of the present invention; and
- image reconstruction procedures 320 for decompressing and reverse transforming image files so as to generate reconstructed image data arrays that are suitable for viewing on the monitor of the client workstation, or for printing or other use.

The client workstation memory 306 will typically include both high speed random access memory and slower non-volatile memory such as a hard disk and/or read-only memory. The client workstation's central processing unit(s) 308 execute operating system procedures and image handling procedures, as well as other applications, thereby performing image processing functions similar to those performed by dedicated circuitry in other embodiments of the present invention.

As indicated above, when the present invention is used in conjunction with, or as part of, a browser application, for management of image storage, some images may be initially received in formats other than "raw" image arrays. For instance, some images may be initially received as JPEG files, or in other proprietary or industry standard formats. To make full use of the capabilities of the present invention, such images are preferably decoded so as to generate reconstructed "raw" image arrays, and then those raw image arrays are wavelet or wavelet-like transformed so as to put the images in a form that enables use of the image quality level management features of the present invention.

While the present invention has been described with reference to a few specific embodiments, the description is illustrative of the invention and is not to be construed as limiting the invention. Various modifications may occur to those skilled in the art without departing from the true spirit and scope of the invention as defined by the appended claims.

WHAT IS CLAIMED IS:

1. Image processing apparatus, for use in conjunction with an image capture mechanism, the image processing apparatus comprising:

a memory device for storing a plurality of image data structures that each represent a respective image, each image data structure having an associated image quality level corresponding to a quality level at which the corresponding image has been encoded in the image data structure; the image quality level of each image data structure being a member of predefined range of image quality levels that range from a highest quality level to a lowest quality level and that include at least two distinct quality levels;

image management logic, including data processing circuitry and state machines for storing and processing image data received from the image capture mechanism, the data processing circuitry and state machines including:

image processing circuitry for applying a predefined transform to image data received from the image capture mechanism to generate transform image data and for applying a data compression method to the transform image data so as to generate a new image data structure having an associated image quality level selected from the predefined range of image quality levels; the new image data structure being stored in the memory device;

image size reduction circuitry for extracting a subset of the data in a first specified one of the image data structures stored in the memory device, and forming a lower quality version of the first specified image data structure that occupies less space in the memory device than was previously occupied by the first specified image data structure; and

image reconstruction circuitry for successively applying a data decompression method and an inverse transform to any specified one of the image data structures so as to generate a reconstructed image suitable for display on an image viewer;

wherein the amount of space occupied by images stored in the form of image data structures in the memory device can be reduced so as to make room for the storage of additional image data structures in the memory device.

2. The image processing apparatus of claim 1, wherein

each image data structure contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes; and

the image size reduction circuitry and one or more state machines includes logic for extracting a portion of an image data structure that excludes the image transform data for at least one bit plane and for replacing the image data structure with an image data structure containing the extracted portion.

3. The image processing apparatus of claim 1, wherein

each of a subset of the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the image size reduction circuitry and one or more state machines includes logic, operative when the first specified data structure is a member of the subset of image data structures, for extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer and for replacing the first specified image data structure with an image data structure containing the extracted portion.

4. Image processing apparatus, for use in conjunction with an image capture mechanism, the image processing apparatus comprising:

a memory device for storing a plurality of image data structures that each represent a respective image, each image data structure having an associated image quality level corresponding to a quality level at which the corresponding image has been encoded in the image data structure; the image quality level of each image data structure being a member of predefined range of image quality levels that range from a highest quality level to a lowest quality level and that include at least two distinct quality levels;

image management logic for storing and processing image data received from the image capture mechanism, including:

a data processor coupled to the memory device;

image management procedures, executable by the data processor, including instructions for storing and processing image data received from the image capture mechanism, the instructions including:

an initial image processing procedure for applying a predefined transform to image data received from the image capture mechanism to generate transform image data and for applying a data compression procedure to the transform image data so as to generate an

image data structure having an associated image quality level selected from the predefined range of image quality levels;

an image size reduction procedure for lowering the quality level of a first specified one of the image data structures, including instructions for extracting a subset of the data in the first specified image data structure and forming a lower quality version of the first specified image data structure that occupies less space in the memory device than was previously occupied by the first specified image data structure; and

at least one image reconstruction procedure for successively applying a data decompression method and an inverse transform to any specified one of the image data structures stored in the memory device so as to generate a reconstructed image suitable for display on an image viewer;

wherein the amount of space occupied by images stored in the form of image data structures in the memory device can be reduced so as to make room for the storage of additional image data structures in the memory device.

5. The image processing apparatus of claim 4, wherein

each of the image data structures contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes; and

the image size reduction instructions include instructions for extracting a portion of an image data structure that excludes the image transform data for at least one bit plane and for replacing the image data structure with an image data structure containing the extracted portion.

6. The image processing apparatus of claim 4, wherein

each of a subset of the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the image size reduction instructions include instructions, operative when the first specified data structure is a member of the subset of image data structures, for extracting a portion of the first specified image data structure that excludes the image transform data for at

least one transform layer and for replacing the first specified image data structure with an image data structure containing the extracted portion.

7. Image processing apparatus, comprising:

image management logic, including:

image processing circuitry for applying a predefined transform to an array of image data so as to generate transform image data and for applying a data compression method to the transform image data so as to generate an image data structure having an associated image quality level selected from a predefined range of image quality levels that range from a highest quality level to a lowest quality level and that include at least two distinct quality levels;

a memory device for storing the image data structure and other image data structures representing a set of images;

the image management logic further including:

image size reduction circuitry for extracting a subset of the data in a first specified one of the image data structures stored in the memory device, and forming a reduced size version of the first specified image data structure that occupies less space in the memory device than was previously occupied by the first specified image data structure and that has a lower associated image quality level than the image quality level associated with the first specified image data structure; and

image reconstruction circuitry for successively applying a data decompression method and an inverse transform to any specified one of the image data structures stored in the memory device so as to generate a reconstructed image suitable for display on an image viewer;

wherein the amount of space occupied by the image data structures in the memory device can be reduced so as to make room for the storage of additional image data structures in the memory device.

8. The image processing apparatus of claim 7, further including a communications interface for receiving the image data from another apparatus.

9. The image processing apparatus of claim 8, wherein

each of a subset of the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the image size reduction circuitry and one or more state machines includes logic, operative when the first specified data structure is a member of the subset of image data structures, for extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer and for replacing the first specified image data structure with an image data structure containing the extracted portion.

10. The image processing apparatus of claim 8, wherein

each of a subset of the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the image size reduction circuitry and one or more state machines includes logic, operative when the first specified data structure is a member of the subset of image data structures, for extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer and for replacing the first specified image data structure with an image data structure containing the extracted portion.

11. Image processing apparatus, comprising:

a communications interface for receiving an image data structure having an associated image quality level selected from a predefined range of image quality levels that range from a highest quality level to a lowest quality level and that include at least two distinct quality levels;

a memory device for storing the image data structure and other image data structures representing a set of images;

image management logic, including:

image size reduction circuitry for extracting a subset of the data in a first specified one of the image data structures to form a reduced size image data structure that occupies less space in the memory device than was previously occupied by the first specified

image data structure and that has a lower associated image quality level than the quality level associated with the first specified image data structure; and

image reconstruction circuitry for successively applying a data decompression method and an inverse transform to any specified one of the image data structures and the reduced size image data structure so as to generate a reconstructed image suitable for display on a display device;

wherein the amount of space occupied by the image data structure in the memory device can be reduced so as to make room for the storage of additional image data structures in the memory device.

12. The image processing apparatus of claim 11, wherein

each image data structure contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes;

the image size reduction circuitry and one or more state machines including logic for extracting a portion of the first specified image data structure that excludes the image transform data for at least one bit plane and for replacing the first specified image data structure with an image data structure containing the extracted portion.

13. The image processing apparatus of claim 8, wherein

each of a subset of the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the image size reduction circuitry and one or more state machines includes logic, operative when the first specified data structure is a member of the subset of image data structures, for extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer and for replacing the first specified image data structure with an image data structure containing the extracted portion.

14. Image processing apparatus, the image processing apparatus comprising:

a communications interface for receiving an image data structure having an associated image quality level selected from a predefined range of image quality levels that range from a

highest quality level to a lowest quality level and that include at least two distinct quality levels;

a memory device for storing the image data structure and other image data structures representing a set of images;

a data processor coupled to the memory device;

image management procedures, executable by the data processor, including instructions for storing and processing image data, the instructions including:

an image size reduction procedure for lowering the quality level of a first specified one of the image data structures, including instructions for extracting a subset of the data in the first specified image data structure and forming a lower quality version of the first specified image data structure that occupies less space in the memory device than was previously occupied by the first specified image data structure; and

at least one image reconstruction procedure for successively applying a data decompression method and an inverse transform to any specified one of the image data structures stored in the memory device so as to generate a reconstructed image suitable for display on an image viewer;

wherein the amount of space occupied by images stored in the form of image data structures in the memory device can be reduced so as to make room for the storage of additional image data structures in the memory device.

15. The image processing apparatus of claim 14, wherein

each of the image data structures contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes; and

the image size reduction instructions include instructions for extracting a portion of an image data structure that excludes the image transform data for at least one bit plane and for replacing the image data structure with an image data structure containing the extracted portion.

16. The image processing apparatus of claim 14, wherein

each of a subset of the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is

stored in distinct portions of the image data structure from image transform data for other transform layers; and

the image size reduction instructions include instructions, operative when the first specified data structure is a member of the subset of image data structures, for extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer and for replacing the first specified image data structure with an image data structure containing the extracted portion.

17. A computer program product, for use in conjunction with a computer system having a memory in which image data structures can be stored, the computer program product comprising a computer readable storage medium and a computer program mechanism embedded therein, the computer program mechanism comprising:

an image handling procedure, including instructions for storing in the memory of the computer system a plurality of image data structures,

an image size reduction procedure for accessing image data structures in the memory of the computer system, each of the image data structures containing image transform data, lowering the quality level of a first specified one of the image data structures, including instructions for extracting a subset of the data in a first specified image data structure and forming a lower quality version of the first specified image data structure that occupies less space in the memory device than was previously occupied by the first specified image data structure; and

at least one image reconstruction procedure for successively applying a data decompression procedure and an inverse transform to any specified one of the image data structures stored in the memory device so as to generate a reconstructed image suitable for display on an image viewer;

wherein the amount of space occupied by images stored in the form of image data structures in the memory device can be reduced so as to make room for the storage of additional image data structures in the memory device.

18. The computer program product of claim 17, wherein

each of the image data structures contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes; and

the image size reduction procedure includes instructions for extracting a portion of the first specified image data structure that excludes the image transform data for at least one bit plane and for replacing the first specified image data structure with an image data structure containing the extracted portion.

19. The computer program product of claim 17, wherein

each of a subset of the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the image size reduction procedure includes instructions, operative when the first specified data structure is a member of the subset of image data structures, for extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer and for replacing the first specified image data structure with an image data structure containing the extracted portion.

20. The computer program product of claim 17, wherein the image handling procedure includes one or more image processing procedures for applying a predefined transform to raw image data to generate transform image data and for applying a data compression procedure to the transform image data so as to generate an image data structure having an associated image quality level selected from the predefined range of image quality levels.

21. A method of processing images, comprising:

storing in a memory device a plurality of image data structures that each represent a respective image, each image data structure having an associated image quality level corresponding to a quality level at which the corresponding image has been encoded in the image data structure; the image quality level of each image data structure being a member of predefined range of image quality levels that range from a highest quality level to a lowest quality level and that include at least two distinct quality levels;

reducing the size of a specified one of the image data structures stored in the nonvolatile memory device, including extracting a subset of the data in the specified image data structure and forming a lower quality version of the specified image data structure that

occupies less space in the nonvolatile memory device than was previously occupied by the specified image data structure; and

successively applying a data decompression method and an inverse transform to a specified one of the image data structures stored in the nonvolatile memory device so as to generate a reconstructed image suitable for display on an image viewer;

wherein the amount of space occupied by images stored in the form of image data structures in the nonvolatile memory device can be reduced so as to make room for the storage of additional image data structures in the nonvolatile memory device.

22. The method of claim 21, wherein the method is performed by a digital camera and the method includes applying a predefined transform to image data received from an image capture mechanism in the digital camera to generate transform image data, applying a data compression method to the transform image data so as to generate an image data structure having an associated image quality level selected from the predefined range of image quality levels, and storing the image data structure in the memory device.

23. The method of claim 21, wherein the method includes applying a predefined transform to raw image data to generate transform image data, applying a data compression method to the transform image data so as to generate an image data structure having an associated image quality level selected from the predefined range of image quality levels, and storing the image data structure in the memory device.

24. The method of claim 21, wherein

each image data structure contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes;

the size reduction step includes extracting a portion of an image data structure that excludes the image transform data for at least one bit plane and for replacing the image data structure in the nonvolatile memory device with an image data structure containing the extracted portion.

25. The method of claim 21, wherein

each of a subset of the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the size reduction step includes extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer and replacing the first specified image data structure with an image data structure containing the extracted portion.

26. Video image processing apparatus, comprising:

a memory device for storing a set of image data structures representing a sequence of video frames, the set of image data structures having an associated image quality level selected from a predefined range of image quality levels that range from a highest quality level to a lowest quality level and that include at least two distinct quality levels; and

image management logic including:

image size reduction circuitry for extracting a subset of the data in the set of image data structures and forming a lower quality version of the set of image data structures that occupies less space in the memory device than was previously occupied by the set of image data structures; and

image reconstruction circuitry for successively applying a data decompression method and an inverse transform to at least a subset of the image data structures so as to generate a reconstructed sequence of video frames suitable for display on a display device;

whereby the amount of space occupied by the set of image data structures in the memory device can be reduced so as to make room for the storage of additional image data structures in the memory device.

27. The video image processing apparatus of claim 26, wherein

each image data structure contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes;

the image size reduction circuitry and one or more state machines including logic for extracting a portion of an image data structure that excludes the image transform data for at

least one bit plane and for replacing the image data structure with an image data structure containing the extracted portion.

28. The video image processing apparatus of claim 26, wherein

the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the image size reduction circuitry and one or more state machines includes logic, operative when the first specified data structure is a member of the subset of image data structures, for extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer and for replacing the first specified image data structure with an image data structure containing the extracted portion.

29. The video image processing apparatus of claim 26, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to at least one video frame in each said sub-sequence of N frames.

30. The video image processing apparatus of claim 29, wherein the wavelet-like transform is applied to at least one difference frame for each said sub-sequence of N frames, the difference frame representing differences between one frame and a next frame in said sub-sequence of N frames.

31. The video image processing apparatus of claim 26, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to separately and in time order to data at each x,y position in the video frames.

32. Video image processing apparatus, comprising:

image management logic, including:

image processing circuitry for applying a predefined transform to a sequence of video frames to generate transform image data and for applying a data compression method to the transform image data so as to generate a set of image data structures having an associated

image quality level selected from a predefined range of image quality levels that range from a highest quality level to a lowest quality level and that include at least two distinct quality levels;

a memory device for storing the set of image data structures;

the image management logic further including:

image size reduction circuitry for extracting a subset of the data in the set of image data structures and forming a lower quality version of the set of image data structures that occupies less space in the memory device than was previously occupied by the set of image data structures; and

image reconstruction circuitry for successively applying a data decompression method and an inverse transform to at least a subset of the image data structures so as to generate a reconstructed sequence of video frames suitable for display on a display device;

whereby the amount of space occupied by the set of image data structures in the memory device can be reduced so as to make room for the storage of additional image data structures in the memory device.

33. The video image processing apparatus of claim 32, wherein

each image data structure contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes;

the image size reduction circuitry and one or more state machines including logic for extracting a portion of an image data structure that excludes the image transform data for at least one bit plane and for replacing the image data structure with an image data structure containing the extracted portion.

34. The video image processing apparatus of claim 32, wherein

the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the image size reduction circuitry and one or more state machines includes logic, operative when the first specified data structure is a member of the subset of image data structures, for extracting a portion of the first specified image data structure that excludes the

image transform data for at least one transform layer and for replacing the first specified image data structure with an image data structure containing the extracted portion.

35. The video image processing apparatus of claim 32, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to at least one video frame in each said sub-sequence of N frames.

36. The video image processing apparatus of claim 35, wherein the wavelet-like transform is applied to at least one difference frame for each said sub-sequence of N frames, the difference frame representing differences between one frame and a next frame in said sub-sequence of N frames.

37. The video image processing apparatus of claim 32, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to separately and in time order to data at each x,y position in the video frames.

38. Video image processing apparatus, comprising:

a memory device for storing a set of image data structures representing a sequence of video frames, the set of image data structures having an associated image quality level selected from a predefined range of image quality levels that range from a highest quality level to a lowest quality level and that include at least two distinct quality levels;

a data processor coupled to the memory device;

image management procedures, executable by the data processor, including

an image size reduction procedure for extracting a subset of the data in the set of image data structures and forming a lower quality version of the set of image data structures that occupies less space in the memory device than was previously occupied by the set of image data structures; and

at least one image reconstruction procedure for successively applying a data decompression method and an inverse transform to at least a subset of the image data structures so as to generate a reconstructed sequence of video frames suitable for display on a display device;

whereby the amount of space occupied by the set of image data structures in the memory device can be reduced so as to make room for the storage of additional image data structures in the memory device.

39. The video image processing apparatus of claim 38, wherein

each image data structure contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes;

the at least one image size reduction procedure includes instructions for extracting a portion of an image data structure that excludes the image transform data for at least one bit plane and for replacing the image data structure with an image data structure containing the extracted portion.

40. The video image processing apparatus of claim 38, wherein

the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the at least one image size reduction procedure and one or more state machines include logic, operative when the first specified data structure is a member of the subset of image data structures, for extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer and for replacing the first specified image data structure with an image data structure containing the extracted portion.

41. The video image processing apparatus of claim 38, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to at least one video frame in each said sub-sequence of N frames.

42. The video image processing apparatus of claim 41, wherein the wavelet-like transform is applied to at least one difference frame for each said sub-sequence of N frames, the difference frame representing differences between one frame and a next frame in said sub-sequence of N frames.

43. The video image processing apparatus of claim 38, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to separately and in time order to data at each x,y position in the video frames.

44. Video image processing apparatus, comprising:

a memory device for storing image data structures;

a data processor coupled to the memory device;

image management procedures, executable by the data processor, including:

at least one image processing procedure for applying a predefined transform to a sequence of video frames to generate transform image data and for applying a data compression method to the transform image data so as to generate a set of image data structures having an associated image quality level selected from a predefined range of image quality levels that range from a highest quality level to a lowest quality level and that include at least two distinct quality levels, and for storing the set of image data structures in the memory device;

an image size reduction procedure for extracting a subset of the data in the set of image data structures and forming a lower quality version of the set of image data structures that occupies less space in the memory device than was previously occupied by the set of image data structures; and

at least one image reconstruction procedure for successively applying a data decompression method and an inverse transform to at least a subset of the image data structures so as to generate a reconstructed sequence of video frames suitable for display on a display device;

whereby the amount of space occupied by the set of image data structures in the memory device can be reduced so as to make room for the storage of additional image data structures in the memory device.

45. The video image processing apparatus of claim 44, wherein

each image data structure contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes;

the at least one image size reduction procedure includes instructions for extracting a portion of an image data structure that excludes the image transform data for at least one bit plane and for replacing the image data structure with an image data structure containing the extracted portion.

46. The video image processing apparatus of claim 44, wherein

the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the at least one image size reduction procedure and one or more state machines include logic, operative when the first specified data structure is a member of the subset of image data structures, for extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer and for replacing the first specified image data structure with an image data structure containing the extracted portion.

47. The video image processing apparatus of claim 44, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to at least one video frame in each said sub-sequence of N frames.

48. The video image processing apparatus of claim 47, wherein the wavelet-like transform is applied to at least one difference frame for each said sub-sequence of N frames, the difference frame representing differences between one frame and a next frame in said sub-sequence of N frames.

49. The video image processing apparatus of claim 44, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to separately and in time order to data at each x,y position in the video frames.

50. A computer program product, for use in conjunction with a computer system having a memory in which image data structures can be stored, the computer program product comprising a computer readable storage medium and a computer program mechanism embedded therein, the computer program mechanism comprising:

an image handling procedure, including instructions for storing in the memory of the computer system a set of image data structures representing a sequence of video frames, the set of image data structures having an associated image quality level selected from a predefined range of image quality levels that range from a highest quality level to a lowest quality level and that include at least two distinct quality levels;

a data processor coupled to the memory device;

an image size reduction procedure for extracting a subset of the data in the set of image data structures and forming a lower quality version of the set of image data structures that occupies less space in the memory device than was previously occupied by the set of image data structures; and

at least one image reconstruction procedure for successively applying a data decompression method and an inverse transform to at least a subset of the image data structures so as to generate a reconstructed sequence of video frames suitable for display on a display device;

whereby the amount of space occupied by the set of image data structures in the memory device can be reduced so as to make room for the storage of additional image data structures in the memory device.

51. The computer program product of claim 50, wherein

each image data structure contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes;

the at least one image size reduction procedure includes instructions for extracting a portion of an image data structure that excludes the image transform data for at least one bit

plane and for replacing the image data structure with an image data structure containing the extracted portion.

52. The computer program product of claim 50, wherein

the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the at least one image size reduction procedure and one or more state machines include logic, operative when the first specified data structure is a member of the subset of image data structures, for extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer and for replacing the first specified image data structure with an image data structure containing the extracted portion.

53. The computer program product of claim 50, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to at least one video frame in each said sub-sequence of N frames.

54. The computer program product of claim 53, wherein the wavelet-like transform is applied to at least one difference frame for each said sub-sequence of N frames, the difference frame representing differences between one frame and a next frame in said sub-sequence of N frames.

55. The computer program product of claim 50, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to separately and in time order to data at each x,y position in the video frames.

56. The computer program product of claim 50, including:

at least one image processing procedure for applying a predefined transform to a sequence of video frames to generate transform image data and for applying a data compression method to the transform image data so as to generate the set of image data structures stored in the memory.

57. The computer program product of claim 56, wherein

each image data structure contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes;

the at least one image size reduction procedure includes instructions for extracting a portion of an image data structure that excludes the image transform data for at least one bit plane and for replacing the image data structure with an image data structure containing the extracted portion.

58. The computer program product of claim 56, wherein

the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the at least one image size reduction procedure and one or more state machines include logic, operative when the first specified data structure is a member of the subset of image data structures, for extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer and for replacing the first specified image data structure with an image data structure containing the extracted portion.

59. The computer program product of claim 56, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to at least one video frame in each said sub-sequence of N frames.

60. The computer program product of claim 59, wherein the wavelet-like transform is applied to at least one difference frame for each said sub-sequence of N frames, the difference frame representing differences between one frame and a next frame in said sub-sequence of N frames.

61. The computer program product of claim 56, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to separately and in time order to data at each x,y position in the video frames.

62. A method of processing video images, comprising:

storing in a memory device a set of image data structures representing a sequence of video frames, the set of image data structures having an associated image quality level selected from a predefined range of image quality levels that range from a highest quality level to a lowest quality level and that include at least two distinct quality levels;

extracting a subset of the data in the set of image data structures and forming a lower quality version of the set of image data structures that occupies less space in the memory device than was previously occupied by the set of image data structures; and

successively applying a data decompression method and an inverse transform to the specified set of image data structures so as to generate a reconstructed sequence of video images suitable for display on a display device;

whereby the amount of space occupied by the set of image data structures in the memory device can be reduced so as to make room for the storage of additional image data structures in the memory device.

63. The method of claim 62, wherein

each of the image data structures contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes;

the extracting step includes extracting a portion of an image data structure that excludes the image transform data for at least one bit plane, and the forming step includes replacing the image data structure with an image data structure containing the extracted portion.

64. The method of claim 62, wherein

the of the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the extracting step includes extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer, and the forming step includes replacing the first specified image data structure with an image data structure containing the extracted portion.

65. The method of claim 62, after performing the extracting and forming steps, applying the predefined transform to a sequence of additional video images to generate transform image data and applying the data compression method to the transform image data so as to generate an additional set of image data structures having an associated image quality, and storing the additional set of image data structures in the memory device.

66. The method of claim 62, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to at least one video frame in each said sub-sequence of N frames.

67. The method of claim 66, wherein the wavelet-like transform is applied to at least one difference frame for each said sub-sequence of N frames, the difference frame representing differences between one frame and a next frame in said sub-sequence of N frames.

68. The method of claim 62, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to separately and in time order to data at each x,y position in the video frames.

69. The method of claim 62, including applying a predefined transform to a sequence of video images to generate transform image data and applying a data compression method to the transform image data so as to generate the set of image data structures stored in the memory device.

70. The method of claim 69, wherein

each of the image data structures contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes;

the extracting step includes extracting a portion of an image data structure that excludes the image transform data for at least one bit plane, and the forming step includes replacing the image data structure with an image data structure containing the extracted portion.

71. The method of claim 69, wherein

the of the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the extracting step includes extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer, and the forming step includes replacing the first specified image data structure with an image data structure containing the extracted portion.

72. The method of claim 69, after performing the extracting and forming steps, applying the predefined transform to a sequence of additional video images to generate transform image data and applying the data compression method to the transform image data so as to generate an additional set of image data structures having an associated image quality, and storing the additional set of image data structures in the memory device.

73. The method of claim 69, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to at least one video frame in each said sub-sequence of N frames.

74. The method of claim 73, wherein the wavelet-like transform is applied to at least one difference frame for each said sub-sequence of N frames, the difference frame representing differences between one frame and a next frame in said sub-sequence of N frames.

75. The method of claim 69, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to separately and in time order to data at each x,y position in the video frames.

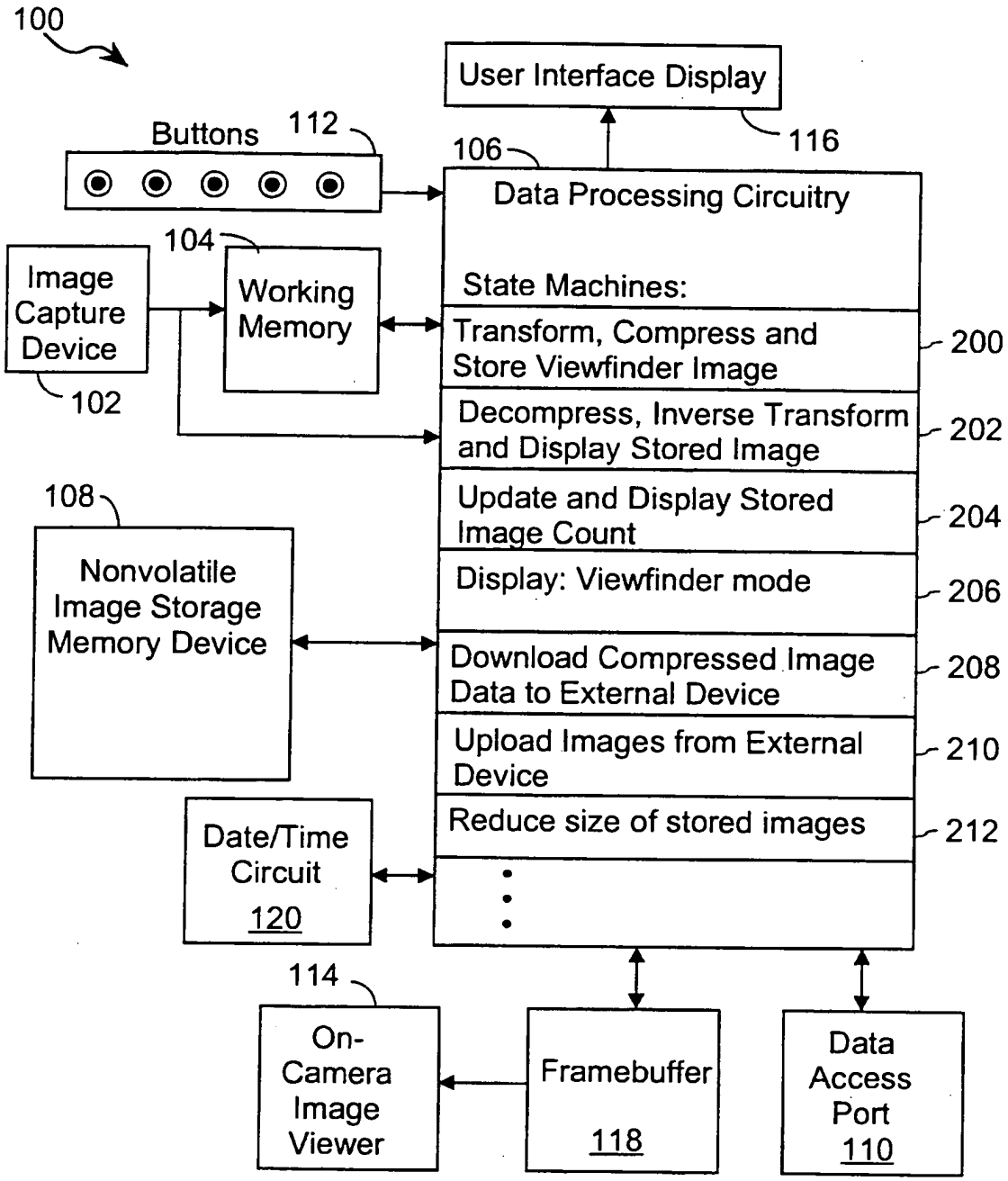

100
User Interface Display
116
Buttons
112
106
Data Processing Circuitry
State Machines:
104
Image Capture Device
Working Memory
102
Transform, Compress and Store Viewfinder Image
200
Decompress, Inverse Transform and Display Stored Image
202
Update and Display Stored Image Count
204
108
Nonvolatile Image Storage Memory Device
Display: Viewfinder mode
206
Download Compressed Image Data to External Device
208
Upload Images from External Device
210
Reduce size of stored images
212
Date/Time Circuit
120
114
On-Camera Image Viewer
Framebuffer
118
Data Access Port
110

FIG. 1

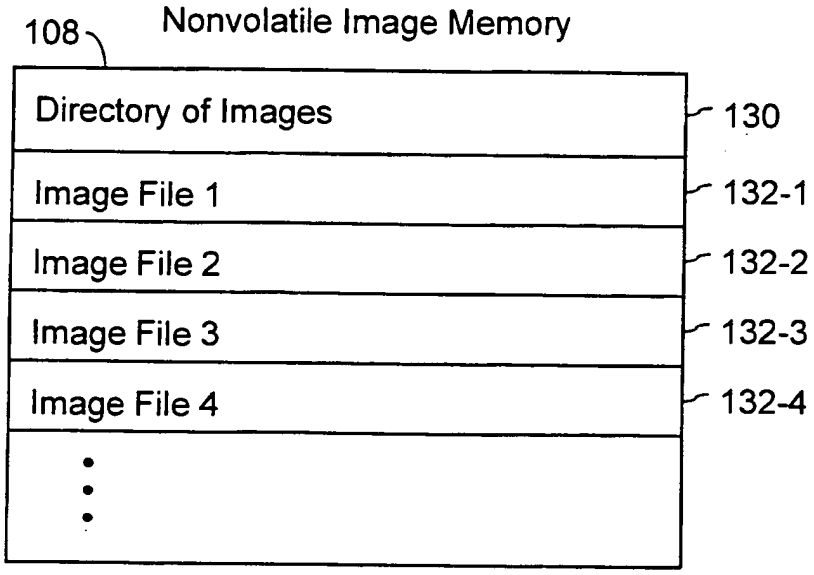

108
Nonvolatile Image Memory
Directory of Images
130
Image File 1
132-1
Image File 2
132-2
Image File 3
132-3
Image File 4
132-4

FIG. 2

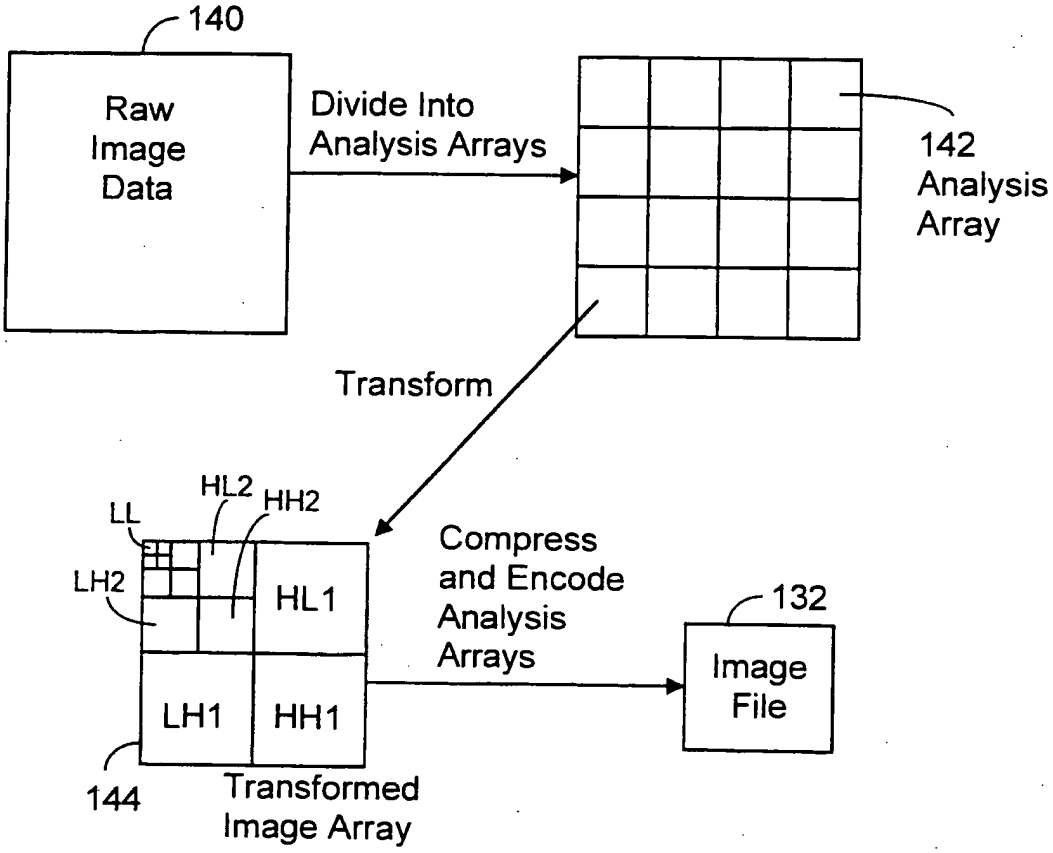

140
Raw Image Data
Divide Into Analysis Arrays
142 Analysis Array
Transform
LL
HL2
HH2
LH2
HL1
LH1
HH1
144
Transformed Image Array
Compress and Encode Analysis Arrays
132
Image File

FIG. 3

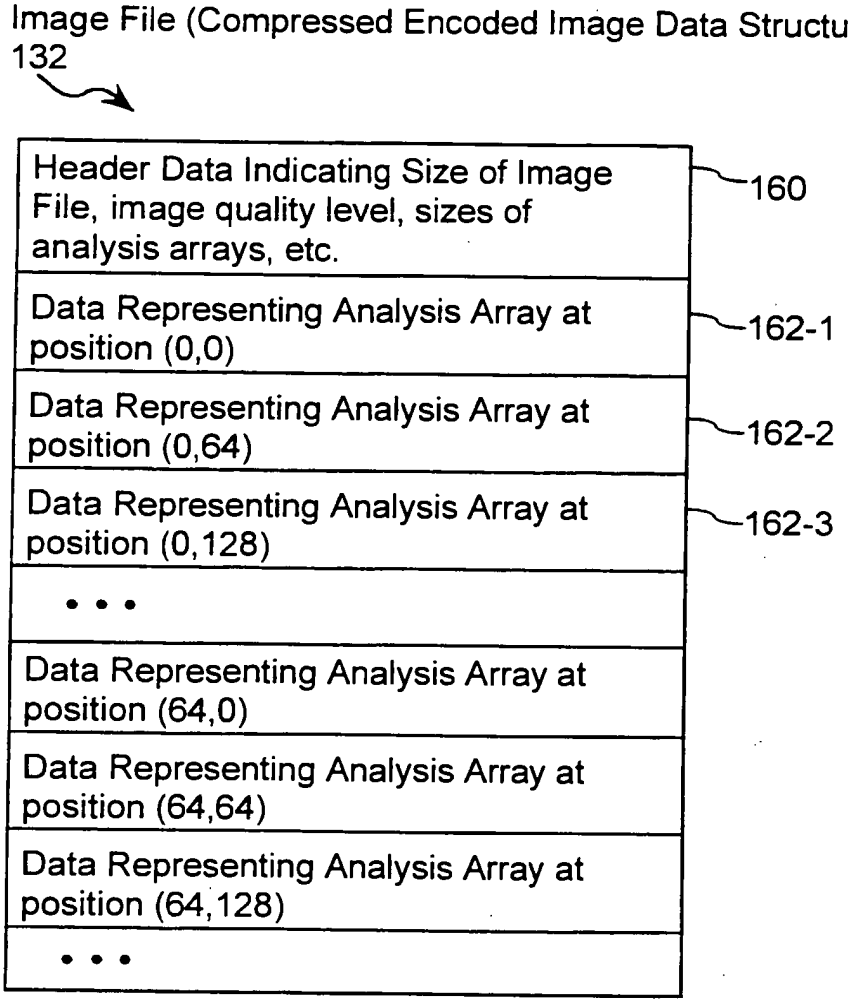


Image File (Compressed Encoded Image Data Structure)
132
Header Data Indicating Size of Image File, image quality level, sizes of analysis arrays, etc.
160
Data Representing Analysis Array at position (0,0)
162-1
Data Representing Analysis Array at position (0,64)
162-2
Data Representing Analysis Array at position (0,128)
162-3
• • •
Data Representing Analysis Array at position (64,0)
Data Representing Analysis Array at position (64,64)
Data Representing Analysis Array at position (64,128)
• • •

FIG. 4A

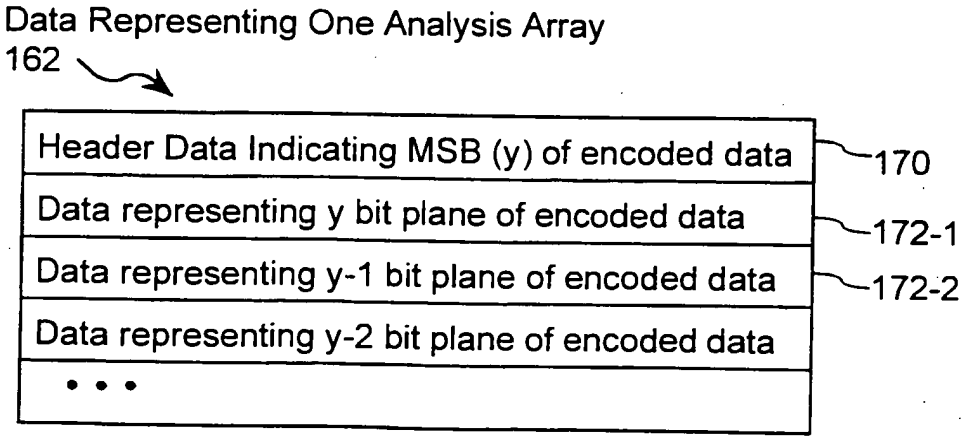

Data Representing One Analysis Array
162
Header Data Indicating MSB (y) of encoded data
170
Data representing y bit plane of encoded data
172-1
Data representing y-1 bit plane of encoded data
172-2
Data representing y-2 bit plane of encoded data
• • •

FIG. 4B

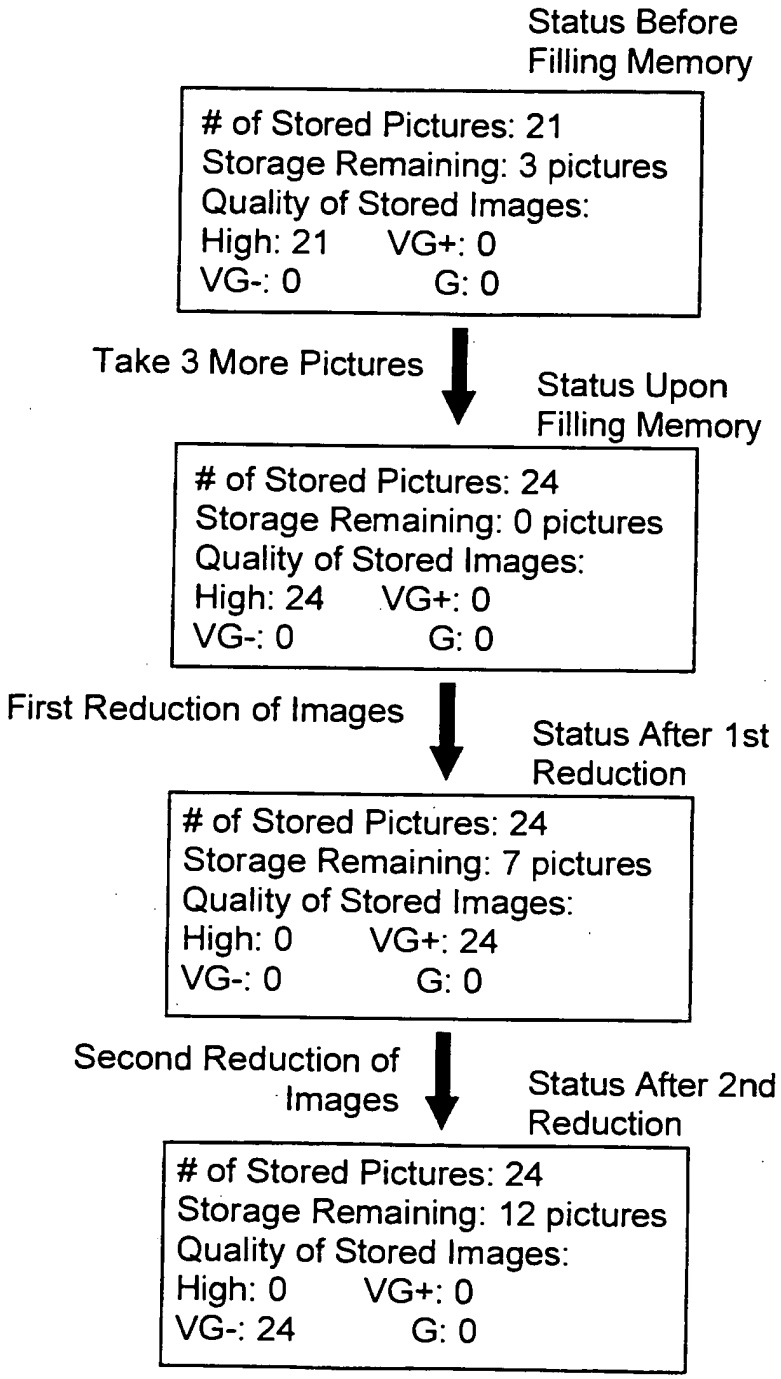

Status Before Filling Memory
# of Stored Pictures: 21
Storage Remaining: 3 pictures
Quality of Stored Images:
High: 21 VG+: 0
VG-: 0 G: 0
Take 3 More Pictures
Status Upon Filling Memory
# of Stored Pictures: 24
Storage Remaining: 0 pictures
Quality of Stored Images:
High: 24 VG+: 0
VG-: 0 G: 0
First Reduction of Images
Status After 1st Reduction
# of Stored Pictures: 24
Storage Remaining: 7 pictures
Quality of Stored Images:
High: 0 VG+: 24
VG-: 0 G: 0
Second Reduction of Images
Status After 2nd Reduction
# of Stored Pictures: 24
Storage Remaining: 12 pictures
Quality of Stored Images:
High: 0 VG+: 0
VG-: 24 G: 0

FIG. 5

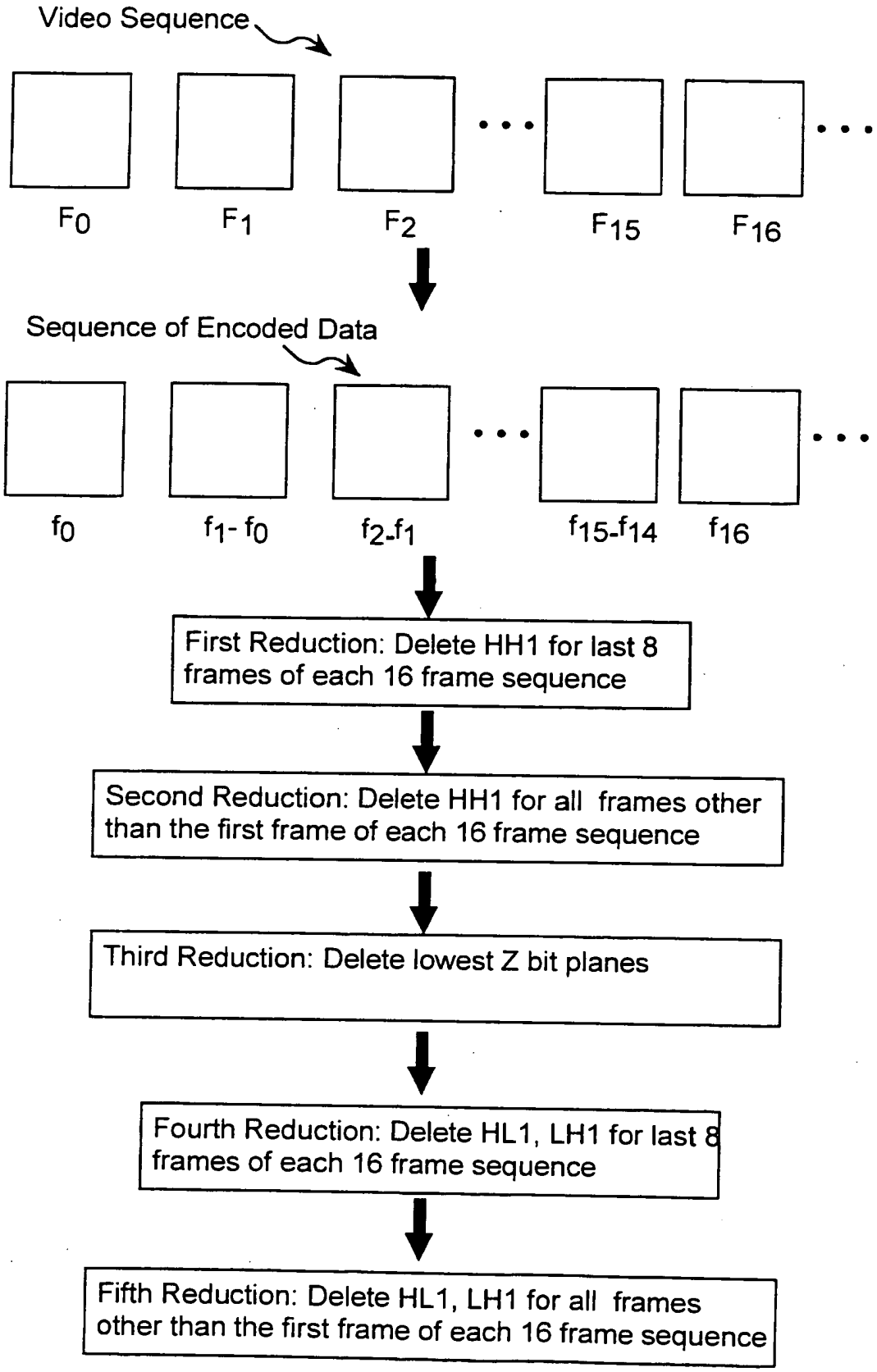

Video Sequence
F0
F1
F2
F15
F16
Sequence of Encoded Data
f0
f1- f0
f2-f1
f15-f14
f16
First Reduction: Delete HH1 for last 8 frames of each 16 frame sequence
Second Reduction: Delete HH1 for all frames other than the first frame of each 16 frame sequence
Third Reduction: Delete lowest Z bit planes
Fourth Reduction: Delete HL1, LH1 for last 8 frames of each 16 frame sequence
Fifth Reduction: Delete HL1, LH1 for all frames other than the first frame of each 16 frame sequence

FIG. 6

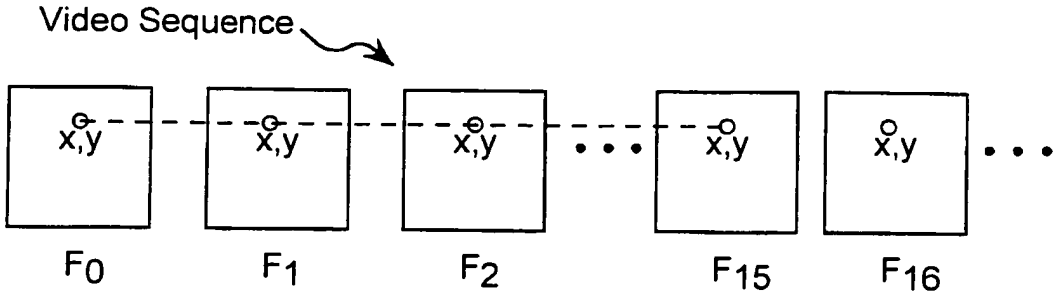


FIG. 7

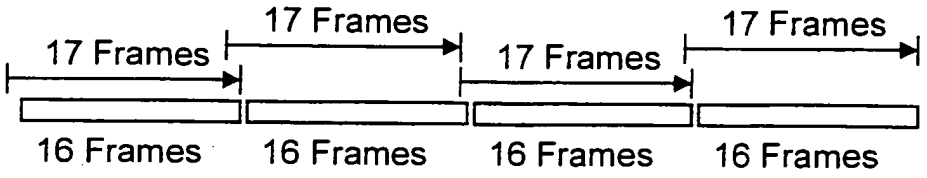


FIG. 8

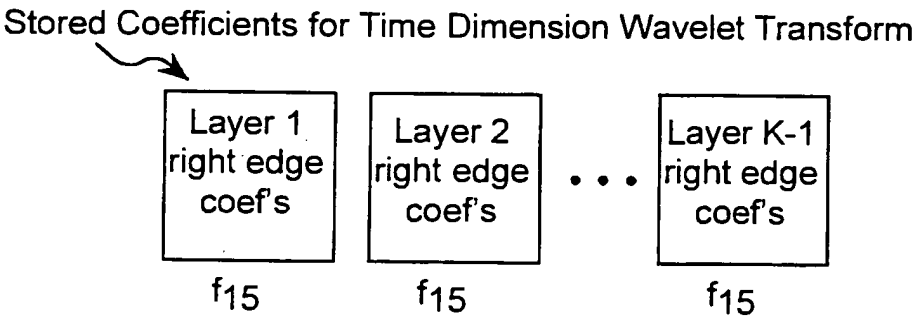


FIG. 9

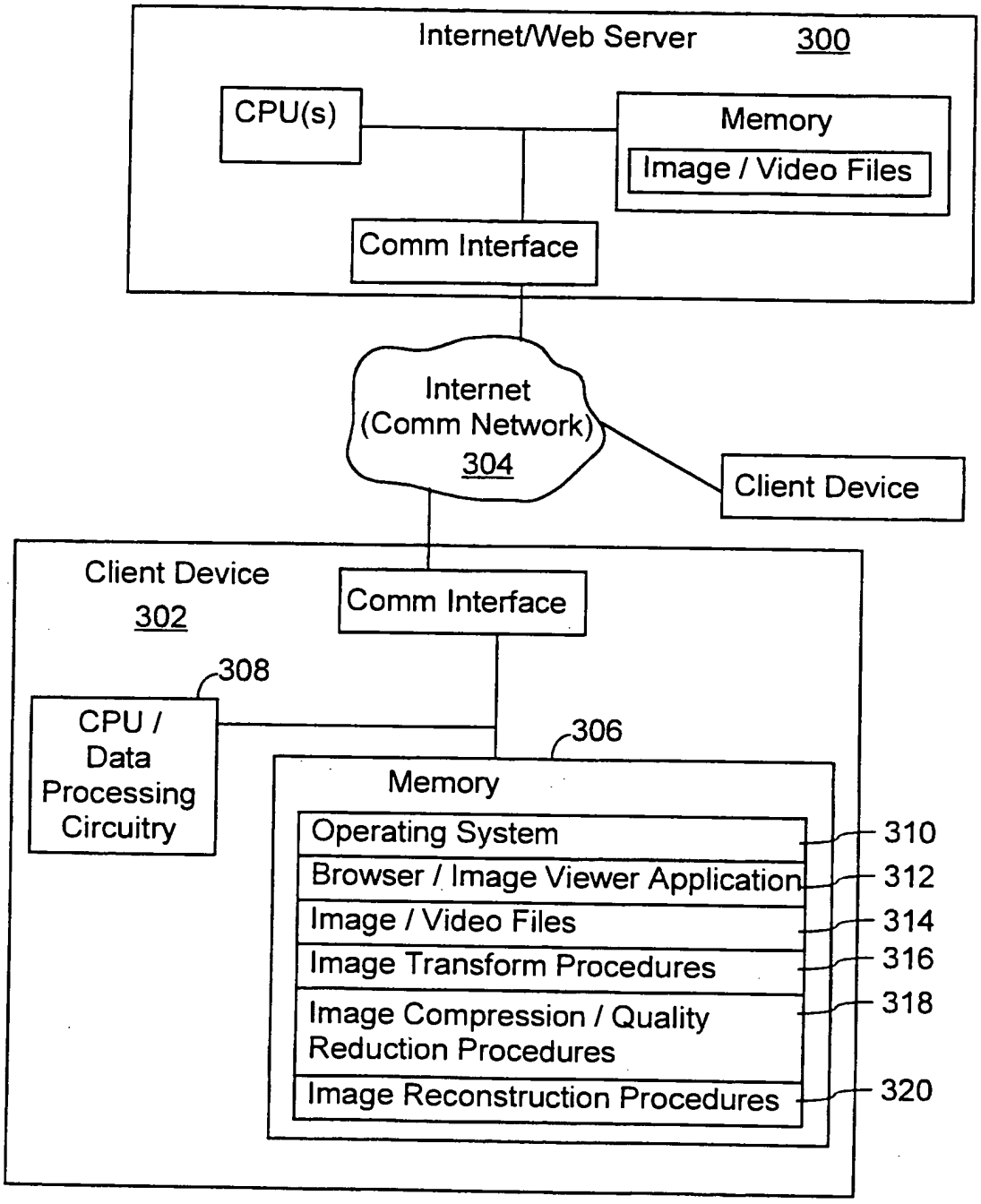

Internet/Web Server 300
CPU(s)
Memory
Image / Video Files
Comm Interface
Internet
(Comm Network)
304
Client Device
Client Device
302
Comm Interface
308
CPU /
Data
Processing
Circuitry
306
Memory
Operating System
Browser / Image Viewer Application
Image / Video Files
Image Transform Procedures
Image Compression / Quality
Reduction Procedures
Image Reconstruction Procedures
310
312
314
316
318
320

FIG. 10

INTERNATIONAL SEARCH REPORT

Interna ..al application No.
PCT/US00/30825

A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) : G06K 9/36, 9/46
US CL : 382/232

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
U.S. : Please See Continuation Sheet

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
WEST

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5,867,602 A (ZANDI ET AL.) 02 February 1999 (02.02.99).	1-75
A	US 5,881,176 A (KEITH ET AL.) 09 March 1999 (09.03.1999).	1-75
A	US 5,966,465 A (KEITH ET AL.) 12 October, 1999 (12.10.1999).	1-75

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
04 January 2001 (04.01.2001)	30 MAR 2001
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703)305-3230	Authorized officer Jose L. Couso Telephone No. (703) 305-8576

Form PCT/ISA/210 (second sheet) (July 1998)

INTERNATIONAL SEARCH REPORT

Inte: ional application No.
PCT/US00/30825

Continuation of B. FIELDS SEARCHED Item 1: 382/232, 233, 234, 240, 244, 246, 247, 248, 260, 263, 264, 276; 341/51, 63, 65, 67, 107; 364/724.011, 724.04, 724.05, 724.13, 724.14, 725.01, 725.02.

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 257 585 A2**

(12) EUROPEAN PATENT APPLICATION

(21) Application number: 87112158.8

(51) Int. Cl.⁴: H04L 9/00

(22) Date of filing: 21.08.87

(30) Priority: 22.08.86 JP 197610/86
22.08.86 JP 197611/86

(43) Date of publication of application:
02.03.88 Bulletin 88/09

(84) Designated Contracting States:
BE DE FR GB

(71) Applicant: NEC CORPORATION
33-1, Shiba 5-chome, Minato-ku
Tokyo 108(JP)

(72) Inventor: Okamoto, Eiji c/o NEC Corporation
33-1, Shiba 5-chome
Minato-ku Tokyo(JP)

(74) Representative: Vossius & Partner
Siebertstrasse 4 P.O. Box 86 07 67
D-8000 München 86(DE)

(54) Key distribution method.

(57) The invention relates to a method of distributing a key for enciphering un unenciphered or plaintext message and for deciphering the enciphered message.

The method comprises the following steps: generating a first random number in a first system (101); generating first key distribution information in the first system (101) by applying a predetermined first transformation to the first random number on the basis of first secret information known only by the first system (101); transmitting the first key distribution information to a second system (102) via a communication channel (103); receiving the first key distribution information in the second system (102); generating a second random number in the second system (102); generating second key distribution information by applying the predetermined first transformation to the second random number on the basis of second secret information known only by the second system (102); transmitting the second key distribution information to the first system (101) via the channel (103); receiving the second key distribution information in the first system (101); and generating an enciphering key in the first system (101) by applying a predetermined second transformation to the second key distribution information on the basis of the first random number and identification information of the second system (102) which is not secret.

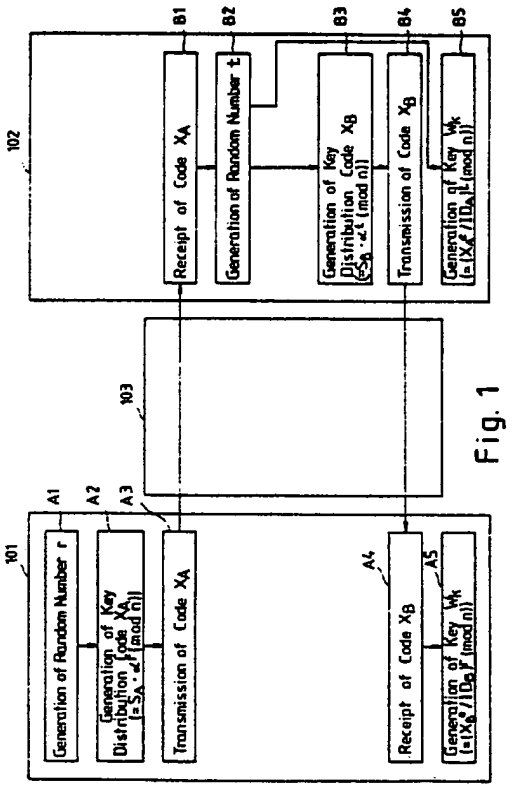


Fig. 1

EP 0 257 585 A2

KEY DISTRIBUTION METHOD

BACKGROUND OF THE INVENTION

The invention relates to a method of distributing a key for enciphering an unenciphered or plaintext message and for deciphering the enciphered message.

A public key distribution method used in a public key cryptosystem as a well-known key distribution method is disclosed in a paper entitled "New Directions in Cryptography" by W. Diffie and M.E. Hellman, published in the IEEE Transactions on Information Theory, Vol. IT-22, No. 6, pp. 644 to 654, November issue, 1976. The key distribution method disclosed in the paper memorizes public information for each of conversers. In the system, before a converser A sends an enciphered message to a converser B, the converser A prepares an enciphering key (which represents a number obtained by calculating $Y_B{}^{X_A} \pmod{p}$) generated from public information Y_B of the converser B and secret information X_A which is kept secret by the converser A. The number p is a large prime number of about 256 bits in binary representation, which is publicly known. a (mod b) means a remainder of division of the number a by the number b. The converser B also prepares the key wk in accordance to $Y_A{}^{X_B} \pmod{p}$ in a similar manner. Y_A and Y_B are selected so as to be equal to $\alpha^{X_A} \pmod{p}$ and $\alpha^{X_B} \pmod{p}$, respectively. As a result, $Y_B{}^{X_A} \pmod{p}$ becomes equal to $Y_A{}^{X_B} \pmod{p}$. It is known that even if Y_A, a and p are known, it is infeasible for anybody except the converser A to obtain X_A which satisfies $Y_A = \alpha^X A \pmod{p}$.

The prior art key distribution system of the type described, however, has disadvantages in that since the system needs a large amount of public information corresponding to respective conversers, the amount of the public information increases as the number of conversers increases. Further, strict control of such information becomes necessary to prevent the information from being tampered.

SUMMARY OF THE INVENTION

An object of the invention is, therefore, to provide a key distribution method free from the above-mentioned disadvantages of the prior art system.

According to an aspect of the invention, there is provided a method which comprises the following steps: generating a first random number in a first system; generating first key distribution information in the first system by applying a predetermined first transformation to the first random number on the basis of first secret information known only by the first system; transmitting the first key distribution information to a second system via a communication channel; receiving the first key distribution information in the second system; generating a second random number in the second system; generating second key distribution information by applying the predetermined first transformation to the second random number on the basis of second secret information known only by the second system; transmitting the second key distribution information to the first system via the channel; receiving the second key distribution information in the first system; and generating an enciphering key in the first system by applying a predetermined second transformation to the second key distribution information on the basis of the first random number and identification information of the second system which is not secret.

According to another aspect of the invention, there is provided a method which comprises the following steps: generating a first random number in the first system; generating first key distribution information by applying a predetermined first transformation to the first random number on the basis of public information in the first system and generating first identification information by applying a predetermined second transformation to the first random number on the basis of first secret information known only by the first system; transmitting the first key distribution information and the first identification information to a second system via a communication channel; receiving the first key distribution information and the first identification information in the second system; examining whether or not the result obtained by applying a predetermined third transformation to the first key distribution information on the basis of the first identification information satisfies a first predetermined condition, and, if it does not satisfy, suspending key distribution processing; generating a second random number if said condition is satisfied in the preceding step; generating second key distribution information by applying the predetermined first transformation to the second random number on the basis of the public information, and generating second identification information by applying the predetermined second transformation to the second random number on the basis of second secret information known only by the second system; transmitting the second key distribution information and the second identification information to the first system via the communication channel; and exam-

ining whether or not the result obtained by applying a third predetermined transformation to the second key distribution information on the basis of the second identification information in the first system satisfies a predetermined second condition, and if the result does not satisfy the second condition, suspending the key distribution processing, or if it satisfies the second condition, generating an enciphering key by applying a fourth predetermined transformation to the first random number on the basis of the second key distribution information.

BRIEF DESCRIPTION OF THE DRAWINGS

Other features and advantages of the invention will become more apparent from the following detailed description when taken in conjunction with the accompanying drawings in which:

FIG. 1 is a block diagram of a first embodiment of the invention;

FIG. 2 is a block diagram of a second embodiment of the invention; and

FIG. 3 is a block diagram of an example of systems 101, 102, 201 and 202.

In the drawings, the same reference numerals represent the same structural elements.

PREFERRED EMBODIMENTS

Referring now to FIG. 1, a first embodiment of the invention comprises a first system 101, a second system 102 and an insecure communication channel 103 such as a telephone line which transmits communication signals between the systems 101 and 102. It is assumed herein that the systems 101 and 102 are used by users or conversers A and B, respectively. The user A has or knows a secret integer number S_A and public integer numbers e, c, α and n which are not necessarily secret while the user B has or knows a secret integer number $S_B$and the public integer numbers. These integer numbers are designated and distributed in advance by a reliable person or organization. The method to designate the integer numbers will be described later.

An operation of the embodiment will next be described on a case in which the user A starts communication. The system 101 of the user A generates a random number γ (Step A1 in FIG. 1) and sends a first key distribution code X_A representative of a number obtained by computing $S_A \cdot \alpha^{\gamma} \pmod{n}$ (Step A2) to the system 102 of the user B (step A3). Next, when the system 102 receives the code X_A(Step B1), it generates a random number t (Step B2), calculates $(X_A{}^e/ID_A)^t \pmod{n}$ (Step B5), and keeps the resulting number as a enciphering key wk for enciphering a message into storage means (not shown). The identification code ID_A represents herein a number obtained by considering as a numeric value a code obtained by encoding the address, the name and so on of the user A. The encoding is, for instance, performed on the basis of the American National Standard Code for Information Interchange. Then, the system 102 transmits to the system 101 of the user A a second key distribution code X_B representative of a number obtained by calculating $S_B \cdot \alpha^t \pmod{n}$ (Steps B3 and B4).

The system 101, on the other hand, receives the code X_B (Step A4), calculates $(X_B{}^e/ID_B)^{\gamma} \pmod{n}$ (Step A5), and keeps the resulting number as the key wk for enciphering a message. The identification code ID_B represents the numbers obtained by considering as a numeric value a code obtained by encoding the name, address, and so on of the user B.

Subsequently, communication between the users A and B will be conducted by transmitting messages enciphered with the enciphering key wk via the channel 103.

The integer numbers S_A, S_B, e, c, α and n are determined as follows. n is assumed to be a product of two sufficiently large prime numbers p and q. For instance, p and q may be 2^{256} or so. e and c are prime numbers which are equal to or less than n, while α is a positive integer number which is equal to or less than n. Further, d is defined as an integer number which satisfies $e.d \pmod{(p-1)\cdot(q-1)} = 1$. S_A and S_B are defined as numbers obtainable from $ID_A{}^d \pmod{n}$ and $ID_B{}^d \pmod{n}$, respectively.

If S_A, S_B, e, c, α, and n are defined as above, ID_A and ID_B become equal to $S_A{}^e \pmod{n}$ and $S_B{}^e \pmod{n}$, respectively. This can be proved from a paper entitled "A Method for Obtaining Digital Signatures and Publick-Key Cryptosystems" by R.L. Rivest et al., published in the Communication of the ACM, Vol. 21, No. 2, pp. 120 to 126. Since the key obtained by $(X_B{}^e/ID_B)^r \pmod{n}$ on the side of the user A becomes equal to $\alpha^{ert} \pmod{n}$ and the key obtained by $(X_A{}^e/ID_A)^t \pmod{n}$ on the side of the user B becomes equal to $\alpha^{ert} \pmod{n}$, they can prepare the same enciphering key. Even if a third party tries to assume the identity of the user A, he cannot prepare the key wk since he cannot find out z which meets $ID_A = Z^e \pmod{n}$.

Referring now to FIG. 2, a second embodiment of the invention comprises a first system 201, a second system 202 and an insecure communication channel 203. It is assumed herein that the systems 201 and 202 are used by users A and B, respectively. The user A has or knows a secret integer number S_A and public integer numbers e, c, α, and n, which are not necessarily secret while

the user B has or knows a secret integer number $S_B$and the public integer numbers. These integer numbers are designated and distributed by a reliable person or organization in advance. The method to designate the integer numbers will be described later.

An operation of the embodiment will next be described on a case where the user A starts communication. The system 201 of the user A generates a random number $\underline{r}$ (Step AA1 in FIG. 2) and determines a first key distribution code X_A representative of a number obtained by computing $\alpha^{e \cdot r} \pmod{\underline{n}}$ as well as a first identification code $Y_A$indicative of a number obtained by computing $S_A \cdot \alpha^{c \cdot r} \pmod{\underline{n}}$ (AA2). The system 201 then transmits a first pair of X_A and Y_A to the system 202 of the user B (Step AA3). Thereafter, the system 202 receives the first pair (X_A , Y_A) (Step BB1), calculates $Y_A^e / X_A^c \pmod{\underline{n}}$, and examines whether or not the number obtained by the calculation is identical to the number indicated by an identification code ID_A obtained by the address, the name and so on of the user A in a similar manner to in the first embodiment (Step BB2). If they are not identical to each other, the system suspends processing of the key distribution (Step BB7). On the other hand, if they are identical to each other, the system 202 generates a random number $\underline{t}$ (Step BB3) and determines a second key distribution code X_B representative of a number obtained by calculating $\alpha^{e \cdot t} \pmod{\underline{n}}$ and a second identification code Y_B obtained by calculating $S_B \cdot \alpha^{c \cdot t} \pmod{\underline{n}}$ (Step BB4). The system 202 then transmits a second pair of X_B and Y_B to the system 201 of the user A (Step BB5). The system 202 calculates $X_A^{\underline{t}} \pmod{\underline{n}}$ and keeps the number thus obtained as a enciphering key wk (Step BB6).

The system 201, on the other hand, receives the second pair (X_B, Y_B) (Step AA4), calculates $Y_B^e / X_B^c \pmod{\underline{n}}$, and examines whether or not the number thus obtained is identical to the number indicated by an identification code ID_B obtained by the address, the name and so on of the user B in a similar manner to in the first embodiment (Step AA5). If they are not identical to each other, the system suspends the key distribution processing (Step AA7). If they are identical to each other, the system 201 calculates $X_B^{\underline{r}} \pmod{\underline{n}}$, and stores the number thus obtained as a enciphering key wk (Step AA6). Although the codes ID_A and ID_B are widely known, they may be informed by the user A to the user B.

The integer numbers S_A, S_B, $\underline{e}$, $\underline{c}$, $\underline{\alpha}$ and $\underline{n}$ are determined in the same manner as in the first embodiment. As a result, ID_A and ID_B becomes equal to $Y_A^e / X_A^c \pmod{\underline{n}}$ ($= S_A^{\underline{e}} \cdot \alpha^{erc} / \alpha^{erc} \pmod{\underline{n}}$) and $Y_B^e / X_B^c \pmod{\underline{n}}$ ($= S_B^{\underline{e}} \cdot \alpha^{etc} / \alpha^{etc} \pmod{\underline{n}}$), respectively. If we presuppose that the above-mentioned reliable person or organization who prepared S_A and S_B do not act illegally, since S_A is possessed only by the user A while S_B is possessed only by the user B, the first pair (x_A, y_A) which satisfies $y_A^e / x_A^c \pmod{\underline{n}} = ID_A$ can be prepared only by the user A while the second pair (x_B , y_B) which satisfies $y_B^e / x_B^c \pmod{\underline{n}} = ID_B$ can be prepared only by the user B. It is impossible to find out a number $\underline{x}$ which satisfies $\underline{x}^{\underline{f}} \pmod{\underline{n}} = \underline{b}$ on the basis of $\underline{f}$, $\underline{b}$and $\underline{n}$ since finding out $\underline{X}$ is equivalent to breaking the RSA public key cryptogram system disclosed in the above-mentioned the Communication of the ACM. It is described in the above-referenced IEEE Transactions on Information Theory that the key wk cannot be calculated from the codes x_A or x_B and $\underline{n}$. The key distribution may be implemented similarly by making the integer number $\underline{C}$ variable and sending it from a user to another.

An example of the systems 101, 102, 201 and 202 to be used in the first and second embodiments will next be described referring to FIG. 3.

Referring now to FIG. 3, a system comprises a terminal unit (TMU) 301 such as a personal computer equipped with communication processing functions, a read only memory unit (ROM) 302, a random access memory unit (RAM) 303, a random number generator (RNG) 304, a signal processor (SP) 306, and a common bus 305 which interconnects the TMU 301, the ROM 302, the RAM 303, the RNG 304 and the SP 306.

The RNG 304 may be a key source 25 disclosed in U.S. Patent No. 4,200,700. The SP 306 may be a processor available from CYLINK Corporation under the trade name CY 1024 KEY MANAGEMENT PROCESSOR.

The RNG 304 generates random numbers $\underline{r}$ or $\underline{t}$ by a command given from the SP 306. The ROM 407 stores the public integer numbers $\underline{e}$, $\underline{c}$, $\underline{\alpha}$, $\underline{n}$ and the secret integer number S_A (if the ROM 407 is used in the system 101 or 201) or the secret integer number S_B (if the ROM 407 is used in the system 102 or 202). The numbers S_A and S_B may be stored in the RAM 303 from the TMU 301 everytime users communicates. According to a program stored in the ROM 407, the SP 306 executes the above-mentioned steps A2, A5, AA2, AA5, AA6 and AA7 (if the SP 306 is used in the system 101 or 201), or the steps B3, B5, BB2, BB4, BB6 and BB7 (if the SP 306 is used in the system 102 or 202). The RAM 303 is used to temporarily store calculation results in these steps.

Each of the systems 101, 102, 201 and 202 may be a data processing unit such as a general purpose computer and an IC (integrated circuit) card.

As described in detail hereinabove, this invention enables users to effectively implement key distribution simply with a secret piece of information and several public pieces of information.

While this invention has thus been described in conjunction with the preferred embodiments thereof, it will now readily be possible for those skilled in the art to put this invention into practice in various other manners.

Claims

1. A key distribution method comprising the following steps:

a) generating a first random number in a first system;

b) generating first key distribution information in said first system by applying a predetermined first transformation to said first random number on the basis of first secret information known only by said first system;

c) transmitting said first key distribution information to a second system via a communication channel;

d) receiving said first key distribution information in said second system;

e) generating a second random number in said second system;

f) generating second key distribution information by applying said predetermined first transformation to said second random number on the basis of second secret information known only by said second system;

g) transmitting said second key distribution information to said first system via said channel;

h) receiving said second key distribution information in said first system; and

i) generating an enciphering key in said first system by applying a predetermined second transformation to said second key distribution information on the basis of said first random number and identification information of said second system which is not secret.

2. A key distribution method as claimed in Claim 1, in which said first system includes first data processing means for executing said steps a), b) and i), and first communication processing means for executing said steps c) and h).

3. A key distribution method as claimed in Claim 1 or 2, in which said second system includes second data processing means for executing said steps e) and f), and second communication processing means for executing said steps d) and g).

4. A key distribution method comprising the following steps:

a) generating a first random number in a first system;

b) generating first key distribution information in said first system by applying a predetermined first transformation to said first random number on the basis of public information and generating first identification information by applying a predetermined second transformation to said first random number on the basis of first secret information known only by said first system;

c) transmitting said first key distribution information and said first identification information to a second system via a communication channel;

d) receiving said first key distribution information and said first identification information in said second system;

e) examining whether or not the result obtained by applying a predetermined third transformation to said first key distribution information on the basis of said first identification information satisfies a predetermined first condition and, if it does not satisfy, suspending key distribution processing;

f) generating a second random number if said first condition is satisfied at said step e);

g) generating second key distribution information by applying said predetermined first transformation to said second random number on the basis of said public information, and generating second identification information by applying said predetermined second transformation to said second random number on the basis of second secret information known only by said second system;

h) transmitting said second key distribution information and said second identification information to said first system via said communication channel; and

i) examining in said first system whether or not the result obtained by applying a predetermined third transformation to said second key distribution information on the basis of said second identification information satisfies a predetermined second condition and, if the result does not satisfy said second condition, suspending said key distribution processing or, if it satisfies said second condition, generating said enciphering key by applying a predetermined fourth transformation to said first random number on the basis of said second key distribution information.

5. A key distribution method as claimed in Claim 4, in which said first system includes first data processing means for executing said steps a), b) and i), and first communication processing means for executing said step c).

6. A key distribution method as claimed in Claim 4 or 5, in which said second system includes second data processing means for executing said steps e), f) and g), and second communication processing means for executing said steps d) and h).

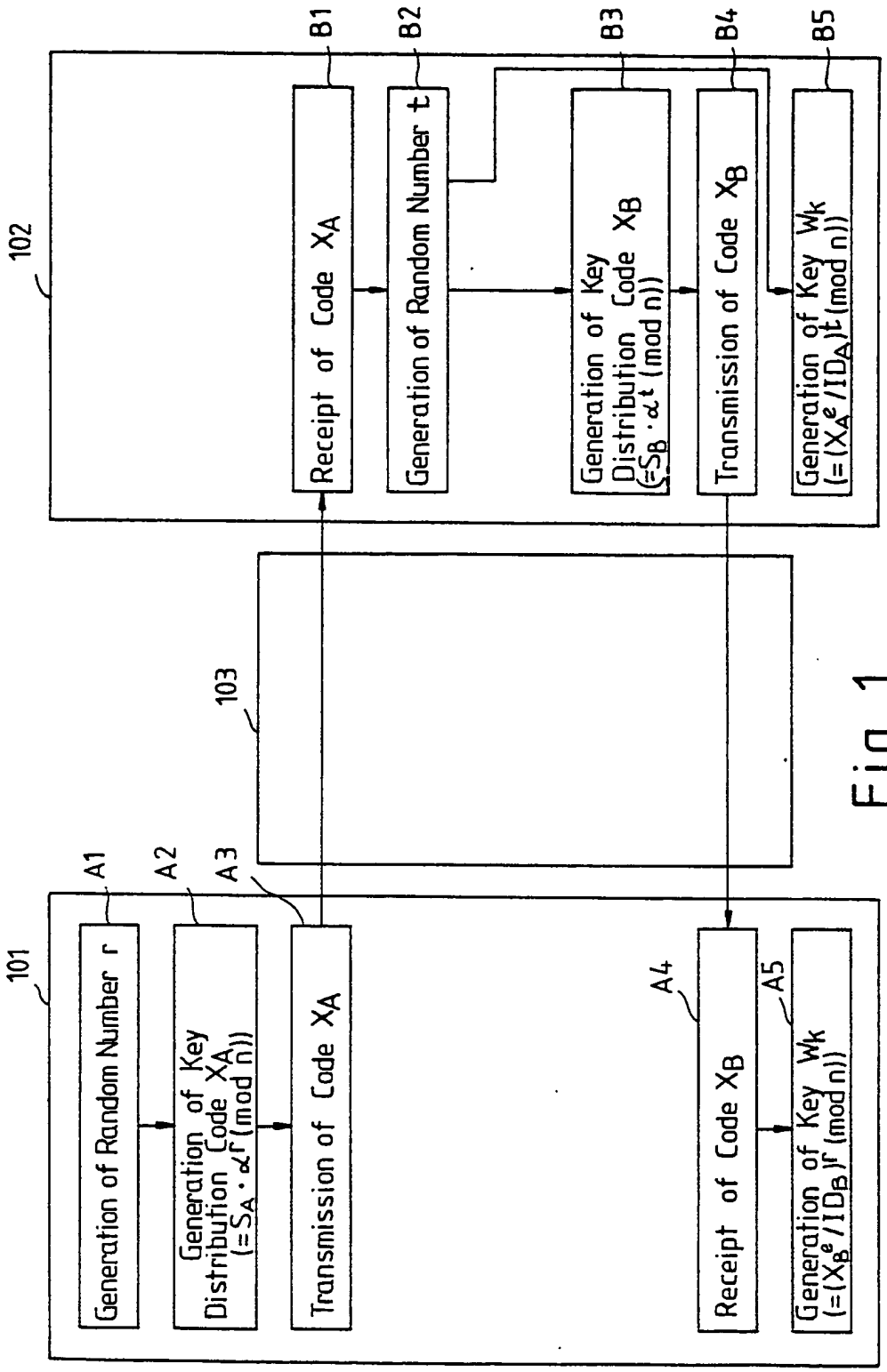

102
B1
B2
B3
B4
B5
Receipt of Code X_A
Generation of Random Number t
Generation of Key Distribution Code X_B ($=S_B \cdot \alpha^t \pmod n$)
Transmission of Code X_B
Generation of Key W_k ($=(X_A^e/ID_A)^t \pmod n$)
103
101
A1
A2
A3
A4
A5
Generation of Random Number r
Generation of Key Distribution Code X_A ($=S_A \cdot \alpha^r \pmod n$)
Transmission of Code X_A
Receipt of Code X_B
Generation of Key W_k ($=(X_B^e/ID_B)^r \pmod n$)

Fig. 1

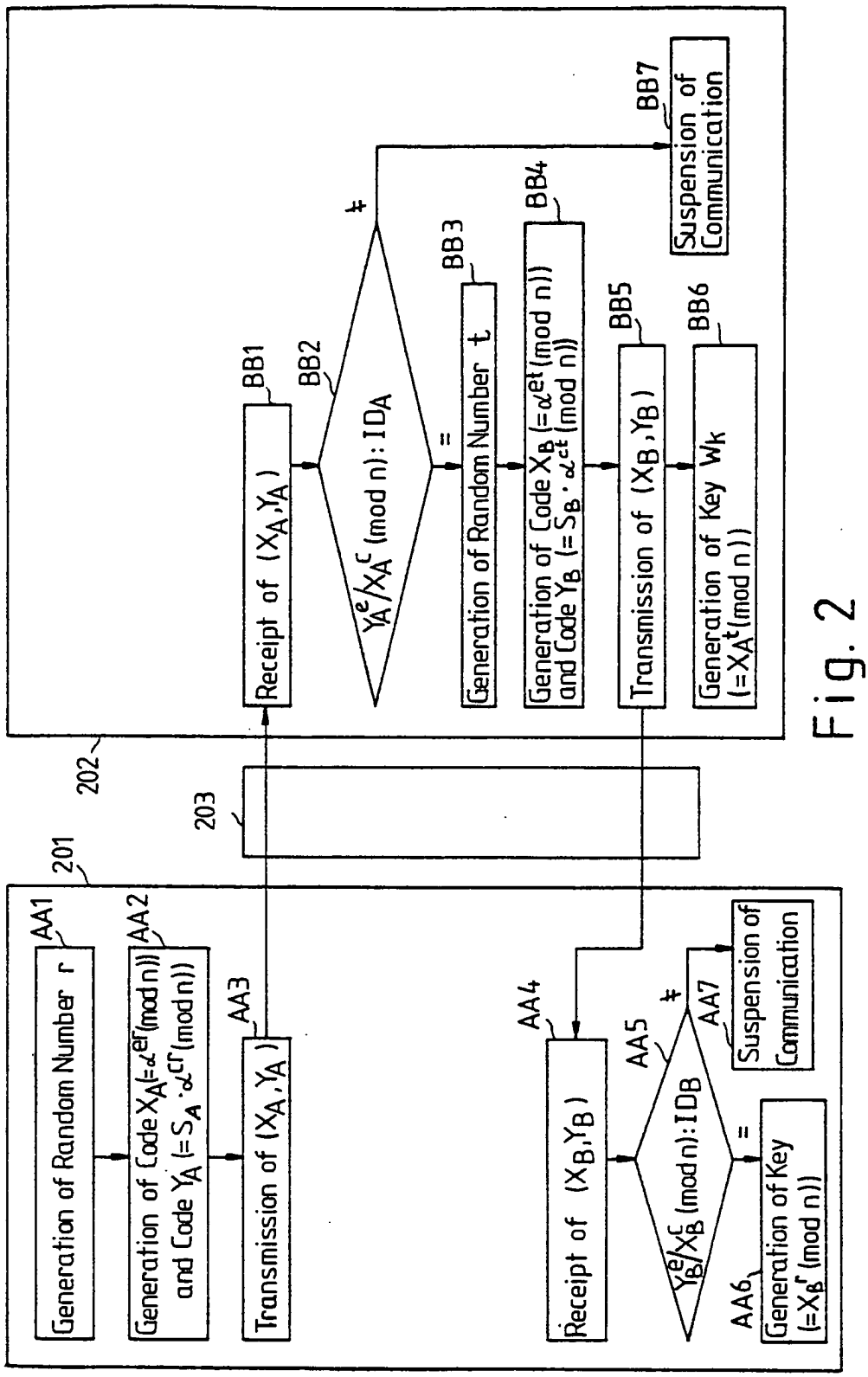

201
202
203
AA1
Generation of Random Number r
AA2
Generation of Code $X_A(=\alpha^{er} \pmod n)$ and Code $Y_A(=S_A \cdot \alpha^{cr} \pmod n)$
AA3
Transmission of (X_A, Y_A)
AA4
Receipt of (X_B, Y_B)
AA5
$Y_B^e/X_B^c \pmod n : ID_B$
AA6
Generation of Key $(=X_B^r \pmod n)$
AA7
Suspension of Communication
BB1
Receipt of (X_A, Y_A)
BB2
$Y_A^e/X_A^c \pmod n : ID_A$
BB3
Generation of Random Number t
BB4
Generation of Code $X_B(=\alpha^{et} \pmod n)$ and Code $Y_B(=S_B \cdot \alpha^{ct} \pmod n)$
BB5
Transmission of (X_B, Y_B)
BB6
Generation of Key W_k $(=X_A^t \pmod n)$
BB7
Suspension of Communication

Fig. 2

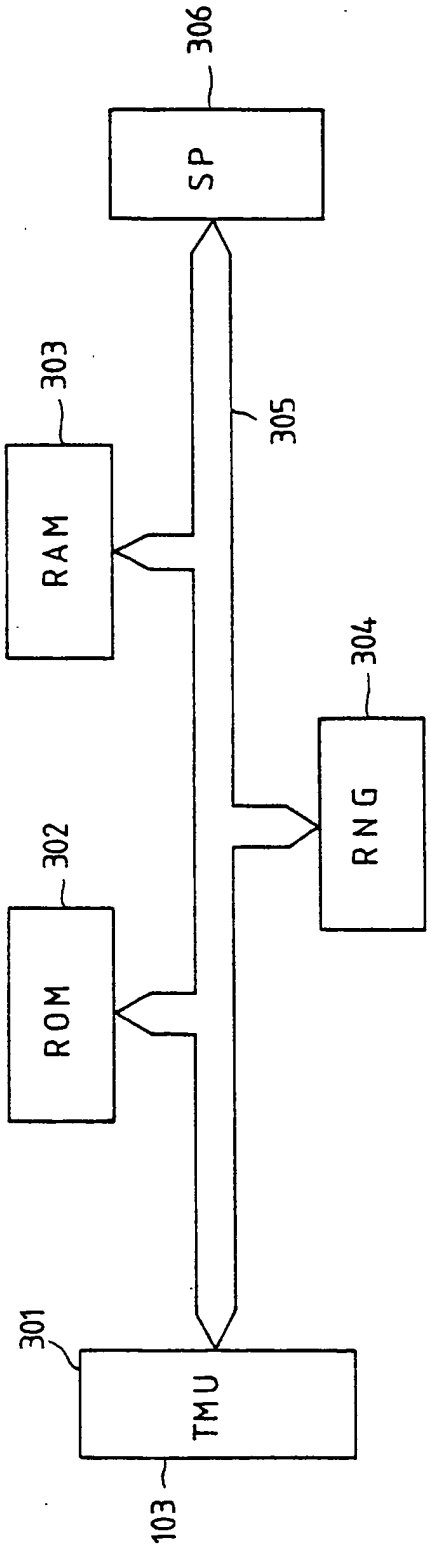


Fig. 3

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 332 304 A2**

(12) EUROPEAN PATENT APPLICATION

(21) Application number: 89301510.7

(51) Int. Cl.[4]: G06F 1/00

(22) Date of filing: 16.02.89

(30) Priority: 07.03.88 US 164944

(43) Date of publication of application:
13.09.89 Bulletin 89/37

(84) Designated Contracting States:
DE FR GB

(71) Applicant: DIGITAL EQUIPMENT CORPORATION
111 Powdermill Road
Maynard Massachusetts 01754-1418(US)

(72) Inventor: Robert, Gregory
12 Carson Circle
Nashua New Hampshire 03062(US)
Inventor: Chase, David
28 Bay View Road
Wellesley Massachusetts 02181(US)
Inventor: Schaefer, Ronald
7 Gioconda Avenue
Acton Massachusetts 01720(US)

(74) Representative: Goodman, Christopher et al
Eric Potter & Clarkson 14 Oxford Street
Nottingham NG1 5BP(GB)

(54) Software licensing management system.

(57) A license management system which includes a license management facility that determines whether usage of a licensed program is within the scope of the license. The license management system maintains a license unit value for each licensed program and a pointer to a table identifying an allocation unit value associated with each use of the licensed program. In response to a request to use a licensed program, the license management system responds with an indication as to whether the license unit value exceeds the allocation unit value associated with the use. Upon receiving the response, the operation of the licensed program depends upon policies established by the licensor.

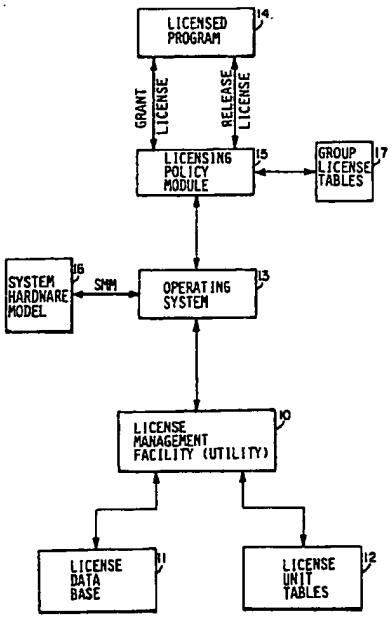


FIG. 1

EP 0 332 304 A2

SOFTWARE LICENSING MANAGEMENT SYSTEM

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention relates generally to the field of digital data processing systems, and more specifically to a system for managing licensing for, and usage of, the various software programs which may be processed by the systems to ensure that the software programs are used within the terms of the software licenses.

2. Description of the Prior Art

A digital data processing system includes three basic elements, namely, a processor element, a memory element and an input/output element. The memory element stores information in addressable storage locations. This information includes data and instructions for processing the data. The processor element fetches information from the memory element, interprets the information as either an instruction or data, processes the data in accordance with the instructions, and returns the processed data to the memory element for storage therein. The input/output element, under control of the processor element, also communicates with the memory element to transfer information, including instructions and data to be processed, to the memory, and to obtain processed data from the memory.

Typically, an input/output element includes a number of diverse types of units, including video display terminals, printers, interfaces to the public telecommunications network, and secondary storage subsystems, including disk and tape storage devices. A video display terminal permits a user to run programs and input data and view processed data. A printer permits a user to obtain a processed data on paper. An interface to the public telecommunications network permits transfer of information over the public telecommunications network.

The instructions processed by the processor element are typically organized into software programs. Recently, generation and sales of software programs have become significant businesses both for companies which are primarily vendors of hardware, as well as for companies which vend software alone. Software is typically sold under license, that is, vendors transfer copies of software to users under a license which govems how the users may use the software. Typically, software costs are predicated on some belief as to the amount of usage which the software program may provide and the economic benefits, such as cost saving which may otherwise be incurred, which the software may provide to the users. Thus, license fees may be based on the power of the processor or the number of processors in the system, or the number of individual nodes in a network, since these factors provide measures of the number of users which may use the software at any given time.

In many cases, however, it may also be desirable, for example, to have licenses and license fees more closely relate to the actual numbers of users which can use the program at any given time or on the actual use to which a program may be put. Furthermore, it may be desirable to limit the use of the program to specified time periods. A problem arises particularly in digital data processing systems which have multiple users and/or multiple processors, namely, managing use of licensed software to ensure that the use is within the terms of the license, that is, to ensure that the software is only used on identified processors or by the numbers of users permitted by the license.

SUMMARY OF THE INVENTION

The invention provides a new and improved licensing management system for managing the use of licensed software in a digital data processing system.

In brief summary, the license management system includes a license management facility and a licensing policy module that jointly determine whether a licensed program may be operated. The license management facility maintains a license unit value for each licensed program and a pointer to a table identifying a license usage allocation unit value associated with usage of the licensed program. In response to a request to use a licensed program, the license management facility determines whether the remaining license unit value exceeds the license usage allocation unit value associated with the use. If the license unit value does not exceed the license usage allocation unit value, the license management facility permits usage of the licensed program and adjusts the license unit value by a function of the license usage allocation unit value to reflect the usage. On the other hand, if the license unit value associated with use of the license program does exceed the li-

cense usage allocation unit value, the licensing policy module determines whether to allow the licensed program to be used in response to other licensing policy factors.

BRIEF DESCRIPTION OF THE DRAWINGS

This invention is pointed out with particularity in the appended claims. The above and further advantages of this invention may be better understood by referring to the following description taken in conjunction with the accompanying drawings, in which:

Fig. 1 is a general block diagram of a new system in accordance with the invention;

Figs. 2 and 3 are diagrams of data structures useful in understanding the detailed operation of the system depicted in Fig. 1; and

Figs. 4A-1 through 4B-2 are flow diagrams which are useful in understanding the detailed operations of the system depicted in Fig. 1.

DETAILED DESCRIPTION OF AN ILLUSTRATIVE EMBODIMENT

Fig. 1 depicts a general block diagram of a system in accordance with the invention for use in connection with a digital data processing system which assists in managing software use in accordance with software licenses. With reference to Fig. 1, the new system includes a license management facility 10 which operates in conjunction with a license data base 11 and license unit tables 12, and under control of an operating system 13 and licensing policy module 15 to control use of licensed programs, such as licensed program 14, so that the use is in accordance with the terms of the software license which controls the use of the software program on a system 16 identified by a system marketing model (SMM) code in a digital data processing system.

As is conventional, the digital data processing system including the licensing management system may include one or more systems 16, each including one or more processors, memories and input/output units, interconnected in a number of ways. For example, the digital data processing system may comprise one processor, which may include a central processor unit which controls the system and one or more auxiliary processors which assist the central processor unit. Alternatively, the digital data processing system may comprise multiple processing systems, in which multiple central processor units are tightly coupled, or clustered or networked systems in which multiple central processor units are loosely coupled, generally operating relatively autonomously, interacting by means of messages transmitted over a cluster or network connection. In a tightly coupled multiple processing system, for example, it may be desirable to control the number of users which may use a particular software program at one time. A similar restriction may be obtained in a cluster or network environment by controlling the number of particular nodes, that is, connections to the communications link in the cluster or network over which messages are transferred. In addition, since the diverse processors which may be included in a digital data processing system may have diverse processing speeds and powers, represented by differing system marketing model (SMM) codes, it may be desirable to include a factor for speeds and power in determining the number of processors on which a program may be used concurrently.

As will be explained in greater detail below, the license data base 11 contains a plurality of entries 20 (described below in connection with Fig. 2) each containing information relating to the terms of the license for a particular licensed program 14. In one embodiment such information may include a termination date, if the license is for a particular time period or expires on a particular date, and a number of licensing units if the license is limited by usage of the license program. In that embodiment, the entry also includes identification of a license unit table 40 (described below in connection with Fig. 3) in the license unit tables 12 that identifies the number of allocation units for usage of the licensed program on the types of systems 16 which may be used in the digital data processing system as represented by the system marketing model (SMM) codes.

When a user wishes to use a licensed program 14, a GRANT LICENSE request message is generated which requests information as to the licensing status of the licensed program 14. The GRANT LICENSE request message is transmitted to the licensing policy module 15, which notifies the operating system of the request. The operating system 13, in turn, passes the request, along with the system marketing model of the specific system 16 being used by the user, to the license management facility 10 which determines whether use of the program is permitted under the license.

In response to the receipt of the GRANT LICENSE request from the user and the system marketing model (SMM) code of the system 16 being used by the user on which the licensed program will be processed, the license management facility 10 obtains from the license data base the entry 20 associated with the licensed program

14 and determines whether the use of the licensed program 14 is within the terms of the license as indicated by the information in the license data base 11 and the license unit tables 12.

In particular, the license management facility 10 retrieves the contents of the entry 20 associated with the licensed program. If the entry 20 indicates a termination data, the license management facility 10 compares the system data, which is maintained by the digital data processing system in a conventional manner, with the termination date identified in the entry. If the system date is after the termination date identified in the entry 20, the license has expired and the license management facility 10 generates a usage disapproved message, which it transmits to the operating system 13. On the other hand, if the termination date indicated in the entry 20 is after the system date, the license has not expired and the license management facility 10 proceeds to determine whether the usage of the licensed program 14 is permitted under other terms of the license which may be embodied in the entry 20.

In particular, the license management facility 10 then determines whether the usage of the licensed program is permitted under usage limitations. In that operation, the license management facility obtains the number of license units remaining, which indicates usage of the licensed program 14 not including the usage requested by the user, as well the identification of the table 40 in license unit tables 12 associated with the licensed program 14. The license management facility 10 then compares the number of license units which would be allocated for use of the licensed program 14, which it obtains from the table 40 identified by entry 20 in the license data base 11, and the number of remaining units to determine whether sufficient license units remain to permit usage of the licensed program 14.

If the number of remaining license units indicated by entry 20 in the license data base 11 exceeds the number, from license unit tables 12, of license units which would be allocated for use of the licensed progam 14, the usage of the licensed program is permitted under the license. Accordingly, the license management facility transmits a usage approved response to the operating system 13. In addition, the license management facility 10 adjusts the number of remaining license units in entry 20 by a function of the license units allocated to use of the licensed program to reflect the usage.

On the other hand, if the number of remaining license units indicated by entry 20 in the license data base is less than the number of license units which would be allocated for use of the licensed program 14, the usage of the licensed program 14 is not permitted by the license. In that case, the license management facility 10 transmits a usage disapproved response to the operating system 13. In addition, the license management facility 10 may also log the usage disapproved response; this information may be used by a system operator to determine whether usage of the licensed program 14 is such as to warrant obtaining an enlarged license.

Upon receipt of either a usage approved response or a usage disapproved response to the GRANT LICENSE request, the operating system 13 passes the response to the licensing policy module 15. If a usage approved response is received, the licensing policy module normally allows usage of the licensed program 14. If a usage disapproved response is received, the licensing policy module determines whether the usage of the licensed program may be permitted for other reasons. For example, usage of the licensed program 14 may be permitted under a group license, whose terms are embodied in entries in group license tables 17. Under a group license, usage may be permitted of any of a group of licensed programs. The operations to determine to whether usage is permitted may be performed in the same manner as described above in connection with license management facility 10. In addition, if the usage of the licensed program 14 is not permitted under a group license, usage may nonetheless be permitted under the licensor's licensing practices, which may be embodied in the licensing policy module 15. If the licensing policy module determines that usage of the program should be permitted, notwithstanding a usage disapproved response from the license management facility 10, because the usage is permitted under a group license or the licensor's licensing practices, the licensing policy module 15 permits usage of the licensed program. Otherwise, the licensing policy module does not permit usage of the licensed program in response to the GRANT LICENSE request.

When a user no longer requires use of a licensed program 14, it transmits a RELEASE LICENSE request to the licensing policy module 15. The operations performed by the licensing policy module depend on the basis for permitting usage of the licensed program. If usage was permitted as a result of a group license, if the group license is limited by usage, the licensing policy module 15, if necessary, adjusts the records in the group license tables 17 related to the group license to reflect the fact that the licensed program 14 related to the group license is not being used. If the usage was permitted as a result of a group license which is not limited by usage, but instead is limited in duration, or if the usage was permitted in response to the licensor's licensing policies, the licensing policy module 15 need do nothing. If the licensing

policy module 15 maintains a log of usage outside the scope of a group or program license, it may make an entry in the log of the RELEASE request.

Finally, if usage was permitted as a result of the license management facility 10 providing an approve usage response to the GRANT LICENSE request, the licensing policy module 15 transmits the RELEASE LICENSE request to the operating system 13. In response, the operating system 13 transfers the RELEASE LICENSE request to the license management facility 10, along with an identification of the system 16 using the licensed program 14. The license management facility 10 then obtains from the license data base the identification of the appropriate license usage allocation unit value table in license unit tables 12, and determines the number of allocation units associated with this use of the licensed program 14 based on the identified allocation table and the processor. The license management facility 10 then adjusts the number of license units for the licensed program 14 in the license data base 11 to reflect the release.

It will be appreciated by those skilled in the art that, the license management facility 10 may, in response to a GRANT LICENSE request, instead of deducting allocation units from the entries in the license data base 11 associated with the licensed programs 14, determine the number of allocation units which would be in use if usage of the licensed program 14 is permitted, and respond based on that determination. If the license mangement facility 10 operates in that manner, it may be advantageous for the entries in license data base 11 relating to each licensed program 14 to maintain a running record of the number of allocation units associated with its usage. The licensing policy module 15 may operate similarly in connection with group licenses that are limited by usage.

It will also be appreciated that the new license management system thus permits the digital data processing system to control use of a licensed program 14 based on licensing criteria in the license data base 11, the license unit tables 12, the group licensing tables 17 and the licensor's general licensing policies rather than requiring an operator to limit or restrict use of a licensed program or charging for the license based on some function of the capacity of all of the processors in the digital data processing system. The new license management system allows for very flexible pricing of licenses and licensing policies, since the digital data processing system itself enforces the licensing terms controlling use of the licensed programs 14 in the system.

Fig. 2 depicts the detailed structure of the license data base 12 (Fig. 1) used in the license management system depicted in Fig. 1. With reference to Fig. 2, the license data base includes a plurality of entries generally identified by reference numberal 20, with each entry being associated with one licensed program 14. Each entry 20 includes a number of fields, including an issuer name field 21 identifying the issuer of the license, an authorization number field 22 which contains an authorization number, a producer name field 23 which identifies the name of the vendor of the licensed program, and a product name field 24 which contains the name of the licensed program. The contents of these fields may be used, for example, in connection with other license management operations, such as determining the source of licensed programs in the event of detection of errors in programs, and in locating duplicate entries in the license data base or entries which may be combined as a result of licenses being obtained and entered by, perhaps different operators or at different times.

Each entry 20 in the licensing data base 11 also includes a license number field 25 whose contents identify the number of licensing units remaining. A license of a licensed program 14 identifies a number of licensing units, which may be a function of the price paid for the license. An availability table field 26 and an activity table field 27 identify license usage allocation unit value tables in the license unit tables 12 (described in connection with Fig. 3) to be used in connection with the GRANT LICENSE and RELEASE LICENSE requests.

By way of background, a license may be in accordance with a licensing paradigm which requires concurrent use of the licensed program 14 on several processors to be a function of the processor power and capacity, and the availability table field 26 identifies a license usage allocation unit table to be used in connection with that. In an alternative, a license may be in accordance with a licensing paradigm which requires concurrent use of the licensed program to be a function of the number of users using the program, and the activity table field 27 identifies a license usage allocation unit valve table in the license unit tables 12 to be used in connection with that. If either licensing paradigm is used to the exclusion of the other, one field contains a non-zero value and the other field contains a zero value. In addition, a license may be in accordance with both licensing paradigms, that is, concurrent use of a program may be limited by both processor power and capacity and by the number of concurrent users, and in that case both fields 26 and 27 have non-zero values.

In one embodiment of the licensing management system, fields 21 through 27 of an entry 20 in the licensing data base 11 are required. In that embodiment, an entry 20 in the licensing data may

also have several optional fields. In particular, an entry 20 may include a date/version number field 30 whose contents comprise either a date or version number to identify the licensed program. If a license is to terminate on a specific date, the entry 20 may include a licensor termination date field 31 or a licensee termination date field 32 whose contents specify the termination date assigned by the licensor or licensee. This may be particularly useful, for example, as a mechanism for permitting licensees to demonstrate or try a program before committing to a long or open term license.

Finally, an entry 20 in the license data base includes a checksum field 33, which includes a checksum of the contents of the other fields 21 through 27 and 30 through 32 in the entry 20, which may be established by means of a mathematical algorithm applied to the contents of the various fields. The general mechanism for establishing checksums is well known in the art, and will not be described further herein. The contents of all fields 21 through 27 and 30 through 33 of a new entry 20 are entered by an operator. Prior to establishment of an entry in the license data base 11, the license management facility 10 may verify correct entry of the information in the various fields by calculating a checksum and comparing it to the checksum provided by the operator. If the checksum provided by the operator and the checksum determined by the license management facility are the same, the entry 20 is established in the license data base 11. On the other hand, if the checksum provided by the operator and the checksum determined by the license management facility differ, the license management facility 10 determines that the information is erroneous or the license is invalid and does not establish the entry 20 in the license data base 11. It will be appreciated that, if the checksum-generation algorithm is hidden from an operator, the checksum provides a mechanism for verifying, not only that the information has been properly loaded into the entry, but also that the license upon which the entry is based is authorized by the licensor.

The structure of group license tables 17 may be similar to the structure of the license data base 11, with the addition that the entries for each license reflected in the group license tables 17 will need to identify all of the licensed programs covered thereby.

As described above, the licensing unit tables 12 (Fig. 1) contain information as to the allocation units for use in determining the number of licensing units associated with use of a licensed program. The structure of a licensing unit table 40 is depicted in Fig. 3. With reference to Fig. 3, the licensing unit table includes a plurality of entries 41(1) through 41(N) (generally identified by reference numeral 41) each identified by a particular type of processor. One entry 41 in the table 40 is provided for each type of processor which can be included in the digital data processing system which can use the licensed programs 14 which reference the license unit table 40. The processor associated with each entry is identified by a processor identification field 42. The successive fields in the entries 41 (which form the various columns in the table 40 depicted in Fig. 3) form license usage allocation unit value tables 43(1) through 43-(M) (generally identified by reference numeral 43). The contents of the availability table field 26 and the activity table field 27 identify a license usage allocation unit value table 43. If there are non-zero contents in both availability field 26 and activity field 27, the contents which identify be the same license usage allocation unit value table 43 or different license usage allocation unit value tables 43. As described above, the contents of the license usage allocation unit value table identify the number of licensing units associated with use of the licensed programs which identify the particular license usage allocation unit value table, for each of the identified processors.

The operation of the licensing management system is depicted in detail in Figs. 4A-1 through 4-B. Figs. 4A-1 through 4A-4 depict, in a number of steps the details of operation of the licensing management system in connection with the GRANT LICENSE request from a licensed program 14. Figs. 4B-1 and 4B-2 depict, in a number of steps, the details of operation in connection with the RELEASE LICENSE request from a licensed program 14. In the Figs., the particular steps performed by the licensing policy module 15, the license management facility 10 and the operating system 13 are indicated in the respective steps. Since the operations depicted in Figs. 4A-1 through 4B-2 are substantially as described above in connection with Fig. 1, they will not be described further herein.

The foregoing description has been limited to a specific embodiment of this invention. It will be apparent, however, that variations and modifications may be made to the invention, with the attainment of some or all of the advantages of the invention. Therefore, it is the object of the appended claims to cover all such variations and modifications as come within the true spirit and scope of the invention.

Claims

1. A license management system for managing usage of a licensed software program comprising:
licensing storage means for storing a licensing unit value identifying a number of licensing units asso-

ciated with the licensed software program;
usage allocation value storage means for storing a usage allocation value identifying a number of licensing units associated with a use of the licensed software program; and
licensing verification means responsive to a usage request to use said licensed software program for determining, based on the contents of said licensing storage means and said usage allocation value storage means, whether usage of said licensed software program is permitted and, if usage is permitted, for adjusting the contents of said licensing storage means by a value to the contents of said usage allocation value storage means.

2. A license management system as defined in claim 1 for use in a digital data processing system which generates a system date value, said licensing storage means includes a plurality of fields including a licensing unit storage field for storing said licensing unit number identifying value and a field identifying a termination date, said licensing verification means further determining whether usage of said licensed software program is permitted in response to a comparison of said system date and said termination date.

3. A license management system as defined in claim 1 for managing usage of plurality of licensed software programs, wherein said licensing storage means includes a plurality of entries each containing a program identification field identifying a licensed software program and a licensing unit storage field for storing said licensing unit value, said licensing verification means including:
request receiving means for receiving a usage request identifying a licensed software program;
licensing unit retrieval means responsive to said request receiving means receipt of a usage request for retrieving the contents of said licensing unit storage field from the entry of said licensing storage means whose program identification field identifies the licensed software program identified in said usage request; and
licensing unit processing means for determining, based on the contents of retrieved licensing unit storage field and said usage allocation value storage means, whether usage of said licensed software program is permitted and, is usage is permitted, for adjusting the contents of said licensing storage means by a value related to the contents of said usage allocation value storage means.

4. A license management system as defined in claim 3 for use in a digital data processing system which generates a system date value, each entry in said licensing storage means further including a termination date field identifying a termination date, said licensing unit processing means further determining whether usage of said licensed software program is permitted in response to a comparison of said system date and said termination date.

5. A license management system as defined in claim 3 wherein said usage allocation value storage means includes a plurality of usage allocation tables each storing a value identifying a number of licensing units, each entry in said licensing storage means further including a usage allocation table identification field identifying a usage allocation table, said licensing verification means further including usage allocation table retrieval means responsive to said request receiving means receipt of a usage request for retrieving the contents of the usage allocation table identified by the contents of said usage allocation table identification field of said retrieved entry, said licensing unit processing means using said retrieved usage allocation table in its determination.

6. A license management system as defined in claim 5 wherein a request message further includes licensing usage allocation value selection criteria and each usage allocation table includes a plurality of entries each identifying a usage allocation value associated with a licensing usage allocation value selection criterion, said licensing verification means including means for retrieving, from the usage allocation table identified by said entry in said licensing storage means, the usage allocation value associated with the licensing usage allocation value selection criterion in said request message and using said retrieved usage allocation value in its determination.

7. A license management system as defined in claim 3 wherein a request message further includes licensing usage allocation value selection criteria and said usage allocation table includes a plurality of entries each identifying a usage allocation value associated with a licensing usage allocation selection criterion, said licensing verification means including means for retrieving the usage allocation value associated with the licensing usage allocation selection criterion in said request message and using said retrieved usage allocation value in its determination.

8. A license management system as defined in claim 1 wherein said licensing verification means further operates in response to a release request message for adjusting the contents of said licensing storage means by a value related to the contents of said usage allocation value storage means.

9. A license management system as defined in claim 8 for managing usage of a plurality of licensed software programs, wherein said licensing storage means includes a plurality of entries each containing a program identification field identifying a licensed software program and a licensing unit storage field for storing said licensing unit value,

said licensing verification means including:
request receiving means for receiving a release request identifying a licensed software program;
licensing unit processing means for adjusting the contents of said licensing storage means by a value related to the contents of said usage allocation value storage means.

10. A license management system as defined in claim 9 wherein said usage allocation value storage means includes a plurality of usage allocation tables each storing a value identifying a number of licensing units, each entry in said licensing storage means further including a usage allocation table identification field identifying a usage allocation table, said licensing verification means further including usage allocation table retrieval means responsive to said request receiving means receipt of a usage request for retrieving the contents of said usage allocation table identification field of said retrieved entry, said licensing unit processing means using retrieved usage allocation table in its adjusting.

11. A license management system as defined in claim 10 wherein a release message further includes licensing usage allocation value selection criteria and each usage allocation table includes a plurality of entries each identifying a usage allocation value associated with a licensing usage allocation value selection criterion, said licensing verification means including means for retrieving, from the usage allocation table identified by said entry in said licensing storage means, the usage allocation value associated with the licensing usage allocation value selection criterion in said request message and using said retrieved usage allocation value in its adjusting.

12. A license management system as defined in claim 8 wherein a release message further includes licensing usage allocation value selection criteria and each usage allocation table includes a plurality of entries each identifying a usage allocation value associated with a licensing usage allocation value selection criterion, said licensing verification means including means for retrieving, from the usage allocation value table identified by said entry in said licensing storage means, the usage allocation value associated with the licensing usage allocation value selection criterion in said request message and using said retrieved usage allocation value in its adjusting.

13. A license management system for use in a digital data processing system including a system date generating means for generating a system date value, said license management system comprising:
licensing storage means including a plurality of entries each associated with a licensed software program, each entry containing a licensing units field for storing a licensing unit value identifying a number of licensing units associated with the license software program, a usage allocation table, and a termination date;
usage allocation table storage means for storing a plurality of usage allocation tables, each usage allocation table having a plurality of usage allocation entries each usage allocation entry being associated with a licensing usage allocation value selection criterion and storing a usage allocation value identifying a number of licensing units; and
licensing verification means including:
usage grant means including:
usage request message receiving means for receiving a usage request message from a licensed software program, said usage request message identifying said licensed software program and usage grant criteria;
entry retrieval means responsive to the receipt of a usage request message for retrieving from said licensing storage means the licensing table entry associated with said licensed software program;
usage allocation table retrieval means for retrieving from said usage allocation table storage means a usage allocation entry identified by said retrieved licensing table entry and the licensing usage allocation value selection criterion identified by the received usage request message;
licensing request processing means including:
usage determination means including licensing unit comparing means for comparing the contents of said licensing units field and said usage allocation units field and date comparison means for comparing the system date value with the contents of said termination date field to determine whether usage of said licensed software program is permitted.
response generation means for generating a message in response to the determination by said usage determination means; and
licensing unit adjusting means for adjusting the contents of said licensing units field in response to a positive determination by said usage determination means;
usage release means including:
usage release message receiving means for receiving a usage request message from a licensed software program; said usage request message identifying said licensed software program and usage grant criteria;
entry retrieval means responsive to the receipt of a usage request message for retrieving from said licensing storage means the licensing table entry associated with said licensed software program;
usage allocation table retrieval means for retrieving from said usage allocation table storage means a usage allocation entry identified by said retrieved licensing table entry and the licensing usage allocation value selection criterion identified by the

received usage request message;
licensing release processing means for adjusting the contents of said licensing units field in relation to the value of said usage allocation entry.

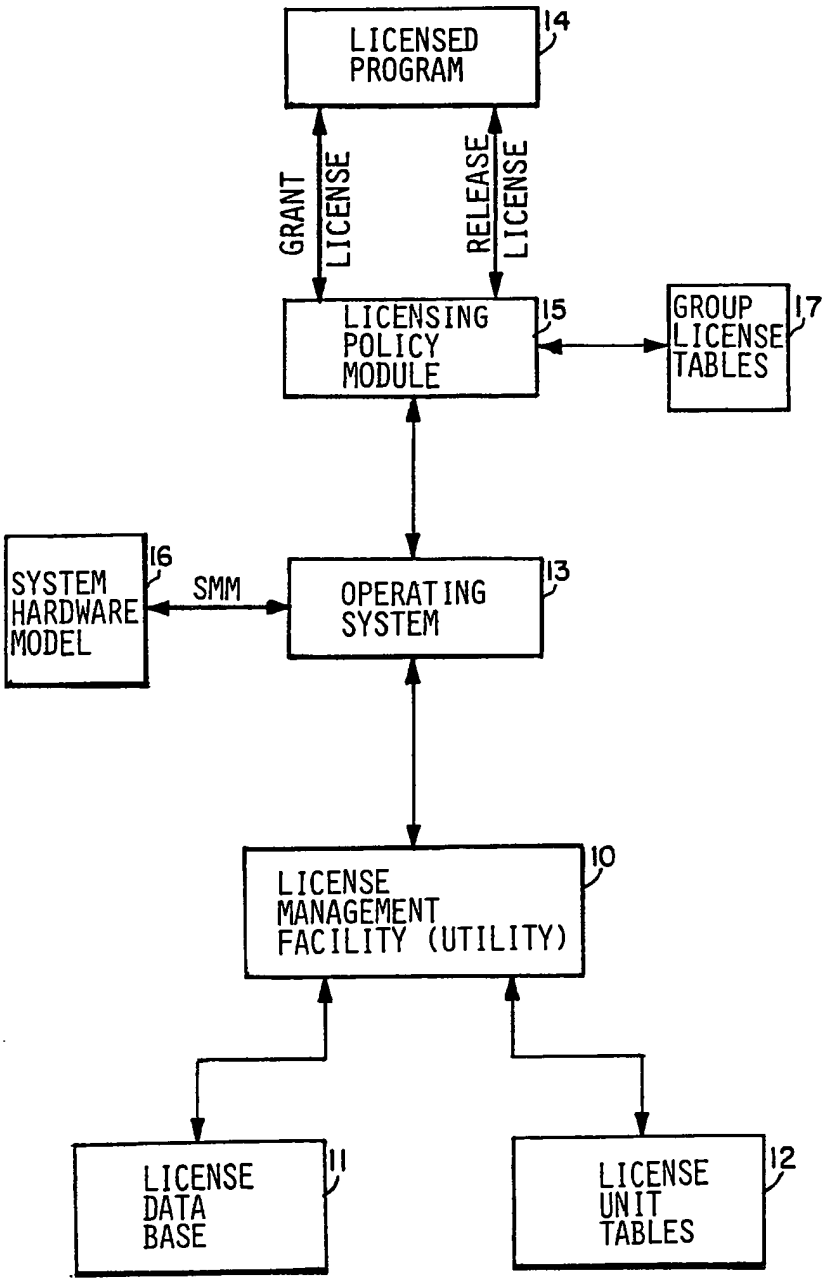


FIG. 1

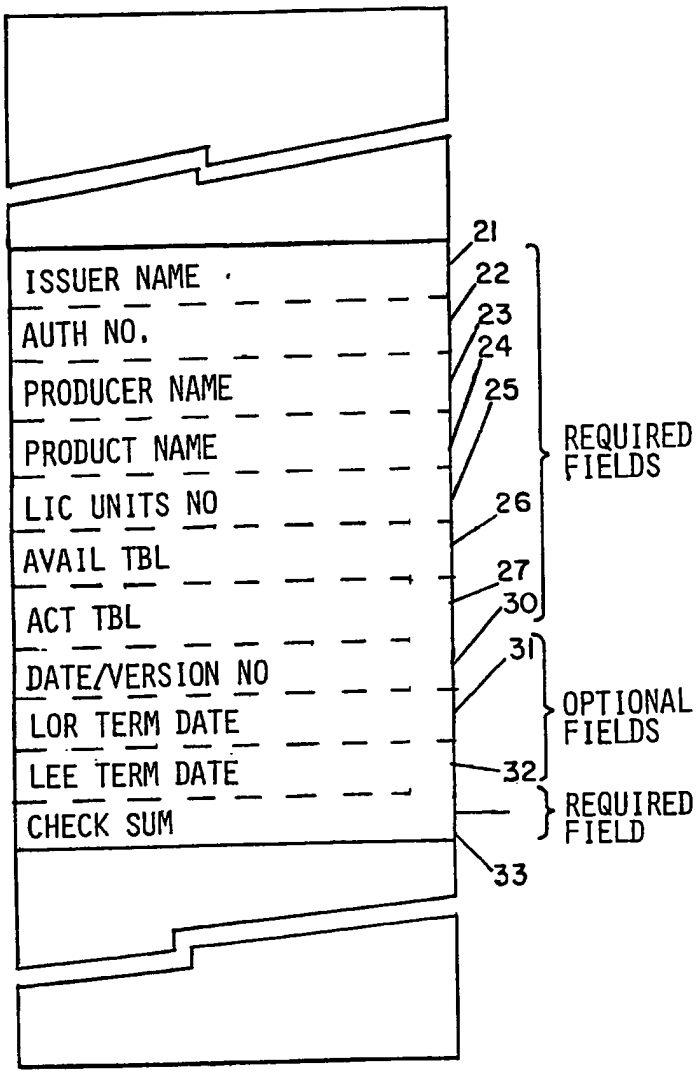
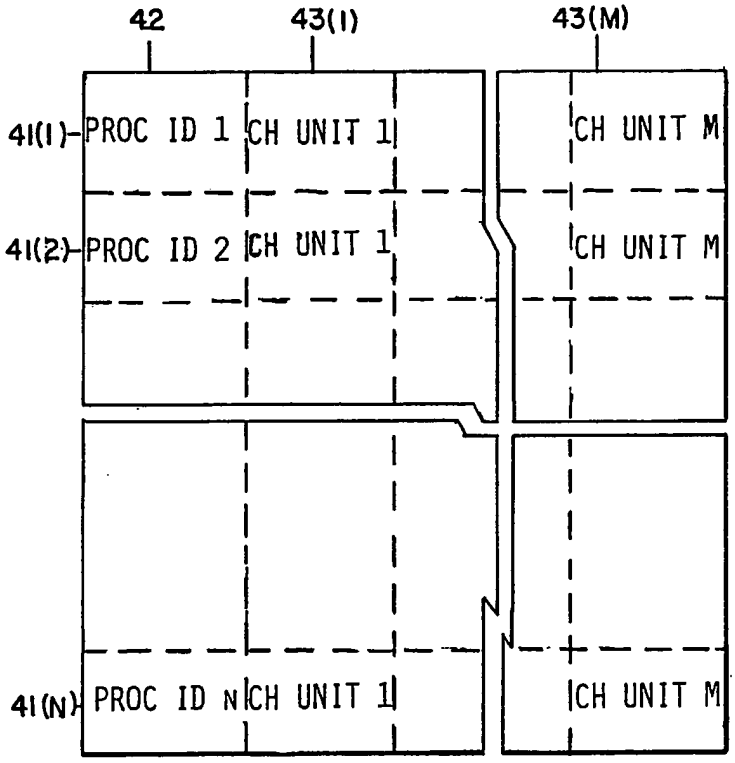


FIG. 2



LICENSE UNIT TABLE 40

FIG. 3

FIG. 4A-1

GRANT LICENSE

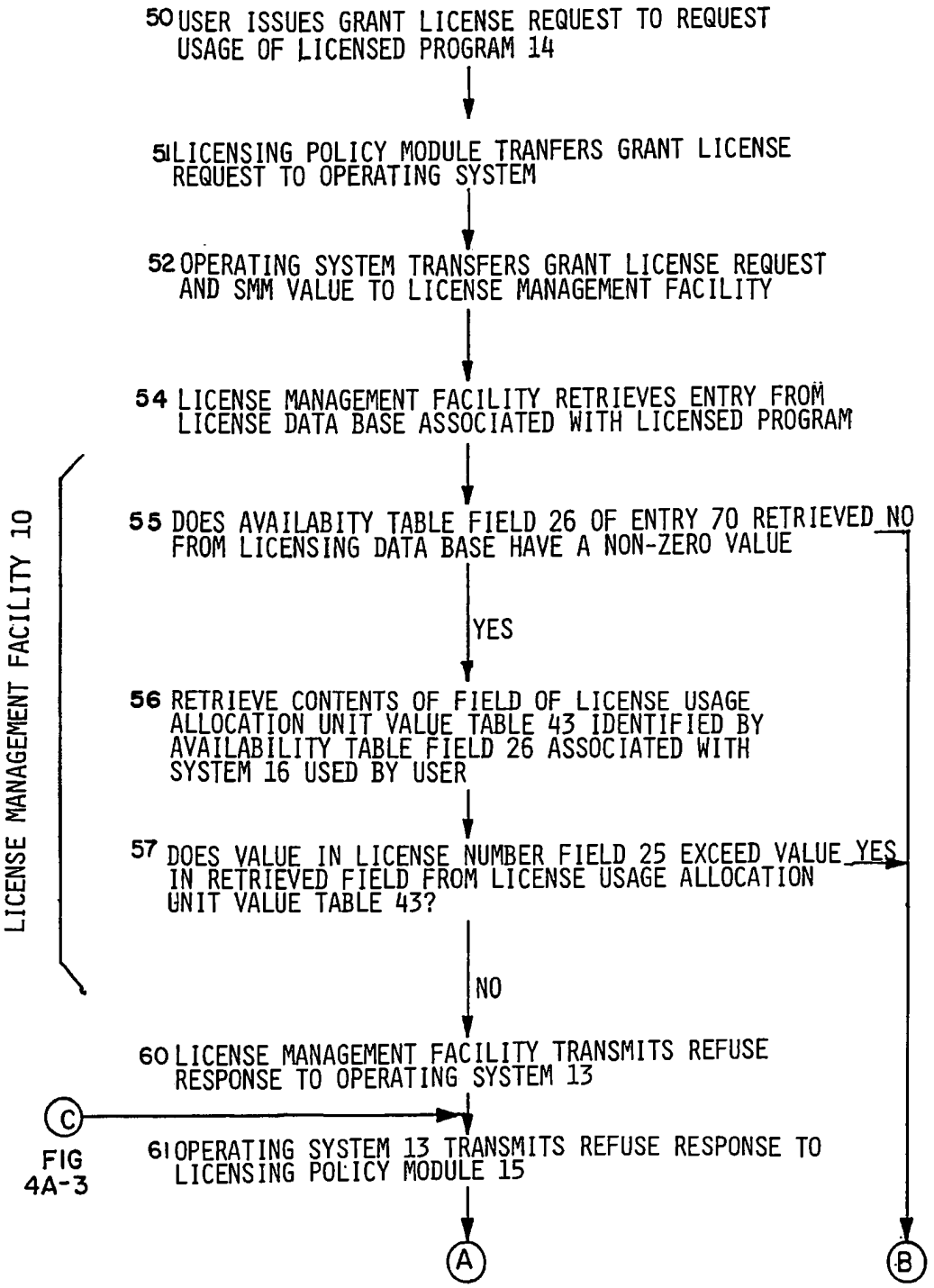


FIG. 4A-2

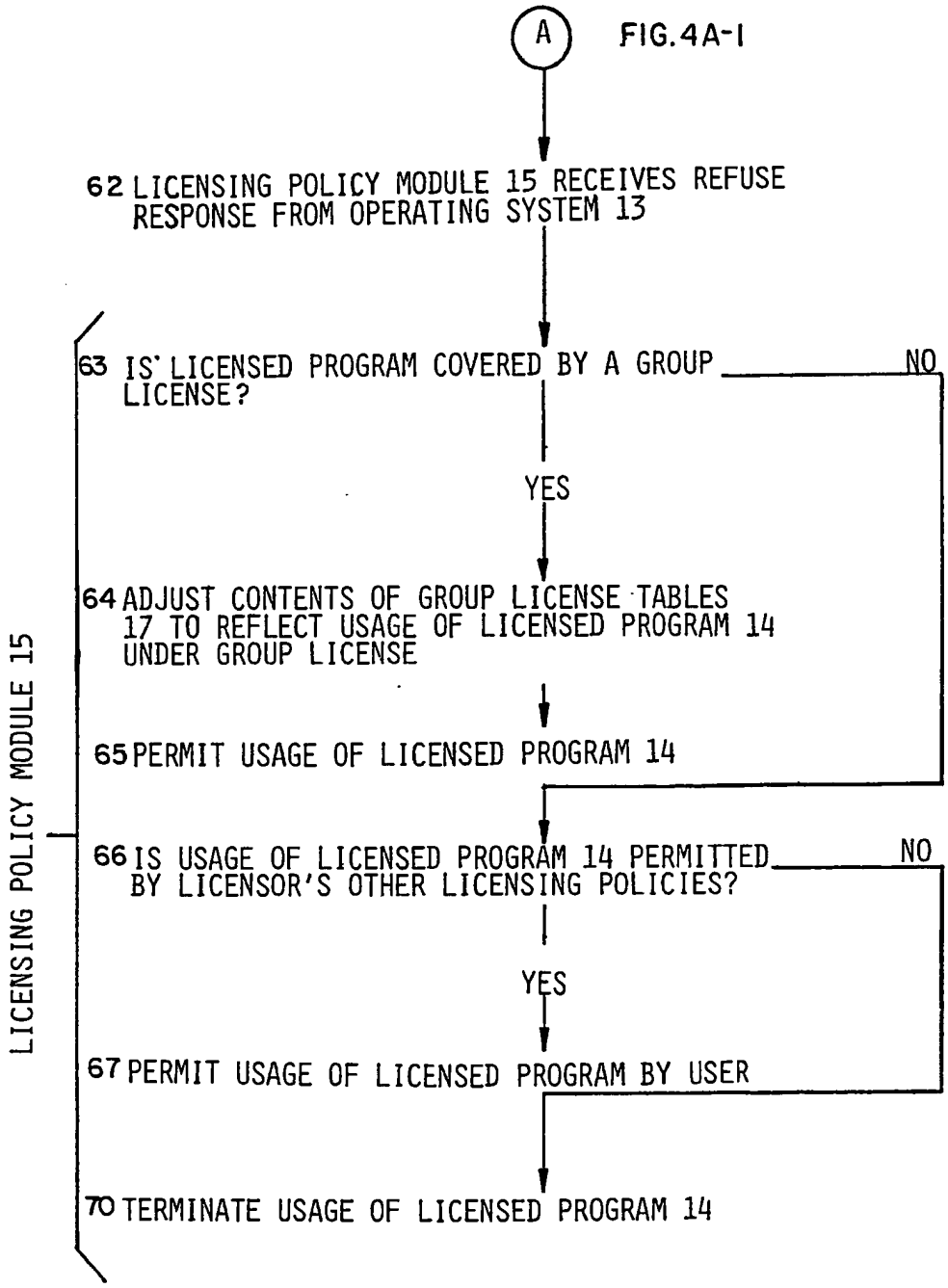


FIG. 4A-3

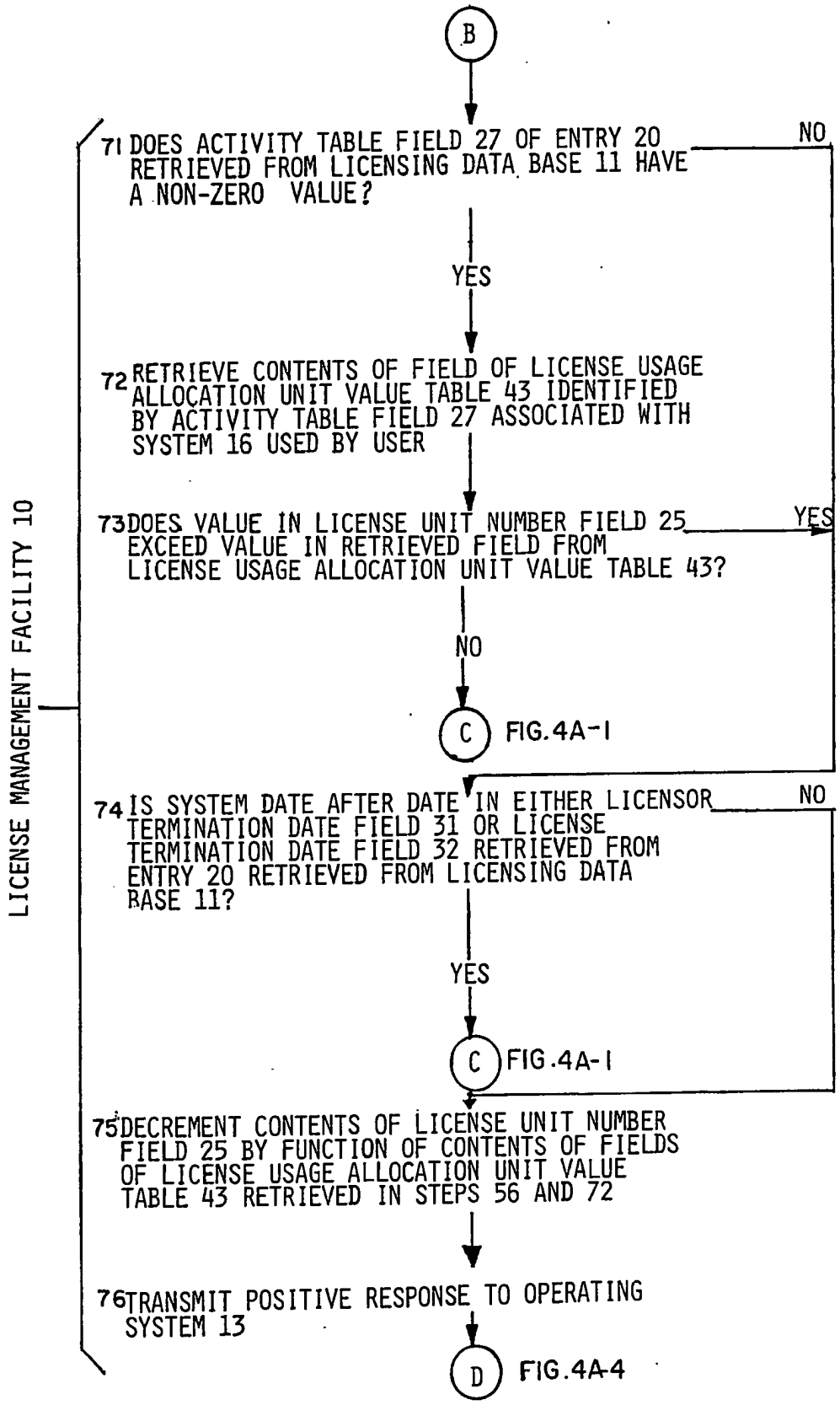

B
71 DOES ACTIVITY TABLE FIELD 27 OF ENTRY 20 RETRIEVED FROM LICENSING DATA BASE 11 HAVE A NON-ZERO VALUE?
NO
YES
72 RETRIEVE CONTENTS OF FIELD OF LICENSE USAGE ALLOCATION UNIT VALUE TABLE 43 IDENTIFIED BY ACTIVITY TABLE FIELD 27 ASSOCIATED WITH SYSTEM 16 USED BY USER
73 DOES VALUE IN LICENSE UNIT NUMBER FIELD 25 EXCEED VALUE IN RETRIEVED FIELD FROM LICENSE USAGE ALLOCATION UNIT VALUE TABLE 43?
YES
NO
C
FIG.4A-1
74 IS SYSTEM DATE AFTER DATE IN EITHER LICENSOR TERMINATION DATE FIELD 31 OR LICENSE TERMINATION DATE FIELD 32 RETRIEVED FROM ENTRY 20 RETRIEVED FROM LICENSING DATA BASE 11?
NO
YES
C
FIG.4A-1
75 DECREMENT CONTENTS OF LICENSE UNIT NUMBER FIELD 25 BY FUNCTION OF CONTENTS OF FIELDS OF LICENSE USAGE ALLOCATION UNIT VALUE TABLE 43 RETRIEVED IN STEPS 56 AND 72
76 TRANSMIT POSITIVE RESPONSE TO OPERATING SYSTEM 13
D
FIG.4A-4
LICENSE MANAGEMENT FACILITY 10

FIG. 4A-4

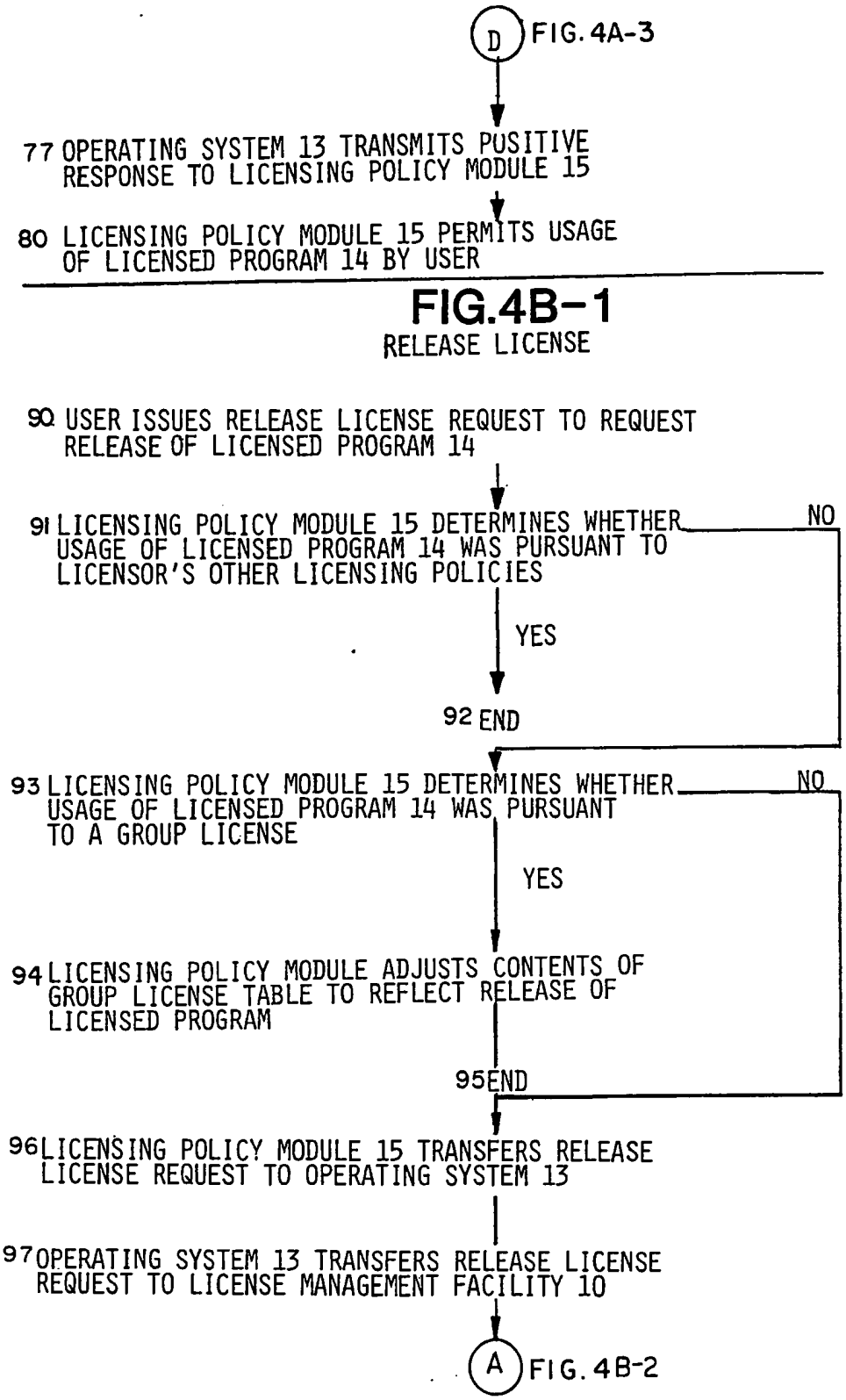
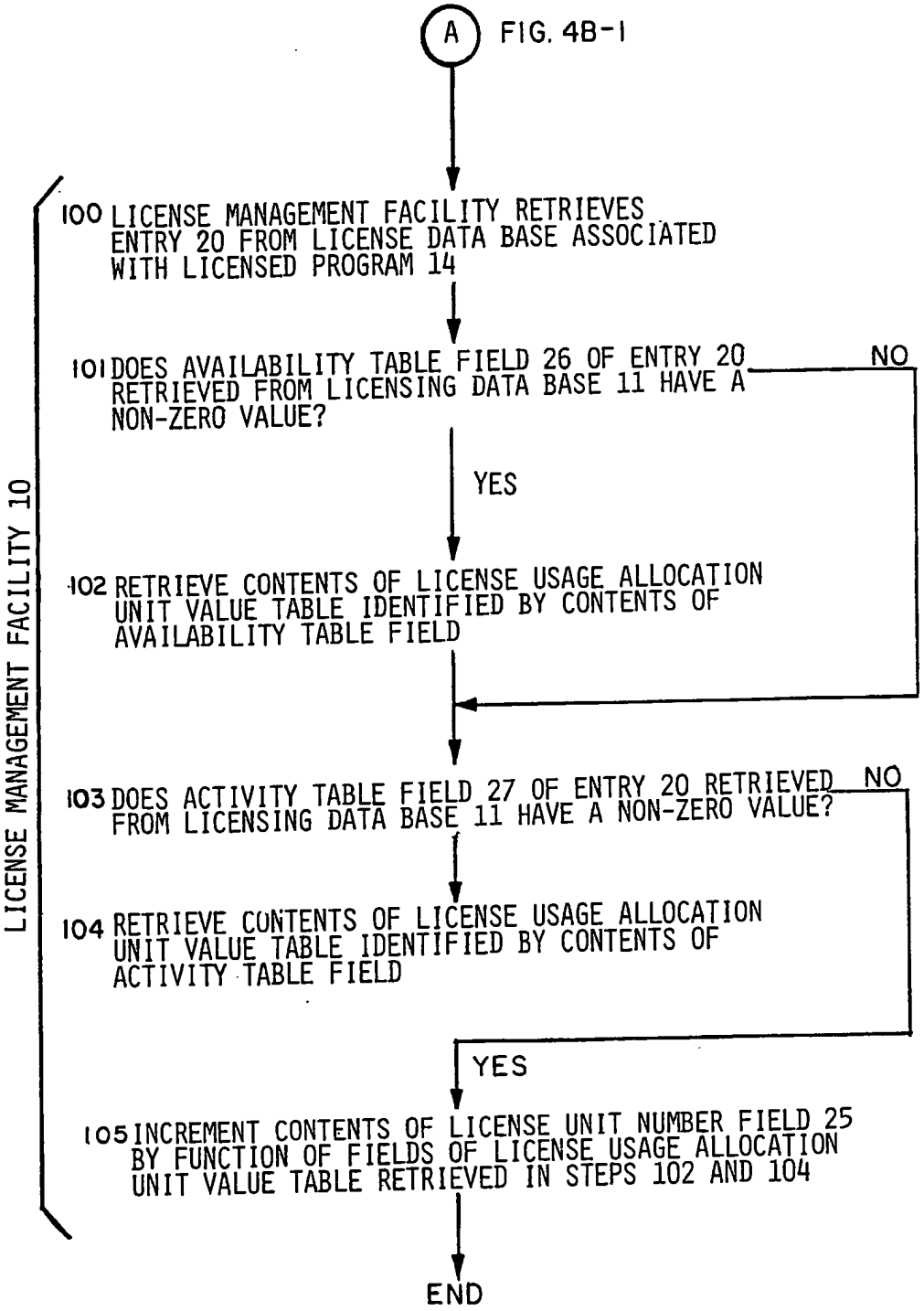


FIG.4B-1

RELEASE LICENSE

FIG. 4B-2


A
FIG. 4B-1
LICENSE MANAGEMENT FACILITY 10
100 LICENSE MANAGEMENT FACILITY RETRIEVES ENTRY 20 FROM LICENSE DATA BASE ASSOCIATED WITH LICENSED PROGRAM 14
101 DOES AVAILABILITY TABLE FIELD 26 OF ENTRY 20 RETRIEVED FROM LICENSING DATA BASE 11 HAVE A NON-ZERO VALUE?
NO
YES
102 RETRIEVE CONTENTS OF LICENSE USAGE ALLOCATION UNIT VALUE TABLE IDENTIFIED BY CONTENTS OF AVAILABILITY TABLE FIELD
103 DOES ACTIVITY TABLE FIELD 27 OF ENTRY 20 RETRIEVED FROM LICENSING DATA BASE 11 HAVE A NON-ZERO VALUE?
NO
104 RETRIEVE CONTENTS OF LICENSE USAGE ALLOCATION UNIT VALUE TABLE IDENTIFIED BY CONTENTS OF ACTIVITY TABLE FIELD
YES
105 INCREMENT CONTENTS OF LICENSE UNIT NUMBER FIELD 25 BY FUNCTION OF FIELDS OF LICENSE USAGE ALLOCATION UNIT VALUE TABLE RETRIEVED IN STEPS 102 AND 104
END

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 393 806 A2**

(12) EUROPEAN PATENT APPLICATION

(21) Application number: **90300115.4**

(51) Int. Cl.5: **H04L 9/32, H04L 9/08**

(22) Date of filing: **05.01.90**

(30) Priority: **17.04.89 US 339555**

(43) Date of publication of application:
24.10.90 Bulletin 90/43

(84) Designated Contracting States:
DE FR GB IT

(71) Applicant: **TRW INC.**
1900 Richmond Road
Cleveland Ohio 44124(US)

(72) Inventor: **Goss, Kenneth C.**
1470 Island Court
Oceano California 93445-9464(US)

(74) Representative: **Allden, Thomas Stanley et al**
A.A. THORNTON & CO. Northumberland
House 303-306 High Holborn
London WC1V 7LE(GB)

(54) Cryptographic method and apparatus for public key exchange with authentication.

(57) A technique for use in a public key exchange cryptographic system, in which two user devices establish a common session key by exchanging information over an insecure communication channel, and in which each user can authenticate the identity of the other, without the need for a key distribution center. Each device has a previously stored unique random number Xi, and a previously stored composite quantity that is formed by transforming Xi to Yi using a transformation of which the inverse in computationally infeasible; then concatenating Yi with a publicly known device identifier, and digitally signing the quantity. Before a communication session is established, two user devices exchange their signed composite quantities, transform them to unsigned form, and authenticate the identity of the other user. Then each device generates the same session key by transforming the received Y value with its own X value. For further security, each device also generates another random number X'i, which is transformed to a corresponding number Y'i. These Y'i values are also exchanged, and the session key is generated in each device, using a transformation that involves the device's own Xi and X'i numbers and the Yi and Y'i numbers received from the other device.

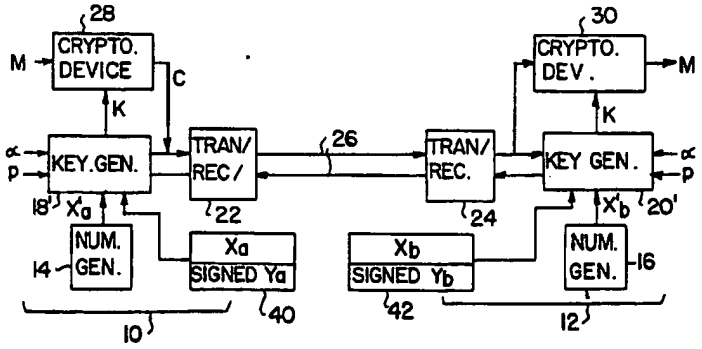


FIG. 3

EP 0 393 806 A2

BACKGROUND OF THE INVENTION

This invention relates generally to cryptographic systems and, more particularly, to cryptographic systems in which an exchange of information on an unsecured communications channel is used to establish a common cipher key for encryption and decryption of subsequently transmitted messages. Cryptographic systems are used in a variety of applications requiring the secure transmission of information from one point to another in a communications network. Secure transmission may be needed between computers, telephones, facsimile machines, or other devices. The principal goal of encryption is the same in each case: to render the communicated data secure from unauthorized eavesdropping.

By way of definition, "plaintext" is used to refer to a message before processing by a cryptographic system. "Ciphertext" is the form that the message takes during transmission over a communications channel. "Encryption" or "encipherment" is the process of transformation from plaintext to ciphertext. "Decryption" or "decipherment" is the process of transformation from ciphertext to plaintext. Both encryption and decryption are controlled by a "cipher key" or keys. Without knowledge of the encryption key, a message cannot be encrypted, even with knowledge of the encrypting process. Similarly, without knowledge of the decryption key, the message cannot be decrypted, even with knowledge of the decrypting process.

More specifically, a cryptographic system can be thought of as having an enciphering transformation E_k, which is defined by an enciphering algorithm E that is used in all enciphering operations, and a key K that distinguishes E_k from other operations using the algorithm E. The transformation E_k encrypts a plaintext message M into an encrypted message, or ciphertext C. Similarly, the decryption is performed by a transformation D_k defined by a decryption algorithm D and a key K.

Dorothy E.R. Denning, in "Cryptography and Data Security," Addison-Wesley Publishing Co. 1983, suggests that, for complete secrecy of the transmitted message, two requirements have to be met. The first is that it should be computationally infeasible for anyone to systematically determine the deciphering transformation D_k from intercepted ciphertext C, even if the corresponding plaintext M is known. The second is that it should be computationally infeasible to systematically determine plaintext M from intercepted ciphertext C. Another goal of cryptography systems is that of data authenticity. This requires that someone should not be able to substitute false ciphertext C' for ciphertext C without detection.

By way of further background, cryptographic systems may be classified as either "symmetric" or "asymmetric." In symmetric systems, the enciphering and deciphering keys are either the same easily determined from each other. When two parties wish to communicate through a symmetric cryptographic system, they must first agree on a key, and the key must be transferred from one party to the other by some secure means. This usually requires that keys be agreed upon in advance, perhaps to be changed on an agreed timetable, and transmitted by courier or some other secured method. Once the keys are known to the parties, the exchange of messages can proceed through the cryptographic system.

An asymmetric cryptosystem is one in which the enciphering and deciphering keys differ in such a way that at least one key is computationally infeasible to determine from the other. Thus, one of the transformations E_k or D_k can be revealed without endangering the other.

In 1976, the concept of a "public key" encryption system was introduced by W. Diffie and M. Hellman, "New Directions in Cryptography," IEEE Trans. on Info. Theory, Vol. IT-22(6), pp. 644-54 (Nov. 1976). In a public key system, each user has a public key and private key, and two users can communicate knowing only each other's public keys. This permits the establishment of a secured communication channel between two users without having to exchange "secret" keys before the communication can begin. As pointed out in the previously cited text by Denning, a public key system can be operated to provide secrecy by using a private key for decryption; authenticity by using a private key for encryption; or both, by using two sets of encryptions and decryptions.

In general, asymmetric cryptographic systems require more computational "energy" for encryption and decryption than symmetric systems. Therefore, a common development has been a hybrid system in which an asymmetric system, such as a public key system, is first used to establish a "session key" for use between two parties wishing to communicate. Then this common session key is used in a conventional symmetric cryptographic system to transmit messages from one user to the other. Diffie and Hellman have proposed such a public key system for the exchange of keys on an unsecured communications channel. However, as will be described, the Diffie-Hellman public key system is subject to active eavesdropping. That is to say, it provides no foolproof authentication of its messages. With knowledge of the public keys, an eavesdropper can decrypt received ciphertext, and then re-encrypt the resulting plaintext for transmission to the intended receiver, who has no way of knowing that

the message has been intercepted. The present invention relates to a significant improvement in techniques for public key exchange or public key management.

One possible solution to the authentication problem in public key management, is to establish a key distribution center, which issues secret keys to authorized users. The center provides the basis for identity authentication of transmitted messages. In one typical technique, a user wishing to transmit to another user sends his and the other user's identities to the center; e.g. (A,B). The center sends to A the ciphertext message $E_A(B,K,T,C)$, where E_A is the enciphering transformation derived from A's private key, K is the session key, T is the current date and time, and $C = E_B(A,K,T)$, where E_B is the enciphering transformation derived from B's private key. Then A sends to B the message C. Thus A can send to B the session key K encrypted with B's private key; yet A has no knowledge of B's private key. Moreover, B can verify that the message truly came from A, and both parties have the time code for further message identity authentication. The difficulty, of course, is that a central facility must be established as a repository of private keys, and it must be administered by some entity that is trusted by all users. This difficulty is almost impossible to overcome in some applications, and there is, therefore, a significant need for an alternative approach to public key management. The present invention fulfills this need.

Although the present invention has general application in many areas of communication employing public key management and exchange, the invention was first developed to satisfy a specific need in communication by facsimile (FAX) machines. As is now well known, FAX machines transmit and receive graphic images over ordinary telephone networks, by first reducing the images to digital codes, which are then transmitted, after appropriate modulation, over the telephone lines. FAX machines are being used at a rapidly increasing rate for the transmission of business information, much of which is of a confidential nature, over lines that are unsecured. There is a substantial risk of loss of the confidentiality of this imformation, either by deliberate eavesdropping, or by accidental transmission to an incorrectly dialed telephone number.

Ideally, what is needed is an encrypting/decrypting box connectable between the FAX machine and the telephone line, such that secured communications can take place between two similarly equipped users, with complete secrecy of data, and identity authentication between the users. For most users, a prior exchange of secret keys would be so inconvenient that they could just as well exchange the message itself by the same secret technique. A public key exchange system is by far the most convenient solution but each available variation of these systems has its own problems, as discussed above. The Diffie-Hellman approach lacks the means to properly authenticate a message, and although a key distribution center would solve this problem, as a practical matter no such center exists for FAX machine users, and none is likely to be established in the near future. Accordingly, one aspect of the present invention is a key management technique that is directly applicable to data transmission using FAX machines.

SUMMARY OF THE INVENTION

The present invention resides in a public key cryptographic system that accomplishes both secrecy and identity authentication, without the need for a key distribution center or other public facility, and without the need for double encryption and double decryption of messages. Basically, the invention achieves these goals by using a digitally signed composite quantity that is pre-stored in each user communication device. In contrast with the conventional Diffie-Hellman technique, in which random numbers Xi are selected for each communication session, the present invention requires that a unique number Xi be preselected and pre-stored in each device that is manufactured. Also stored in the device is the signed composite of a Yi value and a publicly known device identifier. The Yi value is obtained by a transformation from the Xi value, using a transformation that is practically irreversible.

Before secure communications are established, two devices exchange these digitally signed quantities, which may then be easily transformed into unsigned form. The resulting identifier information is used to authenticate the other user's identity, and the resulting Yi value from the other device is used in a transformation with Xi to establish a session key. Thus the session key is established without fear of passive or active eavesdropping, and each user is assured of the other's identity before proceeding with the transfer of a message encrypted with the session key that has been established.

One way of defining the invention is in terms of a session key generator, comprising storage means for storing a number of a first type selected prior to placing the key generator in service, and a digitally signed composite quantity containing both a unique and publicly known identifier of the session key generator and a number of a second type obtained by a practically irreversible transformation of the

number of the first type. The session key generator has a first input connected to receive the number of the first type, and a second input connected to receive an input quantity transmitted over an insecure communications channel from another session key generator, the input quantity being digitally signed and containing both a publicly known identifier of the other session key generator and a number of the second type generated by a practically irreversible transformation of a number of the first type stored in the other session key generator. The session key generator also has a first output for transmitting the stored, digitally signed composite quantity over the insecure communications channel to the other session key generator, a second output, means for decoding the signed input quantity received at the second input, to obtain the identifier of the other session key generator and the received number of the second type, and means for generating a session key at the second output, by performing a practically irreversible transformation of the number of the second type received through the second input, using the number of the first type received through the first input.

For further security of the session key, the session key generator further includes a third input, connected to receive another number of the first type, generated randomly, and means for generating at the first output, for transmission with the digitally signed composite quantity, a number of the second type obtained by a practically irreversible transformation of the number of the first type received through the third input. The session key generator also includes means for receiving from the second input another number of the second type generated in and transmitted from the other session key generator. The means for generating a session key performs a practically irreversible transformation involving both numbers of the first type, received at the first and third inputs, and both numbers of the second type received at the second input, whereby a different session key may be generated for each message transmission session.

More specifically, the number of the second type stored in digitally signed form in the storage means is obtained by the transformation $Ya = \alpha^{Xa} \bmod p$, where Xa is the number of the first type stored in the storage means, and α and p are publicly known transformation parameters. The number of the second type received in the digitally signed composite quantity from the other session key generator is designated Yb, and the means for generating the session key performs the transformation $K = Yb^{Xa} \bmod p$.

When additional numbers $X'a$ and $X'b$ are also generated prior to transmission, the means for generating the session key performs the transformation $K = (Y'b)^{Xa} \bmod p \oplus (Yb)^{X'a} \bmod p$, where $X'a$ is the number of the first type that is randomly generated, $Y'b$ is the additional number of the second type received from the other session key generator, and the $\oplus$ symbol means an exclusive OR operation.

In terms of a novel method, the invention comprises the steps of transmitting from each device a digitally signed composite quantity to the other device, the composite quantity including a publicly known device identifier IDa and a number Ya derived by a practically irreversible transformation of a secret number Xa that it is unique to the device, receiving a similarly structured digitally signed composite quantity from the other device, and transforming the received digitally signed composite quantity into an unsigned composite quantity containing a device identifier IDb of the other device and a number Yb that was derived by transformation from a secret number Xb that is unique to the other device. Then the method performs the steps of verifying the identity of the other device from the device identifier IDb, and generating a session key by performing a practically irreversible transformation involving the numbers Xa and Yb.

Ideally, the method also includes the steps of generating another number $X'a$ randomly prior to generation of a session key, transforming the number $X'a$ to a number $Y'a$ using a practically irreversible transformation, transmitting the number $Y'a$ to the other device, and receiving a number $Y'b$ from the other device. In this case, the step of generating a session key includes a practically irreversible transformation involving the numbers Xa, $X'a$, $Y'b$ and $Y'b$.

In particular, the transformations from X numbers to Y numbers is of the type $Y = \alpha^X \bmod p$, where α and p are chosen to maximize irreversibility of the transformations, and the step of generating a session key includes the transformation $K = (Y'b)^{Xa} \bmod p \oplus (Yb)^{X'a} \bmod p$, where $\oplus$ denotes an exclusive OR operation.

It will be appreciated from this brief summary that the present invention represents a significant advance in the field of cryptography. In particular, the invention provides for both secrecy and identity authenticity when exchanging transmissions with another user to establish a common session key. Other aspects and advantages of the invention will become apparent from the following more detailed description, taken in conjunction with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIGURE 1 is a block diagram showing a public key cryptographic system of the prior art;

FIG. 2 is a block diagram similar to FIG. 1, and showing how active eavesdropping may be used to attack the system;

FIG. 3 is a block diagram of a public key cryptographic system in accordance with the present invention;

FIG. 4 is a block diagram of a secure facsimile system embodying the present invention; and

FIG. 5 is a block diagram showing more detail of the cryptographic processor of FIG. 4.

DESCRIPTION OF THE PREFERRED EMBODIMENT

As shown in the accompanying drawings for purposes of illustration, the present invention is concerned with a public key cryptographic system. As discussed at length in the preceding background section of this specification, public key systems have, prior to this invention, been unable to provide both secrecy and identity authentication of a message without either a costly double transformation at each end of the communications channel, or the use of key distribution center.

U.S. Patent No. 4,200,770 to Hellman et al. discloses a cryptographic apparatus and method in which two parties can converse by first both generating the same session key as a result of an exchange of messages over an insecure channel. Since the technique disclosed in the Hellman et al. '770 patent attempts to provide both secrecy and authentication in a public key cryptographic system, the principles of their technique will be summarized here. This should provide a better basis for an understanding of the present invention.

In accordance with the Hellman et al. technique, two numbers α and p are selected for use by all users of the system, and may be made public. For increased security, p is a large prime number, and α has a predefined mathematical relationship to p, but these restrictions are not important for purposes of this explanation. Before starting communication, two users, A and B, indicated in FIG. 1 at 10 and 12, perform an exchange of messages that results in their both computing the same cipher key, or session key K, to be used in transmitting data back and forth between them. The first step in establishing the session key is that each user generates a secret number in a random number generator 14, 16. The numbers are designated Xa, Xb, respectively, and are selected from a set of positive integers up to p-1. Each user also has a session key generator 18, 20, one function of which is to generate other numbers Y from the numbers X, α and p, using the transformations:

$Ya = \alpha^{Xa} \bmod p$,

$Yb = \alpha^{Xb} \bmod p$.

The values Ya, Yb are then processed through a conventional transmitter/receiver 22, 24, and exchanged over an insecure communications channel 26.

The term "mod p' means modulo p, or using modulo p arithmetic. Transforming an expression to modulo p can be made by dividing the expression by p and retaining only the remainder. For example, 34 mod 17 = 0, 35 mod 17 = 1, and so forth. Similarly, the expression for Ya may be computed by first computing the exponential expression α^{Xb}, then dividing the result by p and retaining only the remainder.

If α and p are appropriately chosen, it is computationally infeasible to compute Xa from Ya. That is to say, the cost of performing such a task, in terms of memory or computing time needed, is large enough to deter eavesdroppers. In any event, new X and Y values can be chosen for each message, which is short enough to preclude the possibility of any X value being computed from a corresponding Y value.

After the exchange of the values Ya, Yb, each user computes a session key K in its session key generator 18, 20, by raising the other user's Y value to the power represented by the user's own X value, all modulo p. For user A, the computation is:

$K = Yb^{Xa} \bmod p$.

Substituting for Yb,

$K = (\alpha^{Xb})^{Xa} \bmod p = \alpha^{XaXb} \bmod p$.

For user B, the computation is:

$K = Ya^{Xb} \bmod p$.

Substituting for Ya,

$K = (\alpha^{Xa})^{Xb} \bmod p = \alpha^{XaXb} \bmod p$.

The two users A, B now have the same session key K, which is input to a conventional cryptographic device 28, 30. A transmitting cryptographic device, e.g. 28, transforms a plaintext message M into ciphertext C for transmission on the communications channel 26, and a receiving cryptographic device 30 makes the inverse transformation back to the plaintext M.

The Hellman et al. 770 patent points out that the generation of a session key is secure from eavesdropping, because the information exchanged on the insecure channel includes only the Y values, from which the corresponding X values cannot be easily computed. However, this form of key exchange system still has two significant problems. One is that the system is vulnerable to attack from active eavesdropping, rather than the passive eavesdropping described in the patent. The other is that identity authentication can be provided only by means of a public key directory.

Active eavesdropping takes place when an unauthorized person places a substitute message on

the communications channel. FIG. 2 depicts an example of active eavesdropping using the same components as FIG. 1. The active eavesdropper E has broken the continuity of the unsecured line 26, and is receiving messages from A and relaying them to B, while sending appropriate responses to A as well. In effect, E is pretending to be B, with device Eb, and is also pretending to be A, with device Ea. E has two cryptographic devices 34a, 34b, two session key generators 36a, 36b, and two number generators 38a, 38b. When device Eb receives Ya from A, it generates Xb′ from number generator 38b, computes Yb′ from Xb′ and transmits Yb′ to A. Device Eb and user A compute the same session key and can begin communication of data. Similarly, device Ea and user B exchange Y numbers and both generate a session key, ifferent from the one used by A and Eb. Eavesdropper E is able to decrypt the ciphertext C into plaintext M, then encipher again for transmission to B. A and B are unaware that they are not communicating directly with each other.

In accordance with the present invention, each user is provided with proof of identity of the party with whom he is conversing, and both active and passive eavesdropping are rendered practically impossible. FIG. 3 shows the key management approach of the present invention, using the same reference numerals as FIGS. 1 and 2, except that the session key generators are referred to in FIG. 3 as 18′ and 20′, to indicate that the key generation function is different in the present invention. The user devices also include a number storage area 40, 42. Storage area 40 contains a preselected number Xa, stored at the time of manufacture of the A device, and another number referred to as "signed Ya," also stored at the time of manufacture. Xa was chosen at random, and is unique to the device. Ya was computed from Xa using the transformation

$Ya = \alpha^{Xa} \bmod p.$

Then the Ya value was concatenated with a number IDa uniquely identifying the user A device, such as a manufacturer's serial number, and then encoded in such a way that it was digitally "signed" by the manufacturer for purposes of authenticity. The techniques for digitally signing data are known in the cryptography art, and some will be discussed below. For the present, one need only consider that the number designated "signed (Ya,IDa)" contains the value Ya and another value IDa uniquely identifying the A device, all coded as a "signature" confirming that the number originated from the manufacturer and from no-one else. User B's device 12 has stored in its storage area 42 the values Xb and signed (Yb,IDb).

Users A and B exchange the signed (Ya,IDa) and signed (Yb,IDb) values, and each session key generator 18, 20 then "unsigns" the received values and verifies that it is conversing with the correct user device. The user identifiers IDa and IDb are known publicly, so user device A verifies that the number IDb is contained in the signed (Yb,IDb) number that was received. Likewise, user device B verifies that the value signed (Ya,IDa) contains the known value IDa. By performing the process of "unsigning" the received messages, the user devices also confirm that the signed data originated from the manufacturer and not from some other entity.

Since the Xa, Xb values are secret values, and it is infeasible to obtain them from the transmitted signed (Ya,IDa) and signed (Yb,IDb) values, the users may both compute identical session keys in a manner similar to that disclosed in the Hellman et al. '770 patent. If an eavesdropper E were to attempt to substitute fake messages for the exchanged ones, he would be unable to satisfy the authentication requirements. E could intercept a signed (Ya,IDa) transmission, could unsign the message and obtain the values Ya and IDa. E could similarly obtain the values Yb and IDb. However, in order for E and A to use the same session key, E would have to generate a value Xe, compute Ye and concatenate it with IDb, which is known, and then digitally "sign" the composite number in the same manner as the manufacturer. As will be explained, digital signing involves a transformation that is very easy to effect in one direction, the unsigning direction, but is computationally infeasible in the other, the signing direction. Therefore, eavesdropper E would be unable to establish a common session key with either A or B because he would be unable to generate messages that would satisfy the authentication requirements.

As described thus far, the technique of the invention establishes a session key that is derived from X and Y values stored in the devices at the time of manufacture. Ideally, a new session key should be established for each exchange of message traffic. An additional unsecured exchange is needed to accomplish this.

The number generator 14 in the A device 10 generates a random number X′a and the number generator 16 in the B device 12 generates a random number X′b. These are supplied to the session key generators 18, 20, respectively, which generate values Y′a and Y′b in accordance with the transformations:

$Y'a = \alpha^{X'a} \bmod p,$

$Y'b = \alpha^{X'b} \bmod p.$

These values are also exchanged between the A and B devices, at the same time that the values of signed (Ya,IDa) and signed (Yb,IDb) are exchanged. After the authenticity of the message has been confirmed, as described above, the session

key generators perform the following transformations to derive a session key. At the A device, the session key is computed as

$Ka = (Y'b)^{Xa} \bmod p \oplus (Yb)^{X'a} \bmod p,$

and at the B device, the session key is computed as

$Kb = (Y'a)^{Xb} \bmod p \oplus (Ya)^{X'b} \bmod p,$

where "⊕" means an exclusive OR operation.

Thus the session key is computed at each device using one fixed number, i.e. fixed at manufacturing time, and one variable number, i.e. chosen at session time. The numbers are exclusive ORed together on a bit-by-bit basis. It can be shown that Ka = Kb by substituting for the Y values. Thus:

$Ka = (\alpha^{X'b})^{Xa} \bmod p \oplus (\alpha^{Xb})^{X'a} \bmod p$
$= (\alpha^{Xa})^{X'b} \bmod p \oplus (\alpha^{X'a})^{Xb} \bmod p$
$= (Ya)^{X'b} \bmod p \oplus (Y'a)^{Xb} \bmod p$
$= (Y'a)^{Xb} \bmod p \oplus (Ya)^{X'b} \bmod p$
$= Kb.$

This common session key satisfies secrecy and authentication requirements, and does not require double encryption-decryption or the use of a public key directory or key distribution center. The only requirement is that of a manufacturer who will undertake to supply devices that have unique device ID's and selected X values encoded into them. For a large corporation or other organization, this obligation could be assumed by the organization itself rather than the manufacturer. For example, a corporation might purchase a large number of communications devices and complete the manufacturing process by installing unique ID's, X values, and signed Y values in the units before distributing them to the users. This would relieve the manufacturer from the obligation.

The process described above uses parameters that must meet certain numerical restrictions. The length restrictions are to ensure sufficient security, and the other requirements are to ensure that each transformation using modulo arithmetic produces a unique transformed counterpart. First, the modulus p must be a strong prime number 512 bits long. A strong prime number is a prime number p that meets the additional requirement that (p-1)/2 has at least one large prime factor or is preferably itself a prime number. The base number must be a 512-bit random number that satisfies the relationships:

$\alpha^{(p-1)/2} \bmod p = p-1,$ and

$1 < \alpha < p-1.$

Finally, the values X and X' are chosen as 512-bit random numbers such that

$1 < X, X' < p-1.$

As indicated above, the process of authentication in the invention depends on the ability of the manufacturer, or the owner of multiple devices, to supply a signed Y value with each device that is distributed. A digital signature is a property of a message that is private to its originator. Basically, the signing process is effected by a transformation that is extremely difficult to perform, but the inverse transformation, the "unsigning," can be performed easily by every user. The present invention is not limited to the use of a particular digital signature technique.

One approach is to use an RSA public key signature technique. The RSA technique takes its name from the initial letters of its originators, Rivest, Shamir and Adleman, and is one of a class of encryption schemes known as exponentiation ciphers. An exponentiation cipher makes the transformation $C = P^e \bmod n$, where e and n constitute the enciphering key. The inverse transformation is accomplished by $P = C^d \bmod n$. With appropriate selection of n, d and e, the values of n and d can be made public without giving away the exponent e used in the encryption transformation. Therefore, a digital signature can be applied to data by performing the exponentiation transformation with a secret exponent e, and providing a public decryption exponent d, which, of course, will be effective to decrpyt only properly "signed" messages.

In the preferred embodiment of the present invention, another approach is used for digital signature, namely a modular square-root transformation. In the expression $x = m^2 \bmod n$, the number m is said to be the square root of x mod n, or the modular square root of x. If n is appropriately selected, the transformation is very difficult to perform in one direction. That is to say, it is very difficult to compute m from x, although easy to compute x from m. If the modulus n is selected to be the product of two large prime numbers, the inverse or square-root transformation can only be made if the factors of the modulus are known. Therefore, the modulus n is chosen as the product of two prime numbers, and the product is 1,024 bits long. Further, the factors must be different in length by a few bits. In the devices using the present invention, the value "signed (Ya,IDa)" is computed by first assembling or concatenating the codes to be signed. These are:

1. A numerical code IDa uniquely identifying the A device. In the present embodiment of the invention, this is a ten-digit (decimal) number encoded in ASCII format, but it could be in any desired format.

2. A number of ASCII numerical codes indicating a version number of the device. This may be used for device testing or analyzing problems relating to device incompatability.

3. The value Ya computed from the chosen value of Xa, encoded in binary form.

4. A random value added to the least-significant end of the composite message, and used to ensure that the composite message is a perfect

modular square.

The last element of the message is needed because of inherent properties of the modular squaring process. If one were to list all possible values of a modular square x, from 1 to n-1, and all corresponding values of the modular square root m, some of the values of x would have multiple possible values of m, but others of the values of x would have no corresponding values of m. The value added to the end of the message ensures that the number for which a modular square root is to be computed, is one that actually has a modular square root. A simple example should help make this clear.

Suppose the modulus n is 7849. It can be verified by calculator that a value x of 98 has four possible values of m in the range 1 to n-1: 7424, 1412, 6437 and 425, such that m^2 mod 7849 = 98. However, the x value 99 has no possible modular square root values m. If the composite message to be signed had a numerical value of 99, it would be necessary to add to it a value such as 1, making a new x value of 100, which has four possible square root values in the range 1 to n-1, namely 1326, 7839, 10 and 6523. In most instances, it does not matter which of these is picked by the modular square root process employed, since the squaring or "unsigning" process will always yield the composite message value 100 again. However, there are a few values of m that should be avoided for maximum security. If the x value is a perfect square in ordinary arithmetic (such as the number 100 in the example), two values of m that should be avoided are the square root of x by ordinary arithmetic (the number 10 in the example), and the number that is the difference between the modulus n and the ordinary-arithmetic square root of x (i.e. 7839 in the example). If a number fitting this definition is used as a signed message, the signature is subject to being "forged" without knowledge of the factors of n. Therefore, such numbers are avoided in assigning signatures, and each device can be easily designed to abort an exchange when the signed message takes the form of one of these avoided numbers.

When the modular square root process is used for digitally signing the composite data stored in each device, the "unsigning" process upon receipt of a signed composite message is simply the squaring of the message, modulo n. The value n is not made public, although it could be determined by close examination of one of the devices. Even with knowledge of the modulus n, however, the computation of the modular square root is computationally infeasible without knowledge of the factorization of n.

With a knowledge of the factorization of the modulus n, the computation of the modular square root becomes a feasible, although laborious task, which may be performed by any known computational method. It will be recalled that this process is performed prior to distribution of the devices embodying the invention, so computation time is not a critical factor.

It will be understood that the cryptographic technique of the invention may be implemented in any form that is convenient for a particular application. Modular arithmetic is now well understood by those working in the field, and may be implemented in hardware form in the manner described in the '770 Hellman et al. patent. More conveniently, off-the-shelf modular arithmetic devices are available for connection to conventional microprocessor hardware. For example, part number CY1024 manufactured by CYLINK, of Sunnyvale, California 94087, performs modular addition, multiplication and exponentiation.

For application to facsimile communications, the technique of the invention may be made completely "transparent" to the user. FIG. 4 shows the architecture of a device for connection between a conventional FAX machine 50 and a telephone line 52. The device includes a first conventional modem 54 (modulator/demodulator) for connection to the FAX machine 50 and a second modem 56 for connection to the telephone line 52. The modems 54, 56 function to demodulate all messages entering the device from either the FAX machine or the telephone line, and to modulate messages for transmission to the FAX machine or onto the telephone line. The device further includes a communications processor 58 connected between the two modems 54, 56, and a cryptographic processor 60 connected to the communications processor 58. The communications processor 58 manages message traffic flow to and from the modems 54, 56 and to and from the cryptographic processor 60, and ensures that the necessary communications protocols are complied with. In one preferred embodiment of the invention, the communications processor is a microprocessor specified by part number MC68000, manufactured by Motorola Corporation.

As shown in FIG. 5, the cryptographic processor 60 includes a conventional microprocessor 62 having a data bus 64 and a data bus 66, to which various other modules are connected. The microprocessor 62 may be, for example, a National Semiconductor Company device specified by part number NSC800. The connected modules include a random access memory (RAM) 68, a read-only memory (ROM) 70, which serves as a storage area for the X value and the signed Y value, an integrated-circuit chip 72 for implementation of the Data Encryption Standard (DES), a modular arithmetic device 74 such as the CYLINK CY1024,

and an interface module 76 in the form of a dual-port RAM, for connection to the communications processor 58.

For transparent operation of the device shown in FIGS. 4 and 5, a user supplies not only the telephone number of a destination FAX machine, but also the ID of the intended destination FAX encoding/decoding device. When the digitally signed Y values are exchanged, the sending user device automatically "unsigns" the transmission by performing a modular squaring function; then compares the intended destination ID with the user ID returned with the Y value, and aborts the session if there is not a match. The key management steps previously described proceed automatically under control of the cryptographic processor 60, and when a session key has been derived, this is automatically applied in a conventional cryptographic process, such as the DES, to encrypt and decrypt a facsimile transmission.

It will be appreciated from the foregoing that the present invention represents a significant advance in cryptographic systems. In particular, the invention provides a technique for establishing a common session key for two users by means of an exchange of messages over an insecure communications channel. What distinguishes the invention from prior approaches to public key exchange systems is that the technique of the invention provides for identity authentication of the users without the need for a key distribution center or a public key register. Further, the technique is resistant to both passive and active eavesdropping. It will also be appreciated that, although an embodiment of the invention has been described in detail for purposes of illustration, various modifications may be made without departing from the spirit and scope of the invention. Accordingly, the invention is not to be limited except as by the appended claims.

Claims

1. A secure key generator, comprising:
storage means for storing a number of a first type selected prior to placing the key generator in service, and a digitally signed composite quantity containing both a unique and publicly known identifier of the key generator and a number of a second type obtained by a practically irreversible transformation of the number of the first type;
a first input connected to receive the number of the first type;
a second input connected to receive an input quantity transmitted over an insecure communications channel from another key generator, the input quantity being digitally signed and containing both a publicly known identifier of the other key generator and a number of the second type generated by a practically irreversible transformation of a number of the first type stored in the other key generator;
a first output for transmitting the stored, digitally signed composite quantity over the insecure communications channel to the other key generator;
a second output;
means for decoding the signed input quantity received at the second input, to obtain the identifier of the other key generator and the received number of the second type; and
means for generating a session key at the second output, by performing a practically irreversible transformation of the number of the second type received through the second input, using the number of the first type received through the first input.

2. A secure key generator as defined in claim 1, wherein the key generator further comprises:
a third input, connected to receive another number of the first type, generated randomly;
means for generating at the first output, for transmission with the digitally signed composite quantity, a number of the second type obtained by a practically irreversible transformation of the number of the first type received through the third input; and
means for receiving from the second input another number of the second type generated in and transmitted from the other key generator;
and wherein the means for generating a session key performs a practically irreversible transformation involving both numbers of the first type, received at the first and third inputs, and both numbers of the second type received at the second input, whereby a different session key may be generated for each message transmission session.

3. A secure key generator as defined in claim 1, wherein:
the number of the second type stored in digitally signed form in the storage means is obtained by the transformation $Ya = \alpha^{Xa} \bmod p$, where Xa is the number of the first type stored in the storage means, and α and p are publicly known transformation parameters;
the number of the second type received in the digitally signed composite quantity from the other key generator is designated Yb; and
the means for generating the session key performs the transformation $K = Yb^{Xa} \bmod p$.

4. A secure key generator as defined in claim 2, wherein:
the number of the second type stored in digitally signed form in the storage means is obtained by the transformation $Ya = \alpha^{Xa} \bmod p$, where Xa is the number of the first type stored in the storage means, and α and p are publicly known transformation parameters;

the number of the second type received in the digitally signed composite quantity from the other key generator is designated Yb; and
the means for generating the session key performs the transformation
$K = (Y'b)^{Xa} \bmod p \oplus (Yb)^{X'a} \bmod p$,
where $X'a$ is the number of the first type that is randomly generated, $Y'b$ is the additional number of the second type received from the other key generator, and the $\oplus$ symbol denotes an exclusive OR operation.

5. A method of generating a secure session key between two user devices connected by an insecure communications channel, comprising the following steps performed at both devices:
transmitting a digitally signed composite quantity to the other device, the composite quantity including a publicly known device identifier IDa and a number Ya derived by a practically irreversible transformation of a secret number Xa that it is unique to the device;
receiving a similarly structured digitally signed composite quantity from the other device;
transforming the received digitally signed composite quantity into an unsigned composite quantity containing a device identifier IDb of the other device and a number Yb that was derived by transformation from a secret number Xb that is unique to the other device;
verifying the identity of the other device from the device identifier IDb; and
generating a session key by performing a practically irreversible transformation involving the numbers Xa and Yb.

6. A method as defined in claim 5, and further including the steps of:
generating another number $X'a$ randomly prior to generation of a session key;
transforming the number $X'a$ to a number $Y'a$ using a practically irreversible transformation;
transmitting the number $Y'a$ to the other device; and
receiving a number $Y'b$ from the other device;
wherein the step of generating a session key includes a practically irreversible transformation involving the numbers Xa, $X'a$, Yb and $Y'b$.

7. A method as defined in claim 6, wherein:
the transformations from X numbers to Y numbers is of the type $Y = \alpha^X \bmod p$, where α and p are chosen to maximize irreversibility of the transformations; and
the step of generating a session key includes the transformation
$K = (Y'b)^{Xa} \bmod p \oplus (Yb)^{X'a} \bmod p$,
where $\oplus$ denotes an exclusive OR operation.

8. A method of authentication in a public key cryptographic system, the method comprising the steps of:
selecting a unique random number Xi for each cryptographic device to be distributed;
transforming the number Xi to a new number Yi using a practically irreversible transformation;
forming a composite quantity by combining the number Yi with a publicly known device identifier IDi;
digitally signing the composite quantity containing Yi and IDi;
storing the signed composite quantity and the number Xi permanently in each device;
exchanging, between two devices, a and b, desiring to establish secured communication, the signed composite quantities stored in each;
authenticating, in each of the two devices, the identity of the other device; and
generating, in each of the two devices, a session key to be used for secured communication.

9. A method as defined in claim 8, wherein the step of authenticating includes:
transforming the digitally signed composite quantity received from the other device into unsigned form; and
comparing the value of IDb in the unsigned quantity with the known IDb of the other device.

10. A method as defined in claim 9, wherein:
the step of generating the session key includes performing a transformation that involves a value Yb received from the other device and the value Xa of this device.

11. A method as defined in claim 10, wherein:
the step of digitally signing includes computing a modular square root of the composite quantity; and
the step of transforming the digitally signed composite quantity to unsigned form includes computing a modular square of the signed quantity.

12. A method as defined in claim 11, wherein:
the steps of computing a modular square root and computing a modular square both employ a modulus that is the product of two prime numbers.

13. A method as defined in claim 8, and further comprising the steps of:
transforming, in each of the two devices, the digitally signed composite quantity received from the other device into unsigned form; and
generating, in each of the two devices, a, b, a random number $X'a$, $X'b$;
transforming the numbers $X'a$, $X'b$ into numbers $Y'a$, $Y'b$ by a transformation that is practically irreversible; and
exchanging the numbers $Y'a$, $Y'b$ between the two devices;
and wherein the step of generating the session key includes performing a practically irreversible transformation involving the numbers Xa, $X'a$, Yb, and $Y'b$ in device a, and the numbers Xb, $X'b$, Ya, and $Y'a$ in device b.

14. A method as defined in claim 13, wherein:

the transformations from X numbers to Y numbers is of the type $Y = \alpha^X \bmod p$, where α and p are chosen to maximize irreversibility of the transformations; and

the step of generating a session key includes the transformations

$K = (Y'b)^{Xa} \bmod p \oplus (Yb)^{X'a} \bmod p$,

for device a, and

$K = (Y'a)^{Xb} \bmod p \oplus (Ya)^{X'b} \bmod p$,

for device b, where $\oplus$ denotes an exclusive OR operation.

15. A method as defined in claim 13, wherein:

the step of digitally signing includes computing a modular square root of the composite quantity; and

the step of transforming the digitally signed composite quantity to unsigned form includes computing a modular square of the signed quantity.

16. A method as defined in claim 15, wherein:

the steps of computing a modular square root and computing a modular square both employ a modulus that is the product of two prime numbers.

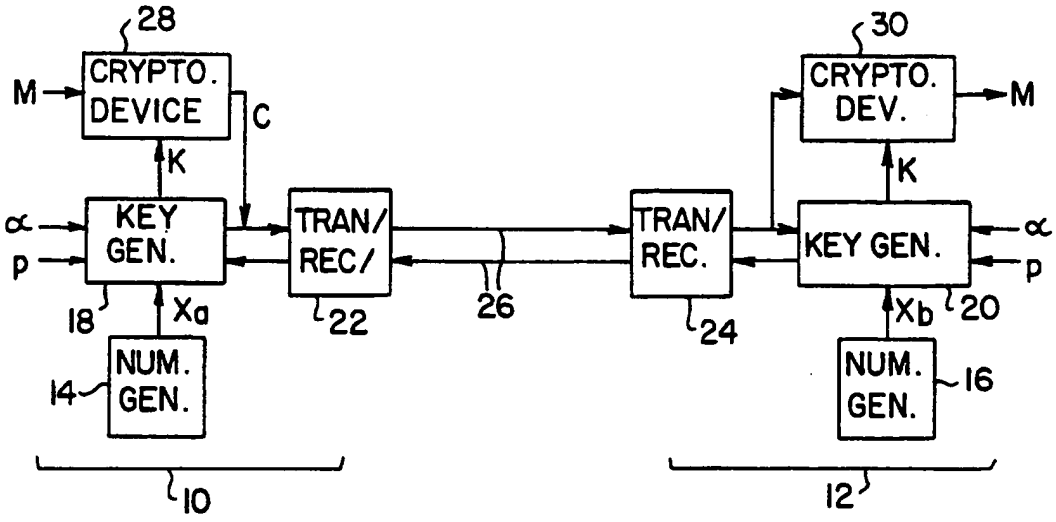


FIG. 1 (PRIOR ART)

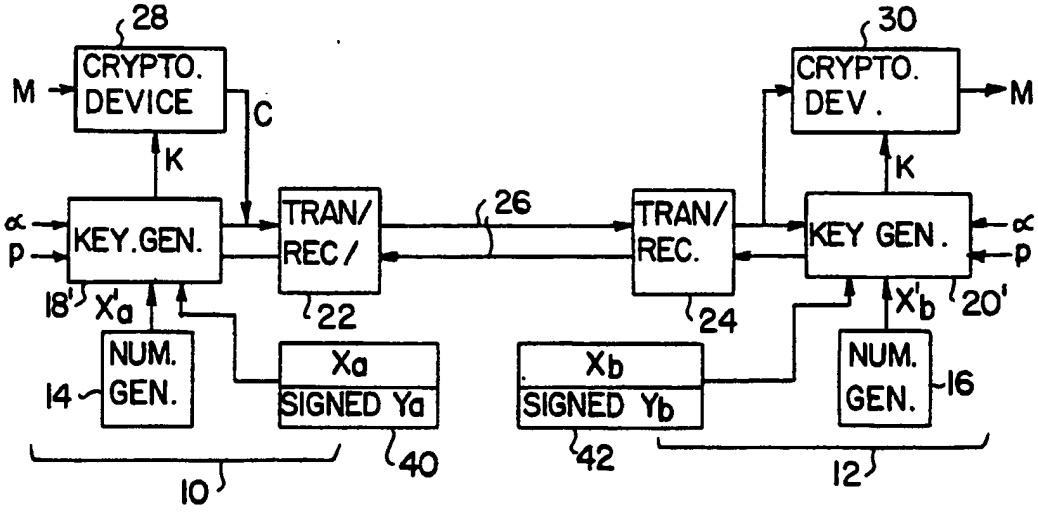


FIG. 3

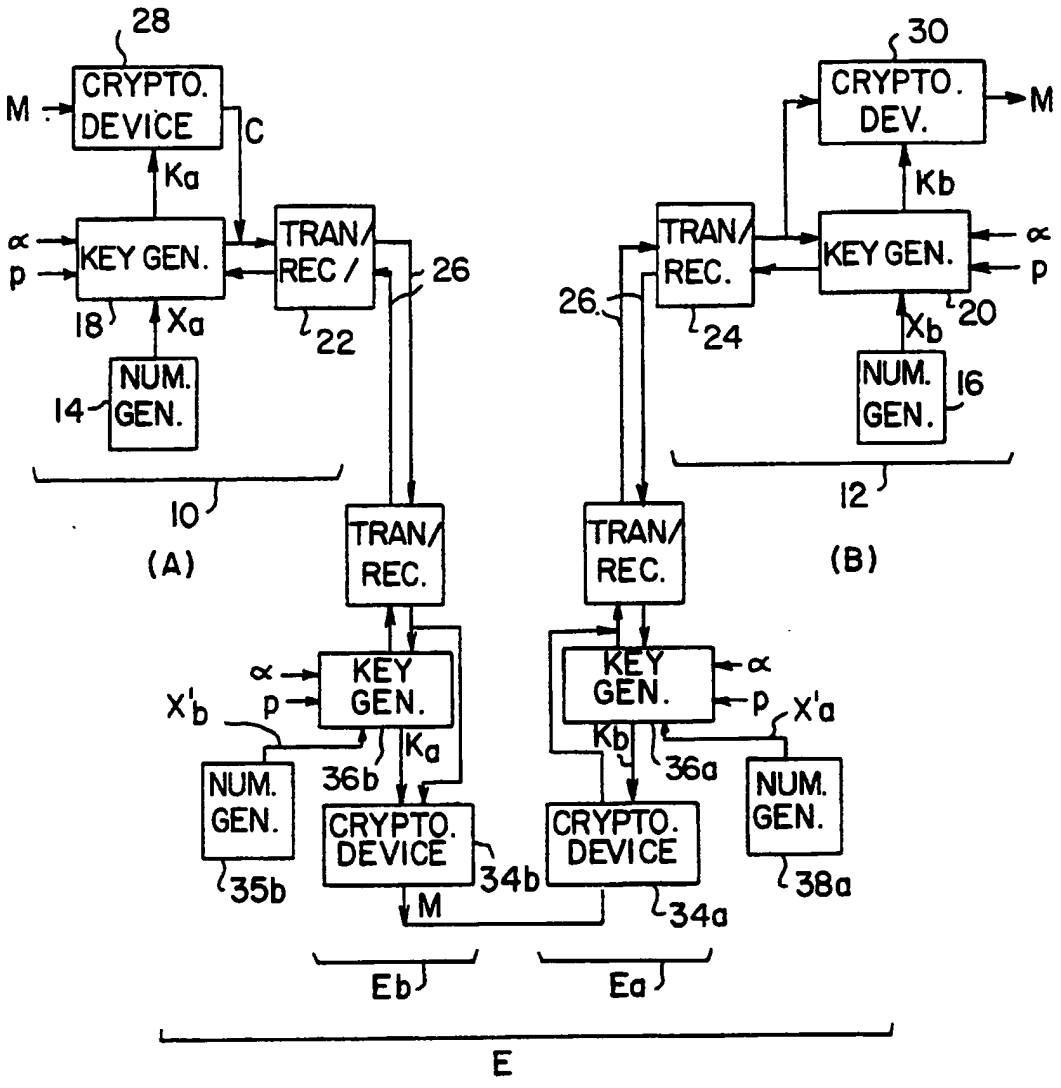
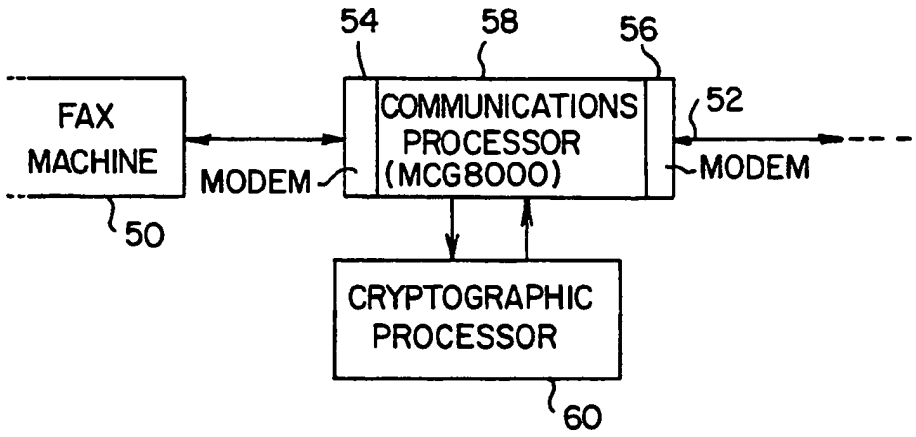


FIG. 2 (PRIOR ART)

FIG. 4


54
58
56
FAX MACHINE
MODEM
COMMUNICATIONS PROCESSOR (MCG8000)
52
MODEM
50
CRYPTOGRAPHIC PROCESSOR
60

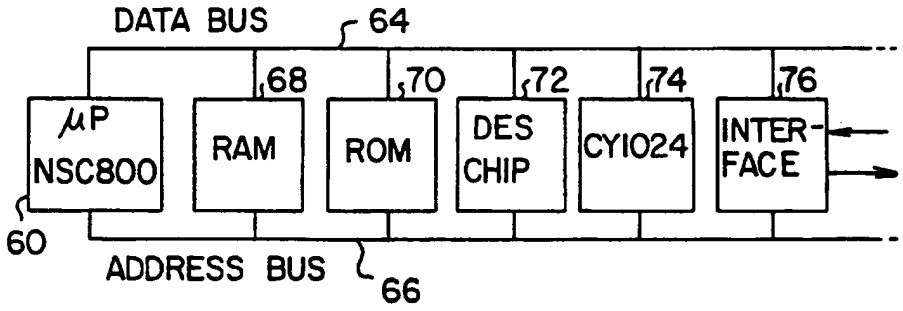

DATA BUS
64
68
70
72
74
76
μP NSC800
RAM
ROM
DES CHIP
CY1024
INTER-FACE
60
ADDRESS BUS
66

FIG. 5

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: 0 450 841 A2

(12) EUROPEAN PATENT APPLICATION

(21) Application number: 91302657.1

(51) Int. Cl.[5]: H04N 7/16

(22) Date of filing: 25.03.91

(30) Priority: 29.03.90 US 501620
29.03.90 US 501682
29.03.90 US 501683
29.03.90 US 561684
29.03.90 US 501685
29.03.90 US 501658

(43) Date of publication of application:
09.10.91 Bulletin 91/41

(84) Designated Contracting States:
BE DE FR GB IT

(71) Applicant: GTE LABORATORIES INCORPORATED
1209 Orange Street
Wilmington Delaware 01901 (US)

(72) Inventor: Walker, Stephen S.
117 Kelleher Road
Marlborough, MA 01752 (US)
Inventor: Sidlo, Clarence M.
5 Lowry Road
Framlingham, MA 01701 (US)
Inventor: Teare, Melvin J.
21 Woodleigh Road
Framlingham, MA 01701 (US)

(74) Representative: Bubb, Antony John Allen et al
GEE & CO. Chancery House Chancery Lane
London WC2A 1QU (GB)

(54) Video control system.

(57) A video control system includes a central facility (11) and a terminal (10). Video program means provided the terminal with a video program including a series of television fields including a first field containing both a random digital code encrypted according to a code encryption key and program identification data, and a second field containing an unintelligible video signal previously transformed from an intelligible video signal according to the random digital code. The terminal (10) includes means (22) for sending the program identification data to the central facility (11). The central facility includes a data base (19) for storing and retrieving at least one code encryption key corresponding to the program identification data and means (20) for sending the code encryption key from the central facility (11) to the terminal (10). The terminal (10) further includes means (22) for receiving the code encryption key from the central facility, decrypting means (23) for decrypting the encrypted digital code of the first frame in accordance with the code encryption key and means (24) for transforming the unintelligible video signal of the second frame to the intelligible video signal using the decrypted random digital code. The video program means may transmit the program to said terminal (10) or be located at the terminal (10) for playing a video recording medium storing the program.

EP 0 450 841 A2

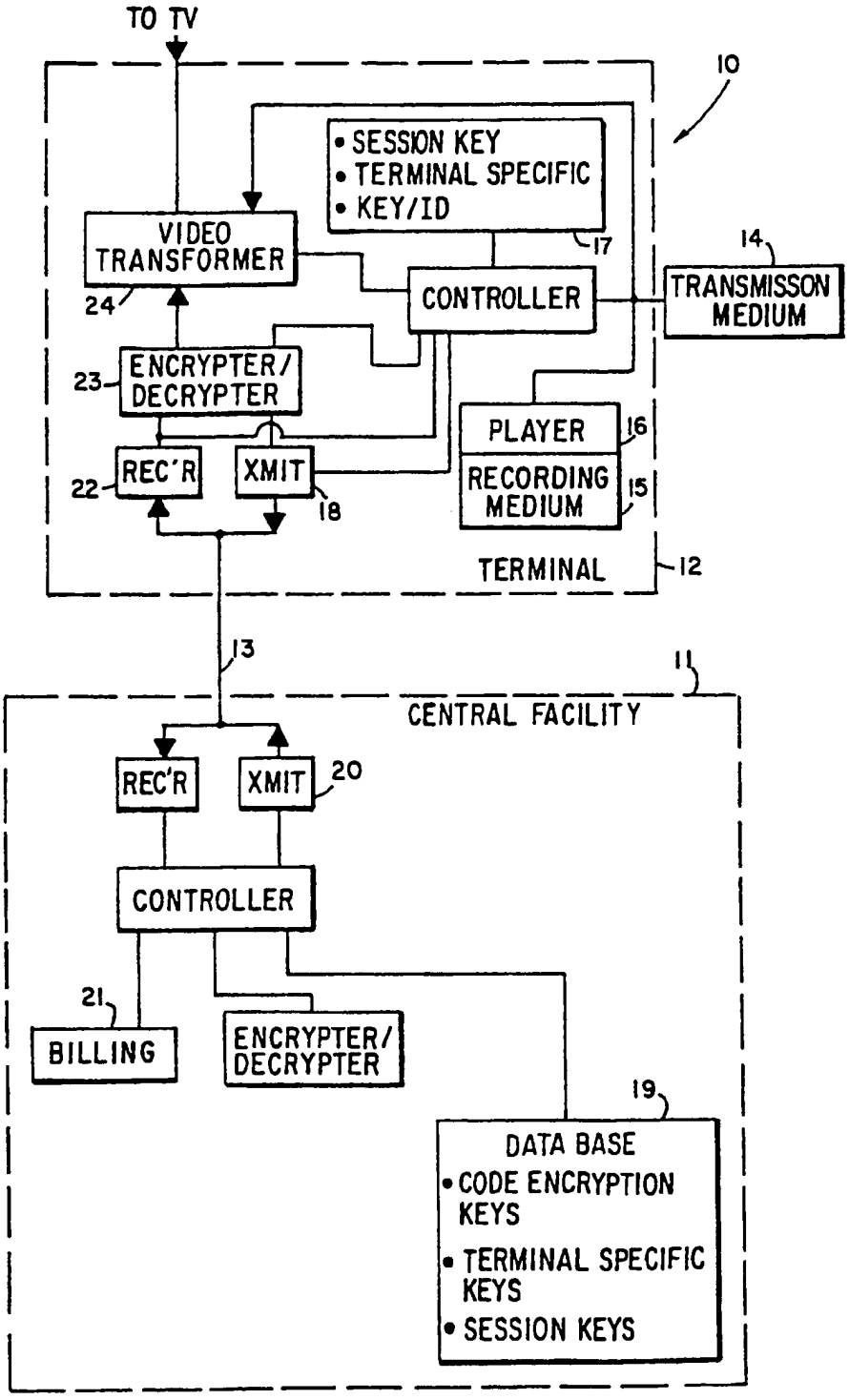

TO TV
10
• SESSION KEY
• TERMINAL SPECIFIC
• KEY/ID
17
VIDEO TRANSFORMER
24
CONTROLLER
14
TRANSMISSON MEDIUM
23
ENCRYPTER/ DECRYPTER
PLAYER
16
RECORDING MEDIUM
15
22
REC'R
XMIT
18
TERMINAL
12
13
11
CENTRAL FACILITY
REC'R
XMIT
20
CONTROLLER
21
BILLING
ENCRYPTER/ DECRYPTER
19
DATA BASE
• CODE ENCRYPTION KEYS
• TERMINAL SPECIFIC KEYS
• SESSION KEYS

FIG. 1

This invention is concerned with video control systems. It is desirable to provide a video control system which decrypts encrypted broadcasts or recorded copies of video material such that the subsequent viewing is controlled. This allows the owner to either forbid viewing, or collect revenue at his or her discretion.

In the prior art, a software distribution system is known wherein a computer program is downloaded once, followed by an access key to allow use of it on each subsequent use. This system uses a dynamic key that constantly changes, and is directly related to a user's decoder box, both by ID and an internal dynamic counter.

Also known is a video system that autonomously controls the viewing of a recording for either 24 hours or once only. It does not have the power of control desired.

Accordingly the present invention provides a video system comprising: a central facility; a terminal; and video program means for providing to said terminal a video program including a series of television fields including a first field containing both a random digital code encrypted according to a code encryption key and program identification data, and a second field containing an unintelligible video signal previously transformed from an intelligible video signal according to said random digital code; said terminal including means for sending said program identification data to said central facility; said central facility including a data base for storing and retrieving at least one code encryption key corresponding to the program identification data and means for sending said code encryption key from said central facility to said terminal; said terminal further including means for receiving the code encryption key from said central facility, decrypting means for decrypting the encrypted digital code of said first frame in accordance with said code encryption key and means for transforming said unintelligible video signal of said second frame to said intelligible video signal using the decrypted random digital code.

One embodiment of the invention will now be described, by way of example, with reference to the accompanying drawings in which:

Figure 1 is a block diagram of a video system embodying the invention; and

Figure 2 shows an encryption arrangement according to the invention.

Reference is made to Figure 1 which is a block diagram of a video system 10 embodying the invention. The video system comprises a central facility 11, a terminal 12, and a duplex communication link 13 between central facility 11 and terminal 12. An overview of the system is first given.

Terminal 12 is provided with a video program including a series of television fields including a first field containing both a random digital code encrypted according to a code encryption key and program identification data, and a second field containing an unintelligible video signal previously transformed from an intelligible video signal according to the random digital code.

The video program may be transmitted by broadcast, cable, satellite, fiber, or any other transmission medium 14. Alternative the video program may be stored on a video recording medium 15 such as magnetic tape or video disk and played by player 16. The unintelligible video signal may be either analog or digital.

A second field has a vertical blanking interval containing both a random digital code encrypted according to a code encryption key and program identification data, is followed by a third field containing an unintelligible video signal previously transformed from an intelligible video signal according to the random digital code of the second field.

Terminal 12 includes means 17 to store terminal identification data and means to send to the central facility 11 the terminal identification data and the program identification data over link 13.

Central facility 11 includes a data base 19 for storing and retrieving at least one code encryption key corresponding to the program identification data, means 20 for sending the code encryption key from the central facility 11 to the terminal 12, and means 21 for generating billing data based on both terminal identification data and program identification data.

Terminal 12 further including means 22 for receiving the code encryption key from central facility 11, decrypting means 23 for decrypting the encrypted random digital code of the first frame in accordance with the code encryption key, and means 24 for transforming the unintelligible video signal of the second frame to the intelligible video signal using the decrypted random digital code.

Each terminal 12 may have a terminal specific encryption key and means 18 to send to the central facility the program identification data and the terminal 11 identification data encrypted according to the terminal specific encryption key. The central facility 11 has means for storing a duplicate of the terminal specific encryption key, means for encrypting the code encryption key according to the terminal specific encryption key; and means for sending the encrypted code encryption key from central facility 11 to terminal 12.

Terminal 12 further includes means 22 for receiving the encrypted code encryption key from central facility 11, decryption means 23 for decrypting the code encryption key according to the terminal specific encryption key, and decrypting the encrypted random digital code of the first frame in accordance with the code encryption key, and means 24 for transforming the unintelligible video signal of the second frame to the intelligible video signal using the decrypted ran-

dom digital code.

Terminal 12 includes means to encrypt the terminal identification data according to the terminal specific encryption key, means to send unencrypted terminal identification data and encrypted terminal identification data to the central facility, which in turn includes means to compare unencrypted and encrypted terminal identification data to verify terminal identity.

A plurality of code encryption keys may be used for one program wherein a desired code encryption key is selected from the plurality of code encryption keys in accordance with code encryption key identification data corresponding to the random digital code.

Various features of the system are now discussed in more detail.

System 10 controls the viewing of video programs, by which is meant any video material, either transmitted or recorded, in television format consisting of a series of fields of lines. Two interlaced fields make up a television frame.

Video programs are rendered unintelligible, e.g. scrambled, by any analog or digital method, and are made intelligible, e.g. descrambled, using random digital codes located in fields. The random digital keys are themselves encrypted, and decrypted by a one or more key obtained from a database located at the central facility, along with user-specific information at the time of viewing. The system does not stop copying, it controls viewing, while protecting revenues. As such, it can encourage copying, which could ease the distribution issue by controlling the playback such that revenue can be collected each time.

Preferably duplex communication link 13 is a continuous data channel between a terminal and a central facility such as an ISDN D-channel or by modem over a regular phone line.

The video program is encrypted, and needs a decrypter in the terminal for viewing. The decrypter uses data embedded in the video program along with a data access to correctly perform the decryption, so the process is completely controlled. The embedded data and key transfer from the remote database may be protected with public domain encryption techniques, providing high level security before first viewing.

The video program may be recorded as is, but it is still unviewable. To view it, the decrypter is used, along with the encrypted embedded data, and an access to a secure database, to perform the decryption. Recordings may be freely copied, but remain unviewable unless used with the decrypter.

To view the programs requires access to the database using encrypted data transfer. This process yields the control of the video program, whether recording or trans mission. The decrypter requires one or more keys that arrives from the database. To get the key, information from the video program as well as terminal identification is sent to the database.

A direct Electronic funds Transfer (EFT) debit can be performed using the information. It the program is a video store copy, the EFT could include the store fee and the copyright fee. Note that the video distribution to video stores becomes trivial, as they are encouraged to take a direct recording with a video store key, along with their authorized converter box, and make as many copies as they like. The revenue control takes place at viewing time. This encourages a shareware type of distribution.

A passkey can be sent to the database, to allow viewing of questionable taste films by adults, controlling access by minors.

On the first access, the database will capture a signature derived from the user's equipment and the recording, and store it for subsequent tracking. As there is a compelled database access in this process, data on usage may be collected. This same process may be used for revenue collection.

The system preferably uses at least one downloadable key, an encrypted video program that uses the key for decryption, and data stored in a field of the video program. It may be implemented in an all digital, analog, or mixed analog/digital environment.

The video programs are encrypted, with data relating to the programs, e.g. where and when, who transmitted it. The data may also contain part of the decryption key. This information would be extracted from the signal, and used to access a database, maintained by the program's owners, to obtain an encrypted key for the decrypter. After a subscriber and/or a credit check is successfully completed, the one or more keys would be transmitted. At this time the owner has obtained usage data, with a specific user's ID, and has the option of billing him. If it is a free program, at least the viewer data is available.

If a user records a transmission or another recording, he captures the encrypted signal, along with embedded data, as described above. This accomplishes the signature part of the process. A recording created by this method may be on a regular VCR, but is encrypted and individually marked. Copying a recording does not affect the system, as the rerecording is only usable with the correct keys. Potentially, the first few minutes of a program might be viewable without the need of a key, to allow the user to see what the contents of the program are, as well as to allow time for the database access and key synchronization process.

To play a recording back, it is necessary to re-obtain the one or more keys. The combination of data stored in a field is used to access the database. Before the keys are made available, there is a check that the terminal identification and the embedded data match.

In the case wherein a recording is rented from a video store, a code may identify the store. The database recognizes the recording as a rental copy, and

charge either the user or the video store a fee. If the recording is viewed a second time, the charge is repeated. In the event a copy is made, when it is played, the database will identify the originating video store, but not the actual copier. However, if validation is performed at rental time, there would be some measure of control. If the entire charging process were to be reversed, such that the viewer carries all the liability for charges, then copying is encouraged, as per shareware, and the distribution problem is minimized, while revenues are maintained on a usage basis.

The program's owner has the responsibility to get a secured copy to whoever deals with the distribution of the programs. The programs are encrypted, and require a database update to enable viewers to make use of the program. The viewer has a terminal including a decrypter, linked to the central facility's database via an automatic dial-up, that, when enabled, decrypts the video program. As appropriate, there can be credit checks and billing from the database, as well as statistics collection.

The encryption has two levels, one for protection of video decryption codes on the program, and one for protection of messages between the terminal and the central facility. Both may use the NBS Data Encryption Standard (DES).

DES encryption and decryption may be implemented with a commercial Motorola 6859 Data Security Device or similar product at the terminal and at the central facility.

The decryption code itself is protected by being DES-encrypted. The decryption key is not on the video program but is retained in the database at the central facility. A program identification number and a decryption key number allow the central facility to recover the decryption key itself and send it to the terminal for decrypting the decryption codes.

A different DES decryption key is not required for every field. One key can span several fields. DES key requests and acknowledgements from the terminal may also act as keep-alive messages to the central facility.

DES decryption keys are transmitted from the central facility to the terminal protected by a higher-level DES "session" key. terminal requests for new keys as the tape progresses are also protected by the DES session key. This key is generated by the central facility at the beginning of the session and remains valid for the duration of the session. The terminal begins the session using a terminal-unique DES key stored in a ROM.

Frame contents are transferred from the Analog Subsystem to the DCSS and the decrypted decryption code from the DCSS to the Analog Subsystem over the analog interface shown in the Figure. Transfer of data between the subsystems may be coordinated by means of the vertical and horizontal blanking signals and their derivative interrupts.

All messages between terminal and central facility use Cyclic Redundancy Code (CRC) checking to verify message integrity. The CRC-CCITT generating polynomial generates two block check characters (BCC) for each message. If the terminal receives a message that is not verified by the BCC, it sends a request (ARQ) to the central facility to retransmit the last message. The central facility does not attempt to ARQ garbled messages. It discards them and waits for a terminal to send again.

Message exchange in the VCS is by a positive acknowledgent scheme in which a response of some kind is expected for every message sent. For example, a terminal expects a DES decryption key message after it sends a request for the same; the central facility expects a key receipt acknowledge after it sends the key message.

When a user begins to play a protected program, the terminal initiates a session by sending a "session start" message (STS) to the central facility containing user and program identifications. The message contains message type, user number and CRC code in the clear, but the balance of the message is DES-encrypted with the initial DES session key stored in the terminal ROM. (The user identification is also stored in ROM.) The central facility uses the unencrypted data to access its database and find the user DES value for decrypting the remainder of the message.

The central facility authenticates the message by comparing clear and decrypted user numbers. If the user numbers are identical, the central facility then confirms that the program serial number is valid. The central facility may also check user credit. If all is well, the central facility accepts the session and generates a new (and random) DES key that is unique for that session. It encrypts this using the initial user value in the database and sends it to the terminal, which decrypts the message and stores the new value in its database (MCU RAM) as the session key for the remainder of the session.

The central facility then uses the tape and decryption key number in the STS message to recover a set of DES decryption keys for the program from the database. These are encrypted with the session key and sent to the terminal at the start of a session or during the course of a session.

The terminal generates session start, key acknowledgement, and ARQ messages. The central facility responds in kind. Both the central facility and the terminal generate and verify block check characters.

The preferred embodiment and best mode of practicing the invention have been described. Alternatives now will be apparent to those skilled in the art in light of these teachings. Accordingly the invention is to be defined by the following claims and not by the particular examples given.

Claims

1. A video system comprising:
a central facility;
a terminal; and
video program means for providing to said terminal a video program including a series of television fields including a first field containing both a random digital code encrypted according to a code encryption key and program identification data, and a second field containing an unintelligible video signal previously transformed from an intelligible video-signal according to said random digital code;
said terminal including means for sending said program identification data to said central facility;
said central facility including a data base for storing and retrieving at least one code encryption key corresponding to the program identification data and means for sending said code encryption key from said central facility to said terminal;
said terminal further including means for receiving the code encryption key from said central facility, decrypting means for decrypting the encrypted digital code of said first frame in accordance with said code encryption key and means for transforming said unintelligible video signal of said second frame to said intelligible video signal using the decrypted random digital code.

2. The system of claim 1 wherein a plurality of code encryption keys are used for one program, and wherein a desired code encryption key is selected from said plurality of code encryption keys in accordance with code encryption key identification data corresponding to the random digital code encrypted with said desired code encryption key.

3. The system of claim 1 or 2 wherein said video program means is means for transmitting said program to said terminal.

4. The system of claim 3 wherein said means for transmitting is a CATV system.

5. The system of any one of claims 1-4 wherein:
said terminal further includes means to store terminal identification data and a terminal specific encryption key; and means to send to said central facility said terminal identification data with said program identification data;
said central facility further includes means for storing a duplicate of said terminal specific encryption key; means for encrypting said code encryption key according to said terminal specific encryption key; and means for sending the encrypted code encryption key from said central facility to said terminal; and
said terminal further further includes means for receiving the encrypted code encryption key from said central facility; and decryption means for decrypting said code encryption key according to said terminal specific encryption key.

6. The video system of any one of claims 1-4 wherein:
said terminal further includes means to store terminal identification data and a terminal specific encryption key; and means to send to said central facility said program identification data and said terminal identification data,
said central facility further includes means for providing a session encryption key; means for encrypting said session encryption key according to said terminal specific encryption key; means for sending the encrypted session encryption key from said central facility to said terminal;
means for encrypting said code encryption key according to said encrypted session encryption key; and means for sending the encrypted code encryption key from said central facility to said terminal; and
said terminal further includes means for receiving the encrypted session encryption key from said central facility; decryption means for decrypting said session encryption key according to said terminal specific encryption key, means for receiving the encrypted code encryption key from said central facility; and decryption means for decrypting said code encryption key according to said session encryption encryption key.

7. The system of claim 5 or 6 wherein said terminal includes means to encrypt said terminal identification data according to said terminal specific encryption key, and means to send unencrypted terminal identification data and encrypted terminal identification data to said central facility, and said central facility includes means to compare unencrypted and encrypted terminal identification data to authenticate terminal identity.

8. The system of any one of claims 5-7 wherein said central facility further includes means for generating billing data based on said terminal identification data and said program identification data.

9. The video system of any one of claims 1-8 wherein said video program means is a means located at said terminal for playing a video recording medium storing said program.

10. A video recording medium storing a video program including a series of television fields including a first field containing both a random digital code encrypted according to a code encryption key and program identification data, and a second field containing an unintelligible video signal previously transformed from an intelligible video signal according to said random digital code.

11. The medium of claim 10 wherein a plurality of code encryption keys are used for one program, and wherein a desired code encryption key is selected from said plurality of code encryption keys in accordance with code encryption key identification data corresponding to the random digital code encrypted with said desired code encryption key.

12. The medium of claim 10 or 11 wherein said second field has a vertical blanking interval containing both a random digital code encrypted according to a code encryption key and program identification data, and is followed by a third field containing an unintelligible video signal previously transformed from an intelligible video signal according to said random digital code of the second field.

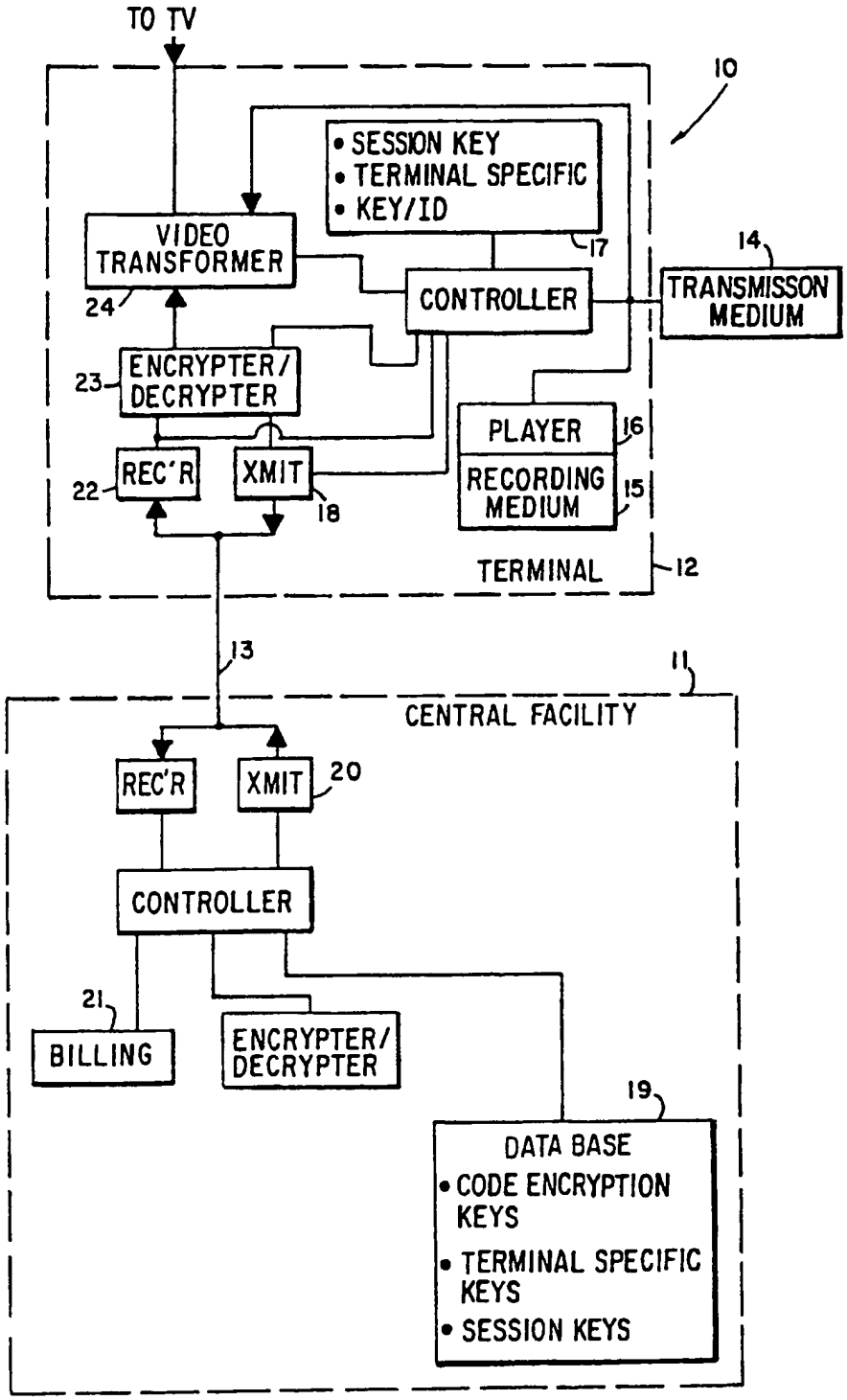


FIG.1

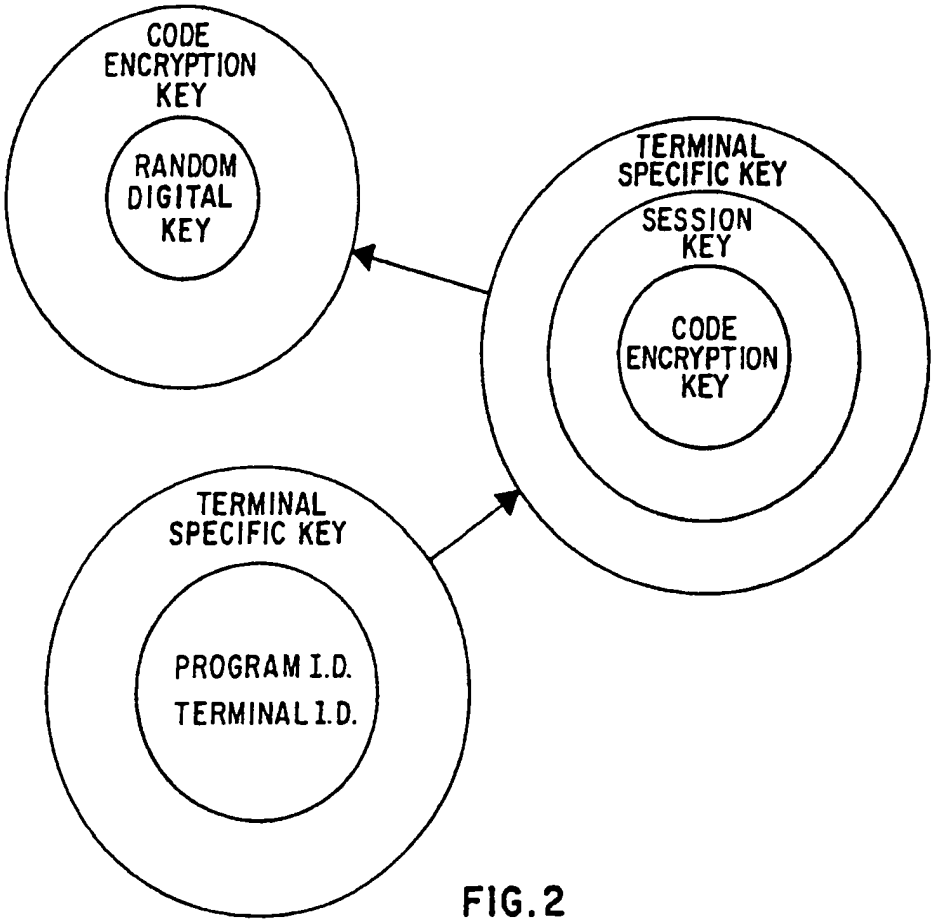


FIG. 2

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 529 261 A2**

(12) EUROPEAN PATENT APPLICATION

(21) Application number: **92111758.6**

(51) Int. Cl.5: **H04L 9/08**

(22) Date of filing: **10.07.92**

(30) Priority: **22.08.91 US 748407**

(43) Date of publication of application:
03.03.93 Bulletin 93/09

(84) Designated Contracting States:
CH DE FR GB IT LI NL SE

(71) Applicant: **International Business Machines Corporation**
Old Orchard Road
Armonk, N.Y. 10504(US)

(72) Inventor: **Matyas, Stephen M.**
10298 Cedar Ridge Drive
Manassas, VA 22110(US)
Inventor: **Johnson, Donald B.**
11635 Crystal Creek Lane
Manassa, VA 22111(US)
Inventor: **Le, An V.**
10227 Battlefield Drive
Manassas, Va 22110(US)
Inventor: **Martin, William C.**
1835 Hilliard Lane
Concord, NC 28025(US)
Inventor: **Prymak, Rostislaw**
15900 Fairway Drive
Dumfries, VA 22026(US)
Inventor: **Rohland, William S.**
4234 Rotunda Road
Charlotte, NC 28226(US)
Inventor: **Wilkins, John D.**
P.O. Box 8
Somerville, VA 22739(US)

(74) Representative: **Herzog, Friedrich Joachim, Dipl.-Ing.**
IBM Deutschland GmbH, Patentwesen und Urheberrecht, Schönaicher Strasse 220
W-7030 Böblingen (DE)

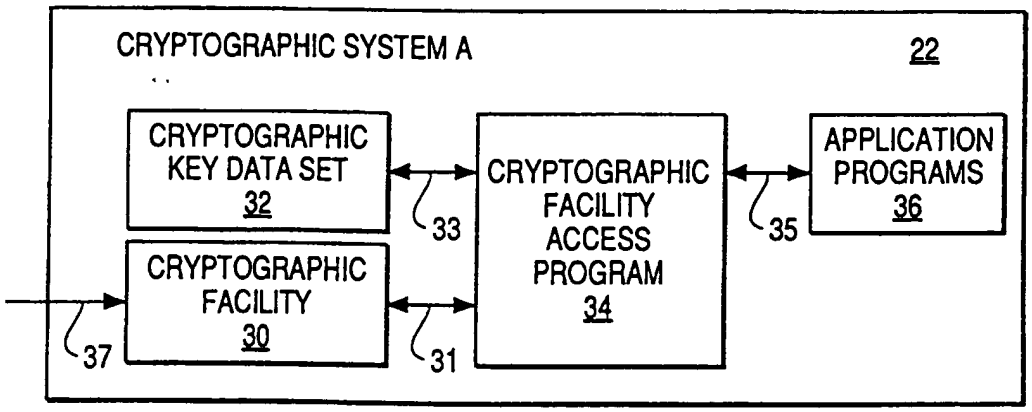
(54) A hybrid public key algorithm/data encryption algorithm key distribution method based on control vectors.

(57) The patent describes a method and apparatus for securely distributing an initial Data Encryption Algorithm (DEA) key-encrypting key by encrypting a key record (consisting of the key-encrypting key and control information associated with that key-encrypting key) using a public key algorithm and a public key belonging to the intended recipient of the key record. The patent further describes a method and apparatus for securely recovering the distributed key-encrypting key by the recipient by decrypting the received key record using the same public key algorithm and private key associated with the public key and re-encrypting the key-encrypting key under a key formed by arithmetically combining the recipient's master key with a control vector contained in the control information of the received key record. Thus the type and usage attributes assigned by the originator of the key-encrypting key in the form of a control vector are cryptographically coupled to the key-encrypting key such that the recipient may only use the received key-encrypting key in a manner defined by the key originator.

The patent further describes a method and apparatus to improve the integrity of the key distribution process by applying a digital signature to the key record and by including identifying information (i.e., an originator identifier) in the control information of the key record. The integrity of the distribution process is enhanced by verifying the digital signature and originator identifier at the recipient node.

EP 0 529 261 A2

FIG. 2



The invention disclosed broadly relates to data processing systems and methods and more particularly relates to cryptographic systems and methods for use in data processing systems to enhance security.

The following patents are related to this invention and are incorporated herein by reference:

B. Brachtl, et al., "Controlled Use of Cryptographic Keys Via Generating Stations Established Control Values," USP 4,850,017, issued July 18, 1989, assigned to IBM Corporation, and incorporated herein by reference.

S. M. Matyas, et al., "Secure Management of Keys Using Control Vectors," USP 4,941,176, issued July 10, 1990, assigned to IBM Corporation and incorporated herein by reference.

S. M. Matyas, et al., "Data Cryptography Operaions Using Control Vectors," USP 4,918,728, issued April 17, 1990, assigned to IBM Corporation, and incorporated herein by reference.

S. M. Matyas, et al., "Personal Identification Number Processing Using Control Vectors," USP 4,924,514, issued May 8, 1990, assigned to IBM Corporation and incorporated herein by reference.

S. M. Matyas, et al., "Secure Management of Keys Using Extended Control Vectors," USP 4,924,515, issued May 8, 1990, assigned to IBM Corporation and incorporated herein by reference.

S. M. Matyas, et al., "Secure Key Management Using Programmable Control Vector Checking," USP 5,007,089, issued April 9, 1991, assigned to IBM Corporation and incorporated herein by reference.

B. Brachtl, et al., "Data Authentication Using Modification Detection Codes Based on a Public One Way Encryption Function," USP 4,908,861, issued March 13, 1990, assigned to IBM Corporation and incorporated herein by reference.

D. Abraham, et al., "Smart Card Having External Programming Capability and Method of Making Same," serial number 004,501, filed January 19, 1987, assigned to IBM Corporation, and incorporated herein by reference.

S. M. Matyas, et al., "Method and Apparatus for Controlling the Use of a Public Key, Based on the Level of Import Integrity for the Key," serial number 07/602,989, filed October 24, 1990, assigned to the IBM Corporation.

S. M. Matyas, et al., "Secure Key Management Using Programmable Control Vector Checking," USP 5,007,089, issued April 9, 1991, assigned to IBM Corporation and incorporated herein by reference.

The cryptographic architecture described in the cited patents by S. M. Matyas, et al. is based on associating with a cryptographic key, a control vector which provides the authorization for the uses of the key intended by the originator of the key. The cryptographic architecture described in the cited patents by S. M. Matyas, et al. is based on the Data Encryption Algorithm (DEA), whereas the present invention is based on both a secret key algorithm, such as the DEA, and a public key algorithm. Various key management functions, data cryptography functions, and other data processing functions are possible using control vectors, in accordance with the invention. A system administrator can exercise flexibility in the implementation of his security policy by selecting appropriate control vectors in accordance with the invention. A cryptographic facility (CF) in the cryptographic architecture is described in the above cited patents by S. M. Matyas, et al. The CF is an instruction processor for a set of cryptographic instructions, implementing encryption methods and key generation methods. A memory in the crypto facility stores a set of internal cryptographic variables. Each cryptographic instruction is described in terms of a sequence of processing steps required to transform a set of input-parameters to a set of output parameters. A cryptographic facility application program is also described in the referenced patents and patent applications, which defines an invocation method, as a calling sequence, for each cryptographic instruction consisting of an instruction mnemonic and an address with corresponding input and output parameters.

Public key encryption algorithms are described in a paper by W. Diffie and M. E. Hellman entitled "Privacy and Authentication: An Introduction to Cryptography," Proceedings of the IEEE, Vol. 67, No. 3, March 1979, pp. 397-427. Public key systems are based on dispensing with the secret key distribution channel, as long as the channel has a sufficient level of integrity. In a public key crypto system, two keys are used, one for enciphering and one for deciphering. Public key algorithm systems are designed so that it is easy to generate a random pair of inverse keys PU for enciphering and PR for deciphering and it is easy to operate with PU and PR, but is computationally infeasible to compute PR from PU. Each user generates a pair of inverse transforms, PU and PR. He keeps the deciphering transformation PR secret, and makes the enciphering transformation PU public by placing it in a public directory. Anyone can now encrypt messages and send them to the user, but no one else can decipher messages intended for him. It is possible, and often desirable, to encipher with PU and decipher with PR. For this reason, PU is usually referred to as a public key and PR is usually referred to as a private key. A corollary feature of public key crypto systems is the provision of a digital signature which uniquely identifies the sender of a message. If user A wishes to send a signed message M to user B, he operates on it

with his private key PR to produce the signed message S. PR was used as A's deciphering key when privacy was desired, but it is now used as his "enciphering" key. When user B receives the message S, he can recover the message M by operating on the ciphertext S with A's public PU. By successfully decrypting A's message, the receiver B has conclusive proof it came from the sender A. Examples of public key cryptography are provided in the following U. S. patents:

USP 4,218,582 to Hellman, et al., "Public Key Cryptographic Apparatus and Method;" USP 4,200,770 to Hellman, et al., "Cryptographic Apparatus and Method;" and USP 4,405,829 to Rivest, et al., "Cryptographic Communications System and Method," which discloses the RSA public-key algorithm.

In general, it is preferable for performance reasons to use symmetric algorithms such as the Data Encryption Algorithm (DEA) bulk data encryption rather to use a public key algorithm for such purposes. However to use DEA both the data originator and intended recipient must first share a common, secret key. This requires the secure distribution of at least one DEA key for each secure "channel" between originator and recipient. The problem can be reduced to distributing one secret DEA key-encrypting key (KEK) between the originating node and receiving node, and thereafter transmitting all other DEA keys encrypted under this common KEK. The usual method of distributing the initial KEK is via trusted couriers.

It is well-known that a hybrid system employing a public key algorithm and the DEA may be effective in solving the initial KEK distribution problem, while still retaining the faster bulk data encryption capabilities of the DEA. In such a hybrid cryptographic system A, a public key PU is transmitted with integrity (see S. M. Matyas, et al., "Method and Apparatus for Controlling the Use of a Public Key, Based on the Level of Import Integrity for the Key", serial number 07/602,989, filed October 24, 1990) to a second hybrid cryptographic system B. A secret DEA KEK, say KK, is generated and encrypted under PU at system B and transmitted to system A. System A uses the corresponding private key PR to decrypt KK. KK may then be used with the DEA algorithm to distribute additional DEA keys for use by systems A and B.

Prior art, however, has not provided a cryptographically secure means to define the type and to control the usage of the generated KEK to insure that the type and uses defined by the originator of the key (system B) are enforced at both the originating node and the recipient node (system A). Without such controls (as described in S. M. Matyas, et al., "Secure Management of Keys Using Control Vectors", USP 4,941,176, issued July 10, 1990), the distributed KEK may be subject to misuse by either party to weaken the security of the system (e.g., by allowing the KEK to be used in a data decrypt operation and thus allowing DEA keys encrypted under the KEK to be decrypted and exposed in the clear).

While the prior art addresses the concept of unidirectional key-encrypting keys, i.e., key-encrypting keys that establish a key distribution channel in one direction only, the method for establishing, with integrity, such a unidirectional channel using a public key algorithm has not been addressed. To accomplish this, a unique Environment Identifier (EID) is stored at each cryptographic device such that a distributed key-encrypting key can be imported only at the designated receiving device, but it does not allow the key-encrypting key to be imported or re-imported at the sending device, as described below.

The originating node B generates the KEK in two forms: one form to be exported to the recipient node A (encrypted under the PU received from A) and a second form to be used at B ultimately to either export or import additional DEA keys (encrypted under some form of the local master key). As was described in the above reference U. S. patent 4,941,176, "Secure Management of Keys Using Control Vectors," it is critical to the security of each cryptographic system that the type and usage attributes of a given KEK on one system be limited to either EXPORTER usage or IMPORTER usage, but never both. Correspondingly, it must not be possible to generate or introduce two copies of the same KEK into the system, one with EXPORTER usage and one with IMPORTER usage. Such a pair of key forms is known as a bi-functional key pair.

Prior art has provided no cryptographically secure means to insure that the generated KEK cannot be re-imported into the originating node to form a bi-functional key pair. Since key PU is public, system A cannot be certain that system B is the originator of the generated KEK.

It is therefore a main object of the invention to provide an improved method for distributing DEA keys using a public key crypto system.

It is another object of the invention to provide an improved method of distributing a DEA key-encrypting key using a public key crypto system.

It is another object of the invention to provide an improved method of distributing a DEA key-encrypting key that does not require the use of couriers.

It is another object of the invention to provide control information associated with a distributed key, which defines the type and usage of the distributed key.

It is another object of the invention to provide a means to cryptographically couple the control information and key using a public-key algorithm.

It is another object of the invention to provide control information that prevents a distributed key from being imported at the originating device.

It is another object of the invention to provide a method of key distribution which is compatible with a key management based on control vectors (in the above referenced patents).

It is another object of the invention to provide a method of key distribution that does not also provide a covert privacy channel.

It is another object of the invention to provide a means for a receiving device to validate that a received distributed key has originated with an expected originating device.

It is another object of the invention to provide a means for a distributed key to be authenticated on the basis of a signature generated on the distributed key by the cryptographic system software.

It is still a further object of the invention to provide a higher integrity means for a distributed key to be authenticated on the basis of a signature generated on the distributed key as an integral part of the cryptographic system hardware export function.

These and other objects, features, and advantages are accomplished by the invention disclosed herein. A method and apparatus are disclosed for generating and distributing a DEA key-encrypting key from a sending device implementing a public-key cryptographic system to a receiving device implementing a public-key cryptographic system. The method and apparatus find application in a cryptographic system implementing both a symmetric encryption algorithm, such as the Data Encryption Standard, and an asymmetric encryption algorithm, such as the RSA public-key algorithm. The method begins by generating a key-encrypting key at a sending device and producing two encrypted copies of the generated key. The generated key is encrypted first under the public key of a designated receiving device and the encrypted key is then electronically transmitted to the receiving device. The generated key is also encrypted under the master key of the sending device and stored in a key storage for later use in a DEA key management scheme for distributing further DEA keys to the designated receiving device. At the receiving device, the encrypted key is decrypted using the private key of the receiving device and the clear key is then re-encrypted under the master key of the receiving device and the encrypted key is stored in a key storage for later use in a DEA key management scheme for receiving further DEA keys from the same sending device. In accordance with the invention, the method of key distribution makes use of a key block containing the distributed key-encrypting key and control information associated with the distributed key, which includes a control vector to limit uses of the key and an environment ID to identify the sender of the key. The method of key distribution also makes use of an optional digital signature generated on the encrypted key block at the originating device and validated at the receiving device.

These and other objects, features, and advantages of the invention will be more fully appreciated with reference to the accompanying figures.

Fig. 1 illustrates a communications network 10 including a plurality of data processors, each of which includes a cryptographic system;

Fig. 2 is a block diagram of a cryptographic system 22;

Fig. 3 is a block diagram of a cryptographic facility 30;

Fig. 4 is a block diagram showing the public and private keys that must first be initialized at two cryptographic systems A and B in order that they may electronically distribute DEA keys using a public key algorithm;

Fig. 5 is a block diagram illustrating DEA key distribution using the GKSP and IDK instructions without digital signatures;

Fig. 6 is a block diagram of a key block;

Fig. 7 is a block diagram of an external key token;

Fig. 8 is a block diagram illustrating DEA key distribution using the GKSP and IDK instructions with digital signatures;

Fig. 9 is a block diagram of the Generate Key Set PKA (GKSP) instruction;

Fig. 10 is a block diagram of the Import DEA Key (IDK) instruction;

Fig. 11 is a block diagram of control vectors for public and private keys used for key distribution (i.e., key management purposes);

Fig. 12 is a block diagram depicting an encrypted channel and a clear channel between two cryptographic systems A and B;

Fig. 13 is a block diagram illustrating the processing of control information at a receiving cryptographic device;

Fig. 14 is a block diagram of a cryptographic facility at a sending location, in accordance with the invention;

Fig. 15 is a block diagram of a cryptographic facility at a receiving location, in accordance with the invention;

Fig. 16 is a block diagram of the crypto-variable retrieval means 40 which is a component of the cryptographic facility shown in Fig. 14.

Environment Description: Fig. 1 illustrates a network block diagram showing a communications network 10 to which is connected a plurality of data

processors including data processor 20, data processor 20', and data processor 20". Also included in each data processor is a cryptographic system, as shown in Fig. 1. Data processor 20 includes cryptographic system 22, data processor 20' includes cryptographic system 22' and data processor 20" includes cryptographic system 22". Each data processor supports the processing of one or more applications which require access to cryptographic services such as for the encryption, decryption and authenticating of application data and the generation and installation of cryptographic keys. The cryptographic services are provided by a secure cryptographic facility in each cryptographic system. The network provides the means for the data processors to send and receive encrypted data and keys. Various protocols, that is, formats and procedural rules, govern the exchange of cryptographic quantities between communicating data processors in order to ensure the interoperability between them.

Fig. 2 illustrates the cryptographic system 22. In the cryptographic system 22, the cryptographic facility (CF) 30 has an input 37 from a physical interface. The cryptographic facility access program (CFAP) 34 is coupled to the cryptographic facility 30 by means of the interface 31. The cryptographic key data set (CKDS) 32 is connected to the cryptographic facility access program 34 by means of the interface 33. The application programs (APPL) 36 are connected to the cryptographic facility access program 34 by means of the interface 35.

A typical request for cryptographic service is initiated by APPL 36 via a function call to the CFAP 34 at the interface 35. The service request includes key and data parameters, as well as key identifiers which the CFAP 34 uses to access encrypted keys from the CKDS 32 at the interface 33. The CFAP 34 processes the service request by issuing one or more cryptographic access instructions to the CF 30 at the interface 31. The CF 30 may also have an optional physical interface 37 for direct entry of cryptographic variables into the CF 30. Each cryptographic access instruction invoked at the interface 31 has a set of input parameters processed by the CF 30 to produce a set of output parameters returned by the CF 30 to the CFAP 34. In turn, the CFAP 34 may return output parameters to the APPL 36. The CFAP 34 may also use the output parameters and input parameters to subsequently invoke instructions. If the output parameters contain encrypted keys, then the CFAP 34, in many cases, may store these encrypted keys in the CKDS 32.

Fig. 3 illustrates the cryptographic facility 30. The cryptographic facility 30 is maintained within a secure boundary 140. The cryptographic facility 30 includes the instruction processor 142 which is coupled to the cryptographic algorithms 144 which are embodied as executable code. The cryptographic facility environment memory 146 is coupled to the instruction processor 142. The physical interface can be coupled over line 37 to the CF environment memory 146, as shown in the figure. The instruction processor 142 is coupled to the cryptographic facility access program (CFAP) 34 by means of the interface at 31.

The instruction processor 142 is a functional element which executes cryptographic microinstructions invoked by the CFAP access instruction at the interface 31. For each access instruction, the interface 31 first defines an instruction mnemonic or operation code used to select particular microinstructions for execution. Secondly a set of input parameters is passed from the CFAP 34 to the CF 30. Thirdly, a set of output parameters is returned by the CF 30 to the CFAP 34. The instruction processor 142 executes the selected instruction by performing an instruction specific sequence of cryptographic processing steps embodied as microinstructions stored in cryptographic microinstruction memory 144. The control flow and subsequent output of the cryptographic processing steps depend on the values of the input parameters and the contents of the CF environment memory 146. The CF environment memory 146 consists of a set of cryptographic variables, for example keys, flags, counters, CF configuration data, etc., which are collectively stored within the CF 30. The CF environment variables in memory 146 are initialized via the interface 31, that is by execution of certain CF microinstructions which read input parameters and load them into the CF environment memory 146. Alternately, initialization can be done via an optional physical interface which permits cryptographic variables to be loaded directly into the CF environment memory 146, for example via an attached key entry device.

The physical embodiment of the cryptographic facility secure boundary 140, incorporates the following physical security features. The physical embodiment resists probing by an insider adversary who has limited access to the cryptographic facility 30. The term "limited" is measured in minutes or hours as opposed to days or weeks. The adversary is constrained to a probing attack at the customer's site using limited electronic devices as opposed to a laboratory attack launched at a site under the control of the adversary using sophisticated electronic and mechanical equipment. The physical embodiment also detects attempts at physical probing or intruding, through the use of a variety of electro-mechanical sensing devices. Also, the physical embodiment of the cryptographic facility 30 provides for the zeroization of all internally

stored secret cryptographic variables. Such zeroization is done automatically whenever an attempted probing or intrusion has been detected. The physical embodiment also provides a manual facility for a zeroization of internally stored secret cryptographic variables. Reference to the Abraham, et al. patent application cited above, will give an example of how such physical security features can be implemented.

Initialization of Public-Key Cryptographic System: Fig. 4 illustrates two cryptographic systems, A and B, that wish to communicate cryptographically using public key cryptography. Cryptographic system A generates a public and private key pair (PUa, PRa), where PUa is the public key of A and PRa is the private key of A. In like manner, cryptographic system B generates a public and private key pair (PUb, PRb), where PUb is the public key of B and PRb is the private key of B.

Referring to Fig. 4, the cryptographic facility 30 of cryptographic system A contains a master key KMa and the cryptographic facility 30' of cryptographic system B contains a master key KMb. KMa and KMb are ordinarily different, being equal only by mere chance. At cryptographic system A, the public key PUa is encrypted with the Data Encryption algorithm (DEA) using variant key KMa.C1 to form the encrypted value eKMa.C1-(PUa), where KMa.C1 is formed as the Exclusive OR product of master key KMa and control vector C1. Likewise, at cryptographic system A, the private key PRa is encrypted with the DEA using variant key KMa.C2 to form the encrypted value eKMa.C2(PRa), where KMa.C2 is formed as the Exclusive OR product of master key KMa and control vector C2. The symbol "." denotes the Exclusive OR operation. The encrypted values eKMa.C1(PUa) and eKMa.C2(PRa) are stored in cryptographic key data set 32.

The control vector specifies whether the key is a public or private key and contains other key usage control information specifying how the key may be used. For example, when the encrypted key eKMa.C2(PRa) is decrypted for use within the cryptographic facility 30, control vector C2 indicates to the cryptographic facility how and in what way the key PRa may be used. Control vector C1 similarly controls the use of public key PUa. The use of the control vector to control key usage is described in U.S. Patents 4,850,017, 4,941,176, 4,918,176, 4,924,514, 4924,515, and 5,007,089 cited in the background art and in co-pending patent application serial number 07/602,989 also cited in the background art. Fig. 11 illustrates control vectors that define public and private keys, where the public and private keys are key management keys used by the cryptographic system to distribute DEA keys. The fields in each control vector consist of a CV TYPE, which specifies whether the control vector is a public or a private key and additionally whether the key pair is a key management key pair for use in distributing DEA keys or whether the key pair is some other kind of key pair. Other types of key pairs are possible, such as user keys which can be used for generation and verification of digital signatures but not for key distribution. Each control vector has a PR USAGE and PU USAGE field. For the public key control vector, the PU USAGE field controls the usage of the public key in cryptographic instructions whereas the PR USAGE field is only informational. For the private key control vector, the PR USAGE field controls the usage of the private key in cryptographic instructions whereas the PU USAGE field is only informational. The ALGORITHM field indicates the public key algorithm to which this key pair pertains. The HIST field records history information, e.g., the options used to import a public key (see co-pending patent application serial number 07/602,989 as cited in the background art, which describes the use of history information fields in the public key control vector). The reader will appreciate that the control vector may contain a variety of different control vector fields for the purpose of controlling the operation and use of the key within the cryptographic network and cryptographic systems within the network.

In an alternate embodiment, the public key PUa may be stored in an unencrypted form, since there is no intent to keep the value of this key secret. Encrypting PUa is done for sake of uniformity, so that all keys in the cryptographic key data set 32 are stored and recovered using one common method. Those skilled in the art will also recognize that the length of PUa and PRa will likely be different than the block size of the DEA, which is 64 bits, and hence PUa and PRa may need to be encrypted in separate 64-bit pieces. The particular method for encrypting PUa and PRa is unimportant to the invention. However, one way that this encryption can be carried out is to use the Cipher Block Chaining (CBC) mode of DEA encryption described in DES modes of operation, Federal Information Processing Standards Publication 81, National Bureau of Standards, US Department of Commerce, December 1980. In cases where KMa is a 128-bit key, the CBC mode of DEA encryption can be adapted to encrypt PUa under KMa. PUa is first encrypted with the leftmost 64 bits of KMa, then decrypted with the rightmost 64 bits of KMa, and then encrypted again with the leftmost 64 bits of KMa.

In like manner, at cryptographic system B, the public and private keys, PUb and PRb, are encrypted with master key KMb and control vectors C3 and C4, per the same method described for cryp-

tographic system A. The encrypted values eKMb.C3(PUb) and eKMb.C4(PRb) are stored in cryptographic key data set 32'. Control vectors C3 and C4 control the usage of PUb and PRb, respectively.

Although encryption of the public and private keys has been described in terms of a DEA-based master key, those skilled in the art will appreciate that the DEA could be replaced by a public key algorithm and the master keys could be replaced by a PKA-based key pair used for this purpose. Moreover, the encryption of the public and private keys has been described in terms of encryption of the keys only. In some implementations it may be more practical to imbed these keys within key records that contain other key-related information besides the keys themselves.

In order for cryptographic systems A and B to carry out cryptographic operations using their respective implemented public key algorithms, they must share their public keys with each other. Thus, at cryptographic system A a function exists that permits the encrypted value eKMa.C1(PUa) to be accessed from cryptographic key data set 32 and decrypted so that the clear value of PUa may be exported to cryptographic system B at 300. At cryptographic system B a function exists that permits the clear value of PUa to be imported and encrypted under the variant key KMb.C1. The so-imported encrypted value eKMb.C1(PUa) is then stored in cryptographic key data set 32'. In like manner, functions exist at B and A that permit public key PUb to be decrypted at B, sent to A at 301, and re-encrypted at A for storage in A's cryptographic key data set.

Co-pending patent application by S. M. Matyas et al., serial number 07/602,989, "Method and Apparatus for Controlling the use of a Public Key, Based on the Level of Import Integrity for the Key," describes a method for generating public and private keys and for distributing public keys in order to initialize a public-key cryptographic system, and is incorporated by reference herein.

Key Distribution: Fig. 5 illustrates the process by which cryptographic system A may distribute a key to cryptographic system B using a public key algorithm (PKA). That is, it illustrates the process of key distribution using a PKA. In a hybrid key distribution scheme, the distributed key is a DEA key, e.g., an initial key-encrypting key to be used later with a DEA-based key distribution scheme to distribute all subsequent DEA keys. However, any key can be distributed using the so-described PKA-based key distribution scheme, including both DEA keys and PKA keys. The distributed DEA and PKA keys can be of any type or designated use. However, for purposes of illustration, Fig. 5 shall assume that the distributed key is a DEA key.

Referring to Fig. 5, the steps involved in distribution of a key from cryptographic system A to cryptographic system B are these. At cryptographic system A, a Generate Key Set PKA (GKSP) instruction is executed within the CF 30. Control information at 303 is provided to the GKSP instruction as input. In response, the GKSP instruction generates a key K and produces two encrypted copies of K, which are returned by the GKSP instruction at 305 and 306. The first encrypted copy of K is produced by encrypting K with the DEA using variant key KMa.C5 formed as the Exclusive OR product of master key KMa and control vector C5. C5 may be input to the GKSP instruction as part of the control information, at 303, or it may be produced within the CF 30 as part of the GKSP instruction, or it may be produced as a combination of both methods. The second encrypted copy of K is produced as follows. A key block (designated keyblk) is first formed. The key block includes the clear value of K, control information, and possibly other information unimportant to the present discussion, as illustrated in Fig. 6. The format of the keyblk is unimportant to the present discussion, and those skilled in the art will recognize that many possible arrangements of the keyblk information are possible. In all cases, the keyblk contains the necessary information to accomplish the task of key distribution. The length of the keyblk is assumed to be equal to the block size of the public key algorithm. For example, if the public key algorithm is the RSA algorithm, then the block size is just the modulus length. Also, it is assumed that the numeric value of the keyblk, say its binary value, is adjusted as necessary to permit it to be encrypted as a single block by the public key algorithm. For example, if the public key algorithm is the RSA algorithm, then the keyblk is adjusted so that its binary value is less than the binary value of the modulus. This can be done by forcing the high order (most significant) bit in the keyblk to zero. Once the keyblk has been formatted, it is encrypted with the public key PUb of cryptographic system B to form the encrypted value ePUb(keyblk), which is returned at 306. To permit this to be accomplished, the encrypted value eKMa.C3(PUb) and control vector C3 are supplied to the GKSP instruction at 304 as inputs and eKMa.C3(PUb) is decrypted under variant key KMa.C3. KMa.C3 is formed as the Exclusive OR product of master key KMa stored within the CF 30 and control vector C3.

The first encrypted output eKMa.C5(K) at 305 is stored in the cryptographic key data set 22 of cryptographic system A. Control vector C5 is also stored in the cryptographic key data set 22 together with the encrypted key eKMa.C5(K). In some implementations it may be convenient to

store eKMa.C5(K) and C5 in an internal key token together with other key-related information. The internal key token is not relevant to the present discussion, and is therefore not shown in Fig. 5. If C5 is generated within the CF 30, it may also be provided as an output at 305 so that it may be store in CKDS 22.

The second encrypted output ePUb(keyblk) at 306 is formatted within an external key token 308. The external key token contains the encrypted key or encrypted key block ePUb(keyblk), control information, and other information unimportant to the present discussion, as shown in Fig. 7. The control information supplied as input to the GKSP instruction at 303 is also stored in external key token 308 at 307. However, the control information at 307 may include additional information available to the cryptographic facility access program (CFAP) which is not specified as an input to the GKSP instruction at 303. In other words, the source of the control information in the external key token 308 may be much broader than the control information supplied as input the the GKSP instruction at 303. One example, is the Environment Identifier (EID) value stored both in the CF 30 and in the CFAP. The EID value is an identifier that uniquely identifies each cryptographic facility or cryptographic system within a network. The EID value is loaded into the CF 30 during an initialization sequence prior to performing routine cryptographic operations within the cryptographic system. Another example of initialization is the loading of the master key KMa. The EID value need not be supplied to the CF since it is already stored in the CF. But the EID value may be stored within the external key token, in which case it is supplied as an input at 307. In like manner, the control information in the keyblk may include a control vector C6 specifying the usage of K at cryptographic system B. In that case, C6 may be supplied as part of the control information at 303, in which case it is also supplied as part of the control information at 307. If however C6 is generated within CF 30, then C6 is not supplied as part of the control information at 303, but is supplied as part of the control information at 307. Those skilled in the art will recognize that various alternatives exist for the specification or derivation of the necessary control information and that different combinations of inputs to the GKSP instruction and to the external key token are therefore possible.

The formatted external key token 308 is transmitted to cryptographic system B where it is processed. The CFAP at B first checks the control information in the external key token for consistency. For example, if the control information contains a control vector C6, then C6 is checked to ensure that it represents a key type and key usage approved by cryptographic system B. Likewise, if the control information contains an EID value, then the EID value is checked to ensure that the external key token and the key to be imported originated from cryptographic system A, i.e., it originated from the expected or anticipated cryptographic system that B 'thinks' it is in communication with and which it desires to establish a keying relationship. Once this has been accomplished, the received key is imported as follows. The encrypted keyblk, ePUb(keyblk) and part or all of the control information in the external key token are supplied as inputs to an Import DEA Key (IDK) instruction at 309, which is executed within CF 30' at cryptographic system B. In response, the IDK instruction decrypts ePUb(keyblk) under the private key PRb belonging to cryptographic system B. To permit this to be accomplished, the encrypted value eKMb.C4(PRb) and control vector C4 are supplied to the IDK instruction at 310 as inputs and eKMb.C4(PRb) is decrypted under variant key KMb.C4. KMb.C4 is formed as the Exclusive OR product of master key KMb stored within the CF 30' and control vector C4. Once ePUb(keyblk) has been decrypted and the clear value of keyblk has been recovered, the keyblk is processed as follows. The control information contained in the keyblk is checked for consistency against the control information, or reference control information, supplied as input at 309. If the consistency checking is satisfactory (okay), then the clear value of K is extracted from keyblk and it is encrypted with the variant key KMb.C6 to produce the encrypted key value eKMb.C6(K). KMb.C6 is formed as the Exclusive OR product of master key KMb stored within CF 30' and control vector C6. Control vector C6 may be obtained in different ways. C6 may be contained in the control information in keyblk, in which case it is extracted from keyblk. In other cases, C6 may be produced within CF 30'. For example, if there is only one key type and key usage permitted, then C6 can be a constant stored within the IDK instruction. The so-produced encrypted key value eKMb.C6(K) is provided as an output of the IDK instruction at 311, and is stored together with its control vector C6 within CKDS 22'. The value of C6 stored in CKDS 22' is obtained either from the control information input to the IDK instruction at 309 or, if C6 is not in the control information input to the IDK instruction at 309, then it is produced by the CFAP in the same way that it is produced by the IDK instruction and stored in CKDS 22'. Alternatively, C6 could be returned as an output of the IDK instruction. Those skilled in the art will realize that several alternatives exist for obtaining C6 depending on how and where it is produced within the cryptographic system and whether it is or is not included as part of the

control information in the external key token.

In the preferred embodiment, the control information at 303 supplied to the GKSP instruction includes a specification of control vectors C5 and C6. This allows the GKSP instruction the freedom and flexibility to generate two encrypted copies of key K that have different key types and usages, as specified by C5 and C6. In that case, the GKSP instruction must incorporate some control vector checking to determine that C5 and C6 constitutes a valid pair. The various options for control vector design and checking pursued here are based on the control vector designs included in prior art, cited in the background art, and already discussed. Likewise, in the preferred embodiment, control vector C6 is included in the control information in the key block (keyblk) and also in the control information in the external key token. This permits the receiving cryptographic system to import keys of different types while still permitting the receiving system to verify that the imported key is one that it wants or expects. This is accomplished by the CFAP first checking the control vector in the external key token to make sure that it prescribes a key type and key usage that it expects or will allow to be imported. C6 is then supplied as an input in the control information at 309 to the CF 30'. At the time the IDK instruction recovers the clear value of keyblk, the value of C6 in the control information in keyblk is checked against the value of C6, or the reference value of C6, supplied as input. This permits the CF to verify that the value of C6 used to import the key K is the same control vector C6 in the external key token. Otherwise, if this check was ignored it would be possible for an adversary to substitute C6' for C6 in the external key token, causing a key to be imported that the CFAP may not permit.

In the preferred embodiment, each cryptographic facility stores a unique EID value, e.g., a 128-bit value set within the CF during an initialization sequence before routine operations are permitted. At the time a keyblk is prepared within the CF by a GKSP instruction, the EID value is obtained from the CF and included within the control information in the keyblk. In like manner, a duplicate copy of the EID value is stored outside the CF with integrity such that it is available to the CFAP. This EID value is obtained by the CFAP and is included within the control information in the external key token. Thus, the CFAP at the receiving cryptographic system can check the EID value in the control information of the received external key token to ensure that the external key token originated from the cryptographic system that is expected or anticipated. That is, B knows that the external key token came from A, which is what is expected. The EID value is also supplied as part of the control information at 309. Thus, when the IDK instruction obtains the clear keyblk, the EID value in the control information in the clear keyblk can be checked against the EID value, or reference EID value, supplied as an input. In this way, the CFAP is sure that the IDK instruction will import K only if the two EID values are equal. This prevents an adversary from changing the EID value in the external key token to a different value that might also be accepted by the receiving device. This might lead to a situation where B imports a key from A, thinking that it came from C.

The EID also serves another purpose, as now described. At the time the clear value of keyblk is obtained by the IDK instruction, a check is performed to ensure that the value of EID in the control information in keyblk is not equal to the value of EID stored in the CF at the receiving device. Thus, the encrypted value of ePUx(keyblk) produced at cryptographic system A, where PUx may be the public key of any cryptographic system in the network, including A itself, cannot be imported by A. This prevents an adversary at A, who specifies his own public key PUa to the GKSP instruction, from importing ePUa(keyblk) at A and thereby obtaining two encrypted copies eKMa.C5-(K) and eKMa.C6(K) of the same key with potentially different key types and key usage attributes. In some cases, bi-functional key pairs are undesirable and the key management design will specifically disallow such key pairs to be created using the key generation facilities provided by the key management services.

Key Distribution with Digital Signatures: The key distribution scheme described in Fig. 5 is not by itself the preferred embodiment of the invention. This is so because, as it stands, the scheme can be attacked by an adversary who knows the public value of B's key, PUb. In public key cryptographic systems, one naturally makes the assumption that PUb is known by anyone, even an adversary. The adversary can forge values of keyblk containing DEA keys of his choosing and freely encrypt these key blocks under PUb. Thus, at B, there is no way to know that an imported key originated with A or with an adversary posing as A. The importing function will import the forged values of keyblk, which results in known values of K being encrypted under the master key, of the form eKMb.C6(K), and stored in CKDS 22'. In that case, data or keys encrypted under K are easily deciphered by the adversary who knows K.

The preferred embodiment of the invention therefore includes a means by which the receiver, say cryptographic system B, can ensure that a received encrypted keyblk of the form ePUb-(keyblk) did in fact originate with the intended sender, say cryptographic system A. To accom-

plish this, the GKSP instruction at cryptographic system A produces a digital signature (designated DSIGa) on ePUb(keyblk) using its private key PRa. The so-produced digital signature is transmitted together with the external key token to cryptographic system B where the key is imported using an IDK instruction. In this case, the IDK instruction first verifies the digital signature DSIGa using the previously imported copy of PUa received from cryptographic system A. Only after DSIGa has been successfully verified will the IDK instruction continues as already described in Fig. 5 and import the key K.

Fig. 8 illustrates the scheme for DEA key distribution with digital signatures, which is the same as the scheme shown in Fig. 5 except as follows. Once the encrypted key value ePUb(keyblk) has been produced, the GKSP instruction additionally produces the digital signature DSIGa from ePUb(keyblk) and the private key PRa belonging to cryptographic system A. A common method for producing such a signature is to first calculate a hash value on ePUb(keyblk) using a one way cryptographic function, such as described in U. S. patent 4,908,861 by Brachtl et al., cited in the background art, which uses either two DEA encryptions or four DEA encryptions per each 64 bits of input text to be hashed, and then decrypt (or transform) the hash value using the private key PRa to produce a DSIGa of the form dPRa(hash value). The clear value of PRa is obtained by decrypting the encrypted value of eKMa.C2(PRa) supplied as an input to the GKSP instruction at 313 using the DEA and the variant key KMa.C2. KMa.C2 is formed as the Exclusive OR product of master key KMa stored in CF 30 and control vector C2 supplied as input to the GKSP instruction at 313. For example, if the public key algorithm is the RSA algorithm, a the digital signature may be calculated using the method as described in ISO Draft International Standard 9796 entitled "Information Technology -- Security Techniques -- Digital Signature Scheme Giving Message Recovery." The so-produced DSIGa 315 is returned as an output at 314. Both the external key token 308 and the DSIGa 315 are transmitted to cryptographic system B. At cryptographic system B, the IDK instruction is used to import the key K in similar fashion as described in Fig. 5 except that the IDK instruction first validates DSIGa using the public key PUa previously imported, encrypted, and stored in CKDS 22'. A DSIGa of the form dPRa(hash value) is validated by encrypting dPRa(hash value) with PUa, calculating a hash value on ePUb(keyblk) using the same one way cryptographic function, called the hash value of reference, and comparing the hash value of reference and the recovered clear hash value for equality. Only if this comparison check is successful does the IDK instruction continue and import the key K. The clear value of PUa is obtained by decrypting the encrypted value of eKMb.C1(PUa) supplied as an input to the IDK instruction at 316 using the DEA and the variant key KMb.C1. KMb.C1 is formed as the Exclusive OR product of master key KMb stored in CF 30' and control vector C1 supplied as input to the IDK instruction at 316. Thus, the GKSP instruction at cryptographic system A produces DSIGa and the IDK instruction at cryptographic system B verifies DSIGa. In an alternate embodiment, DSIGa can be calculated by the CF 30 using a separate instruction for generating digital signatures. In that case, after the GKSP instruction has been executed, the CFAP invokes the generate digital signature instruction causing DSIGa to be generated. In like manner, DSIGa can be verified by the CF 30' using a separate instruction for verifying digital signatures. In that case, before the IDK instruction is invoked, the CFAP invokes the verify digital signature instruction to ensure that DSIGa is valid.

Generate Key Set PKA (GKSP) Instruction: Fig. 9 illustrates the Generate Key Set PKA (GKSP) instruction. The GKSP instruction of Fig. 9 is identical to the GKSP instruction contained within the CF 30 of Fig. 8. The GKSP instruction generates a two encrypted copies of a generated DEA key K. The first copy is of the form eKM.C5(K) and is stored in the cryptographic key data set of the generating cryptographic device, say A. The second copy is of the form ePU(keyblk) and is transmitted to a designated receiving cryptographic device, say B, where the public key PU belonging to the receiving cryptographic device B. Also, the GKSP instruction produces a digital signature DSIG on ePU(keyblk) using the private key PR of the generating cryptographic device A. DSIG is also transmitted to cryptographic device B to serve as proof that ePU(keyblk) was produced at cryptographic device A, i.e., produce at a valid network cryptographic device.

Referring to Fig. 9, GKSP instruction 500 consists of control information retrieval means 504, PU recovery means 506, PR recovery means 507, key generation means 508, eKM.C5(K) production means 509, ePU(keyblk) production means 510, DSIG production means 511, and hash algorithms 512. GKSP instruction 500 is located in instruction processor 142 within cryptographic facility 30, as shown in Fig. 3. The inputs to the GKSP instruction are supplied to the GKSP instruction by CFAP 34, i.e., by the CFAP 34 to the CF 30 across the CFAP-to-CF interface. In similar manner, the outputs from the GKSP instruction are supplied to the CFAP 34, i.e., by the CF 30 to the CFAP 34 across the CFAP-to-CF interface.

The inputs to GKSP instruction 500 are (1) at 501, control information such as control vectors C5 and C6 that specify the key usage attributes of the two encrypted copies of the generated DEA key K, (2) at 502, control vector C3 and encrypted public key eKM.C3(PU), where C3 specifies the key usage attributes of public key PU belonging to the receiving cryptographic device, and (3) at 503, control vector C2 and encrypted private key eKM.C2(PR), where C2 specifies the key usage attributes of private key PR belonging to the sending or generating cryptographic device. The outputs from GKSP instruction 500 are (1) at 521, the encrypted key, eKM.C5(K), where K is encrypted under variant key KM.C5 formed as the Exclusive OR product of master key KM and control vector C5, (2) at 522, the encrypted key block, ePU-(keyblk), where keyblk is encrypted under public key PU belonging to the intended receiving cryptographic device and where keyblk is a key block containing the generated DEA key K, control information, and possibly other information as depicted in Fig. 6, and (3), at 523, a digital signature DSIG generated on ePU(keyblk) using the private key PR belonging to the sending or generating cryptographic device.

Control information retrieval means 504 accepts and parses control information supplied as input to the GKSP instruction at 501. Also, control information retrieval means 504 accesses control information stored within the secure boundary of the cryptographic facility, e.g., the Environment Identifier (EID) at 505. Control information retrieval means 504 may also perform consistency checking on the assembled control information. For example, control vectors C5 and C6 may be checked and cross checked for consistency, i.e., to ensure they are a valid control vector pair. GKSP instruction 500 is aborted if C5 and C6 are incorrect or do not specify the correct key usage required by the GKSP instruction. In an alternate embodiment, it may be possible for control information retrieval means 504 to generate or produce control vector C6 from control vector C5, or vice versa, in which case only one control vector is specified in the control information supplied at 501. In that case, cross checking of C5 and C6 is unnecessary. Control vector checking of C5 or C6 can be performed in control information retrieval means 504 or in eKM.C5(K) production means 509 if the control vector is C5 or in ePU(keyblk) production means 510 if the control vector is C6. The reader will appreciate that control vector checking may be accomplished in variety of ways within the different components parts of the GKSP instruction, and that these variations do not significantly depart of the general framework of the invention. In any event, control information retrieval means 504 makes the control information available to other component parts of the GKSP instruction. C5 is passed to eKM.C5(K) production means 509 and EID and C6 are passed to ePU(keyblk) production means 510. Optionally, control information may also be passed to DSIG production means 511 such as the identifier or name of a hashing algorithm to be used in the preparation of the digital signature. The GKSP instruction may support only one hashing algorithm in which case the identifier or name of a hashing algorithm need not be passed to DSIG production means. Those skilled in the art will recognize that many possible variations exist for inputting and accessing control information, for parsing, checking and making the control information available to different component parts of the GKSP instruction.

PU recovery means 506 decrypts input eKM.C3(PU) under variant key KM.C3 formed as the Exclusive OR product of master key KM stored in clear form within the cryptographic facility and directly accessible to GKSP instruction 500 and control vector C3 specified as an input to GKSP instruction 500. Prior to decrypting eKM.C3(PU), PU recovery means 506 performs control vector checking on C3. GKSP instruction 500 is aborted if C3 is incorrect or does not specify the correct key usage required by the GKSP instruction. Public key PU is stored in encrypted form so that PU Recovery means 506 will be, for all practical purposes, identical to PR Recovery means 507. Encryption of PU is also preferred since it permits control vector C3 to be cryptographically coupled with public key PU. Even though PU is public, and there is no need to protect the secrecy of PU, encryption of PU thus ensures that PU can be used only if C3 is correctly specified as an input to the GKSP instruction. This ensures that PU is used by the GKSP instruction only if it has been so designated for use. In an alternate embodiment, PU could be stored outside the cryptographic facility in clear form and PU Recovery means 506 could be omitted from GKSP instruction 500. In this case, the embodiment may choose to fix the usage attributes of PU so that there is no chance for an adversary to specify a control vector C3 that is incorrect, i.e., C3 is a fixed constant value. In any event, the recovered clear value of PU is supplied as an input to ePU(keyblk) production means 510.

PR recovery means 507 decrypts input eKM.C2(PR) under variant key KM.C2 formed as the Exclusive OR product of master key KM stored in clear form within the cryptographic facility and directly accessible to GKSP instruction 500. Prior to decrypting eKM.C2(PR), PR recovery means 507 performs control vector checking on C2. GKSP instruction 500 is aborted if C2 is incorrect or does not specify the correct key usage required by the GKSP instruction. The recovered clear value of PR

is supplied as an input to DSIG production means 511.

Key generator means 508 is a pseudo random number generator for generating DEA keys. Alternatively, key generator means 508 could be a true random number generator. For sake of simplicity, key generator means 508 generates 64-bit random numbers which are adjusted for odd parity. That is, the eight bit of each byte in the generated random number is adjusted so that the value in each byte is odd. DEA keys may contain either 64 or 128 bits depending on their intended usage. Data-encrypting-keys used for encrypting data are 64-bit keys. Key-encrypting-keys used for encrypting keys are generally 128-bit keys, but may in some cases be 64-bit keys. To produce a 128-bit key, key generator means 508 is invoked twice. The so-generated DEA key is supplied as an input to both eKM.C5(K) production means 509 and ePU(keyblk) production means 510.

eKM.C5(K) production means 509 Exclusive ORs input KM and C5 to produce variant key KM.C5 and then encrypts input K with KM.C5 to form the encrypted output eKM.C5(K), which is returned to the CFAP at 521. If control vector C5 is not consistency checked in control information retrieval means, it may alternatively be checked here.

ePU(keyblk) production means 510 first prepares a key block, designated keyblk, from the inputs K, EID, and C6, and then encrypts keyblk with public key PU to form the encrypted output ePU(keyblk), which is returned to the CFAP at 522. If control vector C6 is not consistency checked in control information retrieval means, it may alternatively be checked here. The value ePU(keyblk) is also supplied as an input to DSIG production means 511 to allow the digital signature DSIG to be produced. The format of keyblk is shown in Fig. 6 and has been discussed previously. The procedure of preparing keyblk accomplishes two main goals. It ensures that all necessary information such as the key, control information, key-related information, keyblk parsing information, etc. is included within keyblk. Also, it ensure that keyblk is constructed in a way that keyblk can be encrypted with PU using the public key algorithm. For example, it may be necessary to pad keyblk so that its length and binary value are such that keyblk is encrypted properly and in conformance with restrictions imposed or that may be imposed by the public key algorithm.

DSIG production means 511 produces a digital signature on ePU(keyblk) using private key PR. To accomplish this, a hash value is first calculated on ePU(keyblk) using hash algorithm 512. Hash algorithm 512 may in fact be a set of hash algorithms. In that case, the hash algorithm is selected on the basis of a hash algorithm identifier or other appropriate encoded value passed by the control information retrieval means 504 to the DSIG production means 511. The so-produced hash value is then formatted in a suitable signature block and decrypted with private key PR to produce DSIG, which is returned to the CFAP at 523. The signature block can in the simplest case consist of the hash value and padding data, so as to construct a signature block whose length and value are in conformance with restrictions imposed or that may be imposed by the public key algorithm, as already discussed above. The DSIG production means 511 may also implement a digital signature method based on a national or international standard, such as International Standards Organization draft international standard (ISO DIS) 9796.

Import DEA Key (IDK) Instruction: Fig. 10 illustrates the Import DEA Key (IDK) instruction. The IDK instruction of Fig. 10 is identical to the IDK instruction contained within the CF 30 of Fig. 9 The IDK instruction permits a cryptographic device, say B, to import an encrypted DEA key of the form ePU(keyblk) that has been received from a sending cryptographic device, say A. The received digital signature DSIG is used by the IDK instruction to verify that ePU(keyblk) originated with cryptographic device A, i.e., at a valid network cryptographic device.

Referring to Fig. 10, IDK instruction 600 consists of PU recovery means 606, PR recovery means 607, control information retrieval means 608, hash algorithms 610, DSIG verification means 611, keyblk recovery means 612, eKM.C6(K) production means 613, and control information consistency checking means 614. IDK instruction 600 is located in instruction processor 142 within cryptographic facility 30, as shown in Fig. 3. The inputs to the IDK instruction are supplied to the IDK instruction by CFAP 34, i.e., by the CFAP 34 to the CF 30 across the CFAP-to-CF interface. In similar manner, the outputs from the IDK instruction are supplied to the CFAP 34, i.e., by the CF 30 to the CFAP 34 across the CFAP-to-CF interface.

The inputs to the IDK instruction 600 are (1) at 601, control vector C1 and encrypted public key eKM.C1(PU), where C1 specifies the key usage attributes of public key PU belonging to the sending cryptographic device, (2) at 602, digital signature DSIG, (3) at 603, encrypted key block ePU(keyblk), where keyblk is encrypted under public key PU belonging to the the receiving cryptographic device and where keyblk is a key block containing the to-be-imported DEA key K, control information, and possibly other information as depicted in Fig. 6, (4) at 604, control vector C4 and encrypted private key eKM.C4(PR), where C4 specifies the key usage attributes of private key PR belonging to the receiving cryptographic device, and (5) at 605,

control information, such as a reference control vector C6 and a reference EID value of the sending cryptographic device. The output of the IDK instruction 600 is the encrypted key eKM.C6(K), where K is the to-be-imported DEA key encrypted under variant key KM.C6 formed as the Exclusive OR product of master key KM and control vector C6.

PU recovery means 606 decrypts input eKM.C1(PU) under variant key KM.C1 formed as the Exclusive OR product of master key KM stored in clear form within the cryptographic facility and directly accessible to IDK instruction 600 and control vector C1 specified as an input to IDK instruction 600. Prior to decrypting eKM.C1(PU), PU recovery means 606 performs control vector checking on C1. IDK instruction 600 is aborted if C1 is incorrect or does not specify the correct key usage required by the IKK instruction. Public key PU is stored in encrypted form so that PU recovery means 606 well be, for all practical purposes, identical to PR recovery means 607 Encryption of PU is also preferred since it permits control vector C1 to be cryptographically coupled with public key PU, as argued previously under the description of the GKSP instruction. In an alternate embodiment, PU could be stored outside the cryptographic facility in clear form and PU recovery means 606 could be omitted from IDK instruction 600. In this case, the embodiment may choose to fix the usage attributes of PU so that there is no chance for an adversary to specify a control vector C1 that is incorrect, i.e., C1 is a fixed constant value. In any event, the recovered clear value of PU is supplied as an input to DISG verification means 611.

PR recovery means 607 decrypts input eKM.C4(PR) under variant key KM.C4 formed as the Exclusive OR product of master key KM stored in clear form within the cryptographic facility and directly accessible to IDK instruction 600. Prior to decrypting eKM.C4(PR), PR recovery means 607 performs control vector checking on C4. IDK instruction 600 is aborted if C4 is incorrect or does not specify the correct key usage required by the IDK instruction. The recovered clear value of PR is supplied as an input to keyblk recovery means 612.

Control information retrieval means 608 accepts and parses control information supplied as input to the IDK instruction at 605. Also, control information retrieval means 608 accesses control information stored within the secure boundary of the cryptographic facility, e.g., the Environment Identifier (EID) at 609. Control information retrieval means 608 supplies control information to control information consistency checking means 614 and possibly to other component parts of the IDK instruction, such as a hash algorithm identifier supplied to DSIG verification means 611 (not shown in Fig. 10).

DSIG verification means 611 uses public key PU belonging to the sending cryptographic device to verify the digital signature DSIG generated on ePU(keyblk) at the sending cryptographic device. To accomplish this, a hash value is first calculated on ePU(keyblk) using hash algorithm 512. Hash algorithm 512 may in fact be a set of hash algorithms. In that case, the hash algorithm is selected on the basis of a hash algorithm identifier or other appropriate encoded value passed by the control information retrieval means 608 to the DSIG verification means (not shown in Fig. 10). The clear public key PU obtained from PU recovery means 606 is then used to encrypt the value of DSIG specified as an input at 602. This recovers the original signature block in clear form, which is then parsed to recover the original hash value. The recovered hash value and the calculated hash value are then compared for equality. If this comparison is favorable, then DSIG is considered valid; otherwise, DSIG is not considered valid and IDK instruction 600 is aborted. The signature block recovery and processing of course will depend on the method of digital signature implemented. In the description of the GKSP instruction it was indicated that the signature block may consist of the hash value and padding data or it may be constructed on the basis of a national or international standard, such as International Standards Organization draft international standard (ISO DIS) 9796. Those skilled in the art will appreciate that many possible implementations of the digital signature are possible and that the precise method of digital signatures is unimportant to the invention. What is important is that a method of digital signature is used in the preferred embodiment to ensure that the receiving cryptographic device can authenticate that the to-be-imported DEA key did in fact originate from a valid network cryptographic device. As the reader will also see, the digital signature is made an integral part of the GKSP and IDK instructions themselves, which ensures that the process of signature production and signature verification occurs as part of the key export and key import processes and therefore the highest possible integrity over these processes is achieved. Although it is possible to perform signature production and signature verification as separate instructions, which achieves complete compatibility with the present descriptions of the GKSP and IDK instructions, one also sees that less integrity is achieved. This is so because the signature generate instruction has no way to ensure that a key of the form ePU(keyblk) was in fact produced by the GKSP instruction.

Keyblk recovery means 612 decrypts input ePU(keyblk), provided as an input to the IDK in-

struction at 603, under private key PR, provided as an output of PR recovery means 607. The recovered clear key block, keyblk, is provided as an output to both eKM.C6(K) production means 613 and control information consistency checking means 614.

Control information consistency checking means 614 checks the control information in the recovered keyblk output from keyblk recovery means 612 and the reference control information output from control information retrieval means 608 for consistency. A first check consists in checking control vector C6 in keyblk for consistency with the reference control vector C6 supplied as an input to the IDK instruction at 605. This ensures that the receiving cryptographic application imports a key from with the expected or intended key usage attributes. In this case, reference control vector C6 represents the expected control vector, whereas the recovered control vector C6 represents the actual control vector. The simplest form of consistency checking consists of checking these two control vectors for equality. However, a more refined procedure is possible wherein attributes in the reference control vector are allowed to override corresponding attributes in the recovered control vector. For example, the reference control vector could disable the ability to re-export the imported DEA key K, whereas the recovered control vector may or may not permit the imported DEA key K to be re-exported. More generally, the receiving device may disable any attribute granted within the received control vector. One will appreciate that taking away a right is not the same as granting a right, which only the sending cryptographic device is permitted to do. The IDK instruction can be designed to permit this kind of control vector override or it may not, depending on the desires of the designer of the IDK instruction. A second check consists of checking the EID value in keyblk for equality with the reference EID value supplied as an input to the IDK instruction at 605. This ensures that the receiving cryptographic application imports a key from the expected or intended sending cryptographic device. In this case, the reference EID value is the EID of the intended sending cryptographic device, which is checked against the EID value in keyblk which represents the EID value of the actual sending cryptographic device. A third check consists of checking the EID value in keyblk for inequality with the EID value stored in the cryptographic facility of the receiving device. This ensures that the imported DEA K originated at another cryptographic device, i.e., that A can't import a K produced at A, that B can't import a K produced at B, etc. The usefulness of this check has been discussed previously. In all cases, if the consistency checking fails, then the IDK instruction is aborted.

eKM.C6(K) production means 613 extracts the clear value of DEA key K and the control vector C6 from keyblk, obtained as an output from the keyblk recovery means 612. and K is then encrypted under variant key KM.C6 formed as the Exclusive OR product of master key KM and control vector C6 recovered from C6 to produce the encrypted key value eKM.C6(K). In an alternative embodiment where the reference control vector C6 can override the recovered control vector C6, the value of C6 used to form the variant key KM.C6 can be the reference control vector C6. In yet another alternative embodiment the IDK instruction itself can modify information in the control vector C6, so that K is encrypted with variant key KM.C6', where C6' is the IDK modified value of C6. In any event, the encrypted key eKM.C6(K) is returned to the CFAP as an instruction output at 615.

The reader will appreciate that the IDK instruction has been designed to perform consistency checking within the cryptographic facility in lieu of returning the recovered clear values of C6 and EID to the CFAP and performing this consistency checking outside of the cryptographic facility. In the preferred embodiment, this consistency checking is performed in the cryptographic facility hardware and the recovered clear values of C6 and EID are not exposed outside the CF. The reason for doing this is to ensure that the DEA key distribution channel does not also provide a covert privacy channel whereby secret data may be incorporated in the control information portion of the key block and transmitted from the sending cryptographic device to the receiving cryptographic device. In a good cryptographic design, the cryptographic instructions will perform only those cryptographic functions for which they were designed, and no more. Doing so, limits the ways in which an attacker can manipulate the cryptographic instructions for the purpose of subverting their intended security. For example, a system administrator in charge of security policy for the sending and recieving locations, may have a security policy which prohibits the transmission of private messages over the communications link, for example when the link is dedicated merely to the transmission of new keys. In an alternate security policy where the system administrator is to selectively allow privacy channels, there should be no "back door" method for subverting the system administrator's authority in enabling or prohibiting such privacy channels. The use of the control information transmitted over the separate channel to the receiver, is to enable the recipient to inspect the type of uses imposed on the receive key and allow the recipient the option of rejecting the keyblock. However, an alternate embodiment is possible wherein the recovered

clear values of C6 and EID are returned to the CFAP and consistency checking is then performed by the CFAP.

Control Information: Fig. 12 further illustrates the unique role played by the encrypted key block, ePU(keyblk), and the external key token. Although Figs. 5, 7, and 8 depict external key token as containing an encrypted key block of the form ePU(keyblk) and reference control information in clear form, in the logical sense there are two information channels over which information flows: (1) and encrypted channel and (2) a clear channel. Referring now to Fig. 12, therein is shown two cryptographic systems, A and B, that communicate via a key distribution protocol through an encrypted channel 701 and a clear channel 702. Encrypted channel 701 is facilitated via the encrypted key block, ePU(keyblk). Control information in keyblk, which is subsequently encrypted with public key PU, is thus sent from A to B via an encrypted channel. Clear channel 702 is facilitated via the external key token. Control information in clear form stored in the external key token is, for all intents and purposes, passed from A to B via a clear channel.

Another distinguishing feature of the two channels is this. Encrypted channel 701 is a logical channel between the cryptographic facility 30 of cryptographic system A and cryptographic facility 30' of cryptographic system B. Clear channel 702 is a logical channel between application 36 in cryptographic system A and application 36' in cryptographic system B. and possibly a logical channel between CFAP 34 in cryptographic system A and CFAP 34' in cryptographic system B, depending on how the external key tokens are to be managed. In any event, the key distribution process is designed such that (1) in the case of encrypted channel 701, only the cryptographic facilities have access to the control information in keyblk, whereas (2) in the case of clear channel 702, the applications and possibly the CFAPs have access to the control information in the external key token as a routine part of the key distribution protocol. Since the CF is typically implemented within secure hardware, the CF is said to have a higher level of integrity than other parts of the the cryptographic system, such as the CFAP and applications operating within the cryptographic system. Thus, a higher degree of protection is achieved within the key distribution process by controlling that process, to the degree possible, from within the CF itself. To this end, control information is passed via encrypted channel 701, in keyblk, from A to B, thus enabling B to process the imported keyblk with a high degree of integrity. Of course, the assumption is made here that digital signatures are also a part of the key distribution process, as shown in Fig. 8, which forms another underpinning or layer of integrity that augments and enhances the overall integrity of information passed via encrypted channel 701.

Fig. 13 illustrates the process of reconciliation between control information transmitted via encrypted channel 701 and control information, called reference control information, transmitted via clear channel 702. The importance in having these two channels for passing control information can now be seen. Control information transmitted via encrypted channel 701 can be 'seen' by the cryptographic facility, but by no one outside the cryptographic facility. This ensures that the key distribution channel is not used as a covert privacy channel. Thus, the only way that the application program or the CFAP has of validating the control information transmitted via encrypted channel 701 is the specify reference control information to the CF in clear form. Since it is the application program or the CFAP that specifies the reference control information, the reference control information is consistency checked to determine its accuracy before being passed to the CF. Inside the protected boundary of the CF, the control information recovered from the decrypted keyblk is checked for consistency with the reference control information supplied in clear form by the application to the CFAP and thence by the CFAP to the CF. This permits all parties (CF, CFAP, and application) to be sure that the the control information associated with the to-be-imported key is correct and in accordance with expectations. In summary, all parties look at the reference control information and have a chance to agree or disagree with it, but only the CF sees the control information passed in keyblk and only the CF with highest integrity determines whether the control information received via encrypted channel 701 (i.e., in keyblk) is consistent with the reference control information received in the external key token by the application, or by the CFAP depending on whether key distribution is implemented at the application layer or at the cryptographic facility access program layer. Referring now to Fig. 13, reference control information (designated RCI) received via Clear Channel 702 is inspected by the receiving application program APPL 36. If APPL 36 finds the reference control information to be okay, i.e., it is accurate acceptable, and in accordance with the protocol, in all respects, then APPL 36 will issue a request to CFAP 34 to import the received DEA key, passing the reference control information in the received external key token to CFAP 34. If the CFAP 34 finds the reference control information to be okay, then CFAP 36 will issue an IDK instruction to CF 30 to import the received DEA key, passing the reference control information in the received external key token to CF 30. The control information

(designated CI) received via Encrypted Channel 701 and the reference control information RCI received from the CFAP 34 are inspected by themselves for consistency and then they are compared or consistency checked, one against the other, to determine that CI is consistent with RCI. Only if this consistency checking succeeds, will the CF 30 import the DEA key.

Fig. 14 is a block diagram of the cryptographic facility 30 in the sending location A, as it is organized for performing the generate key set PKA (GKSP) instruction, illustrated in Fig. 9. Fig. 14 shows the cryptographic facility 30 at the sending location which includes the crypto variable retrieval means 40, shown in greater detail in Fig. 16. To prepare a crypto variable such as the key K for transmission from the cryptographic facility 30 to the cryptographic facility 30' at the receiving location, the key K is accessed from the crypto variable retrieval means 40 over the line 62 and applied to the concatenation means 42. In addition, control information such as a control vector and an environmental identification are accessed over line 60 from the crypto variable retrieval means 40 and are applied to the concatenation means 42. Concatenation means 42 will concatenate the key K with the control vector and the environmental identification and that will form the key block 80 which is applied to the public key algorithm encryption means 44. The public key is accessed over line 70 from the crypto variable retrieval means 40 and is applied to the key input of the encryption means 44. The key block 80 is encrypted forming the encrypted key block 85 which is then applied to the transmitting means 46. The encrypted key block 85 is then transmitted over the transmission link 12 to the cryptographic facility 30' at the receiving location shown in Fig. 15. The control information consisting of the control vector and the environmental identification which has been accessed over line 60 is also output as a separate information unit to the transmitting means 46 for transmission over the link 12 to the cryptographic facility 30' at the receiving location. The control information, which can be referred here as the reference control information, is separate from the encrypted key block 85. The reference control information can be transmitted over the same physical communications link 12 as the encrypted key block 85, in a different time slot or frequency slot in the case of frequency division multiplexing. Alternately, completely separate physical communication links can be employed to transmit the reference control information as distinguished from the transmission of the encrypted key block 85. The transmission of the reference control information can be in either clear text form, or alternately the reference control information can be encrypted and transmitted over a privacy channel if the sender and receiver share suitable keys.

In the receiving cryptographic facility 30' shown in Fig. 15, the reference control information is transferred from the communications link 12 to the overline 74 to the control information comparison means 59. The encrypted key block 85 is transferred from the receiving means 56 to the public key algorithm decryption means 54. A privacy key is accessed over line 72 from the crypto variable retrieval means 40' and is applied to the key input of the decryption means 54. The operation of the decryption means 54 generates the recovered key block 80 which is applied to the extraction means 52. The extraction means 52 extracts the control information 60 from the recovered key block 80 and applies the extracted control information to the control information comparison means 59. The control information comparison means 59 then compares the identity of the extracted control information from the key block 80 with the reference control information received from the communications link 12 over line 74. The control information comparison means 59 has an enabling output signal 90 which is produced if the comparison is satisfied. The enabling signal 90 is applied to an enabling input of the crypto variable storage means 50. The crypto variable, in this example the key K, is output from the extraction means 52 on line 62 and applied to the crypto variable storage means 50. The key K will be successfully stored in a crypto variable storage means 50 if the enabling signal 90 is applied from the comparison means 59. In addition, the control information which can include the control vector and the environmental ID of the sending location, can also be stored in the crypto variable storage means 50, if the enabling signal 90 is present.

Further in accordance with the invention, a comparison can be made between the environmental ID of the receiving station B and the environmental ID of the transmitting station A, in order to ensure that the environmental ID for the receiving station B is not identical with the environmental ID contained in the recovered key block 80. This comparison can also be performed in the comparison means 59 and the successful comparison can be made necessary to the generation of the enabling signal 90 as described above. The environmental ID in the reference control information should successfully compare with the environmental ID extracted by the extraction means 52 from the key block 80. In addition, the environmental ID extracted from the key block 80, which represents the environmental ID of the sending location A, should not be the same as the environmental ID of the receiving station B. When these two conditions exist and also when the control vector in the refer-

ence control information successfully compares with the control vector extracted by the extraction means 52 from the key block 80, then the comparison means 59 will output an enabling signal 90 to the storage means 50.

The crypto variable retrieval means 40 and 40' is shown in greater detail in Fig. 16. Input parameters 311 can be transferred over line 33 from the external storage 400 in the CFAP 34. These input parameters can then be applied to the crypto facility 30, over lines 31. Op codes 310 in the CFAP 34 can also be applied over lines 31 to the crypto facility 30. The crypto facility 30 includes the crypto variable retrieval means 40 which contains a random number generator 95, a data encryption algorithm decryption means 410 and an output selection means 420. A master key storage 99 is contained in the crypto facility 30, having an output connected to the key input of the decryption means 410. The random number generator 95 can generate a first type key K' to be applied to the output selection means 420. Alternately, a second type key K" in clear text form can be applied to the output selection means 420. Alternately, a third type key K"' can be applied to the output selection means 420, which is derived from the decryption by the decryption means 410 of an encrypted form of the key K"' which has been encrypted under the exclusive OR product of the master key KM and the control vector C5. The output of the selection means 420 is the key K which is the crypto variable which is discussed in relation to Figs. 14 and 15.

In the preferred embodiment of the invention, public key encryption is used as the encryption technique for transmitting the key block from the sending location to the receiving location, however, it is within the scope of the invention to use symmetric, private key techniques for enciphering and deciphering the key block. Also, in the preferred embodiment of the invention, where digital signatures are employed, as described above, the public key encryption technique for forming digital signatures is employed. However, in an alternate embodiment, conventional Message Authentication Code (MAC) techniques may be employed using a private key algorithm. In the preferred embodiment of the invention, Data Encryption Standard (DES) key is the crypto variable which is transmitted in the key block from the sending location to the receiving location, however, in alternate embodiments of the invention, the crypto variable can be a public key or a non-key-type expression.

Although a specific embodiment of the invention has been disclosed, it will be understood by those having skill in the art that changes can be made to the specific embodiment without departing from the spirit and the scope of the invention.

Claims

1. In a data processing system having a plurality of communicating nodes, at least a pair of nodes in the system exchanging cryptographic communications, an apparatus for enabling a first node of the pair to control a crypto variable after its transmission from the first node to a second node of the pair, comprising:

   a storage means at a transmitting node in the system for storing a crypto variable which is to be transmitted to a receiving node in the system;

   said storage means storing control information including a control vector to control said crypto variable after it is transmitted from said transmitting node;

   said storage means storing a first key expression;

   concatenating means at said transmitting node, coupled to said storage means, for concatenating said crypto variable with said control information, forming a key block;

   encryption means at said transmitting node, coupled to said storage means and said concatenating means, for encrypting said key block with said first key expression, forming an encrypted key block; and

   transmitting means at said transmitting node coupled to said encryption means and coupled over a communications link to a receiving means at said receiving node, for transmitting said encrypted key block to said receiving node.

2. In a data processing system having a plurality of communicating nodes, at least a pair of nodes in the system exchanging cryptographic communications, an apparatus for enabling a first node of the pair to control a crypto variable after its transmission from the first node to a second node of the pair, comprising:

   a first storage means at a transmitting node in the system for storing a crypto variable which is to be transmitted to a receiving node in the system;

   a second storage means at said transmitting node for storing control information to control said crypto variable after it is transmitted from said transmitting node said control information

including a control vector to limit the uses of said crypto variable;

a third storage means at said transmitting node for storing a first key expression;

concatenating means at said transmitting node, coupled to said first and second storage means, for concatenating said crypto variable with said control information, forming a key block;

encryption means at said transmitting node, coupled to said third storage means and said concatenating means, for encrypting said key block with said first key expression, forming an encrypted key block;

transmitting means at said transmitting node coupled to said encryption means and coupled over a communications link to a receiving means at said receiving node, for transmitting said encrypted key block to said receiving node;

said transmitting means coupled to said second storage means, for transmitting a second copy of said control information to said receiving node;

fourth storage means at said receiving node, for storing a second key expression corresponding to said first key expression;

decryption means at said receiving node coupled to said receiving means and to said fourth storage means, for decrypting said encrypted key block using said second key expression, to obtain a recovered key block;

extraction means at said receiving node coupled to said decryption means, to extract said control information and said crypto variable from said recovered key block;

comparison means at said receiving node coupled to said extraction means and coupled to said receiving means for comparing said control information extracted from said recovered key block to said second copy of said control information, said comparison means having an enabling output for signalling when said comparison is satisfied;

control means coupled to said extraction means and having an enabling input coupled to said output of said comparison means, for controlling said crypto variable with said control information.

3. Apparatus for generating and distributing a Data Encryption Algorithm (DEA) key in a communications network, comprising:

a) sending means for generating and producing at least two copies of a key-encrypting key (k-ek), and control information including a control vector for permitted uses of the k-ek;

b) means included in the sending means for encrypting one copy of the k-ek under the public key of a receiving means and transmitting the public key encrypted k-ek to the receiving means in association with said control information;

c) means further included in the sending means for encrypting another copy of the k-ek under a master key of the sending means;

d) means further included in the sending means for storing the master key encrypted k-ek as a common distributing key for other encrypted keys used in the network, in association with said control information;

e) control means included in the sending means, to limit uses of the k-ek to said permitted uses in response to said control information.

4. The apparatus of claim 1, 2, or 3, wherein:

said first key expression and said second key expression being symmetric keys, and/or wherein

said first key expression being a public key issued by said receiving node and said second key expression being a private key corresponding to said public key.

5. The apparatus of anyone of the preceeding claims, wherein said control means further comprises:

a reference storage means at said receiving node for storing a reference control vector characterizing required uses of said crypto variable at said receiving station;

a received control vector storage means at said receiving node, coupled to said extraction means, for storing said received control vector extracted from said recovered key block;

said comparison means at said receiving node coupled to said reference storage means and to said received control vector storage means,

for comparing said reference control vector with said received control vector, and outputting an acceptance signal if the comparison succeeds;

crypto variable storage means at said receive node coupled to said extraction means and to said comparison means, for storing said crypto variable extracted by said extraction means if said acceptance signal is received from said compare means.

6. The apparatus of claim 5, wherein said crypto variable storage means further comprises:

a master key storage means at said receiving node for storing a master key;

an exclusive OR means at said receiving node coupled to said master key storage means and to said received or reference control vector storage means respectively, for forming an exclusive OR product of said master key and said received or reference control vector, respectively, forming a product key expression;

an encryption engine at said receiving node having a key input coupled to said exclusive OR means for inputting said product key expression, and having an operand input coupled to said crypto variable storage means, for encrypting said crypto variable under said master key, forming an encrypted crypto variable;

an encrypted crypto variable storage means at said receiving node, coupled to said encryption engine, for storing said encrypted crypto variable.

7. The apparatus of claim 6, which further comprises:

a control vector checking means at said receiving node coupled to a user input, for receiving a request from a user for using said crypto variable;

said control vector checking means being coupled to said received control vector storage means, for checking said received control vector to determine if said requested uses are permitted;

said control vector checking means outputting an enabling signal if said requested uses are permitted;

a processing means at said receiving node coupled to said control vector checking means, to said received control vector storage means and to said master key storage means, for receiving said enabling signal and in response thereto, forming an exclusive OR product of said master key and said received control vector, forming a product key expression;

a decryption engine at said receiving node having a key input coupled to said exclusive OR means for inputting said product key expression, and having an operand input coupled to said encrypted crypto variable storage means, for decrypting said encrypted crypto variable under said master key, recovering said crypto variable.

8. The apparatus of claim 6 or 7, which further comprises:

a control vector checking means at said receiving node coupled to a user input, for receiving a request from a user for using said crypto variable;

said control vector checking means being coupled to said reference control vector storage means, for checking said reference control vector to determine if said requested uses are permitted;

said control vector checking means outputting an enabling signal if said requested uses are permitted;

a processing means at said receiving node coupled to said control vector checking means, to said reference control vector storage means and to said master key storage means, for receiving said enabling signal and in response thereto, forming an exclusive OR product of said master key and said reference control vector, forming a product key expression;

a decryption engine at said receiving node having a key input coupled to said exclusive OR means for inputting said product key expression, and having an operand input coupled to said encrypted crypto variable storage means, for decrypting said encrypted crypto variable under said master key, recovering said crypto variable;

wherein said reference control vector is received preferably from said transmitting node.

9. The apparatus of claim 8, which further comprises:

said received control vector is a first hashed product of said reference control vector, received from said transmitting node;

hashing means in said receiving node coupled to said reference control vector storage means, for forming a second hash product of said reference control vector;

second comparison means coupled to said received control vector storage means and to said hashing means, for comparing said first hashed product with said second hashed product and outputting a second acceptance signal when the comparison is satisfied.

10. The apparatus of anyone of claims 1 to 4, which further comprises:

said control information includes a hashed control vector which represents limitations on uses of said crypto variable.

11. The apparatus of claim 10, wherein said control means further comprises:

a reference control vector storage means at said receiving node for receiving from said transmitting node and storing a reference control vector characterizing required uses of said crypto variable at said receiving station;

a hashed control vector storage means at said receiving node, coupled to said extraction means, for storing said hashed control vector extracted from said recovered key block;

hashing means in said receiving node coupled to said reference control vector storage means, for forming a hash product of said reference control vector;

compare means at said receiving node coupled to said hashing means and to said hashed control vector storage means, for comparing said hash product with said hashed control vector, and outputting an acceptance signal if the comparison succeeds;

crypto variable storage means at said receive node coupled to said extraction means and to said compare means, for storing said crypto variable extracted by said extraction means if said acceptance signal is hashed from said compare means.

12. The apparatus of claim 11, wherein said crypto variable storage means further comprises:

a master key storage means at said receiving node for storing a master key;

an exclusive OR means at said receiving node coupled to said master key storage means and to said hashed control vector storage means, for forming an exclusive OR product of said master key and said hashed control vector, forming a product key expression;

an encryption engine at said receiving node having a key input coupled to said exclusive OR means for inputting said product key expression, and having an operand input coupled to said crypto variable storage means, for encrypting said crypto variable under said master key, forming an encrypted crypto variable;

an encrypted crypto variable storage means at said receiving node, coupled to said encryption engine, for storing said encrypted crypto variable.

13. The apparatus of claim 12, which further comprises:

a control vector checking means at said receiving node coupled to a user input, for receiving a request from a user for using said crypto variable;

said control vector checking means being coupled to said reference control vector storage means, for checking said reference control vector to determine if said requested uses are permitted;

said control vector checking means outputting an enabling signal if said requested uses are permitted;

a processing means at said receiving node coupled to said control vector checking means, to said hashed control vector storage means and to said master key storage means, for receiving said enabling signal and in response thereto, forming an exclusive OR product of said master key and said hashed control vector, forming a product key expression;

a decryption engine at said receiving node having a key input coupled to said exclusive OR means for inputting said product key expression, and having an operand input coupled to said encrypted crypto variable storage

means, for decrypting said encrypted crypto variable under said master key, recovering said crypto variable.

14. The apparatus of anyone of the preceeding claims, which further comprises:

said control information includes a transmitting node environment identification which characterizes the identity of said transmitting node.

15. The apparatus of claim 14, wherein said control means further comprises:

a receiving node environment identification storage means at said receiving node for storing a receiving node environment identification;

a received transmission node environment identification storage means at said receiving node, coupled to said extraction means, for storing said transmitting node environment identification extracted from said recovered key block;

compare means at said receiving node coupled to said receiving node environment identification storage means and to said received transmission node environment identification storage means, for comparing said receiving node environment identification and said transmitting node environment identification and outputting an acceptance signal if the comparison fails;

crypto variable storage means at said receive node coupled to said extraction means and to said compare means, for storing said crypto variable extracted by said extraction means if said acceptance signal is received from said compare means.

16. In a data processing system having a plurality of communicating nodes, at least a pair of nodes in the system exchanging cryptographic communications, a method for enabling a first node of the pair to control a crypto variable after its transmission from the first node to a second node of the pair, comprising:

storing a crypto variable which is to be transmitted to a receiving node in the system, at a transmitting node;

storing control information to control said crypto variable after it is transmitted from said transmitting node, at said transmitting node said control information including a control vector to limit the uses of said crypto variable;

storing a first key expression at said transmitting node;

concatenating said crypto variable with said control information, forming a key block, at said transmitting node;

encrypting said key block with said first key expression, forming an encrypted key block, at said transmitting node;

transmitting said encrypted key block to said receiving node;

transmitting a second copy of said control information to said receiving node;

storing a second key expression corresponding to said first key expression, at said receiving node;

decrypting said encrypted key block using said second key expression, to obtain a recovered key block, at said receiving node;

extracting said control information and said crypto variable from said recovered key block, at said receiving node;

comparing said control information extracted from siad recovered key block with said second copy of said control information and generating an enabling signal when the compare is satisfied;

controlling said crypto variable with said control information when said enabling signal has been generated.

17. In a data processing system having a plurality of communicating nodes, at least a pair of nodes in the system exchanging cryptographic communications, a method for enabling a first node of the pair to control a crypto variable after its transmission from the first node to a second node of the pair, comprising:

concatenating a crypto variable with control information including a control vector to control said crypto variable after it is transmitted from said transmitting node, forming a key block, at said transmitting node;

encrypting said key block with a first key expression, forming an encrypted key block, at said transmitting node;

transmitting said encrypted key block to said receiving node;

decrypting said encrypted key block using a second key expression, to obtain a recovered key block, at said receiving node;

extracting said control information and said crypto variable from said recovered key block, at said receiving node;

validating said control information extracted from said recovered key block and generating an enabling signal;

controlling said crypto variable with said control information when said enabling signal has been generated.

18. In a data processing system having a plurality of communicating nodes, at least a pair of nodes in the system exchanging cryptographic communications, a program for execution on the data processing system for enabling a first node of the pair to control a crypto variable after its transmission from the first node to a second node of the pair, comprising:

said program controlling the data processing system for storing a crypto variable which is to be transmitted to a receiving node in the system, at a transmitting node;

said program controlling the data processing system for storing control information to control said crypto variable after it is transmitted from said transmitting node, at said transmitting node said control information including a control vector to limit the uses of said crypto variable;

said program controlling the data processing system for storing a first key expression at said transmitting node;

said program controlling the data processing system for concatenating said crypto variable with said control information, forming a key block, at said transmitting node;

said program controlling the data processing system for encrypting said key block with said first key expression, forming an encrypted key block, at said transmitting node;

said program controlling the data processing system for transmitting said encrypted key block to said receiving node;

said program controlling the data processing system for transmitting a second copy of said control information to said receiving node;

said program controlling the data processing system for storing a second key expression corresponding to said first key expression, at said receiving node;

said program controlling the data processing system for decrypting said encrypted key block using said second key expression, to obtain a recovered key block, at said receiving node;

said program controlling the data processing system for extracting said control information and said crypto variable from said recovered key block, at said receiving node;

said program controlling the data processing system for comparing said control information extracted from said recovered key block with said second copy of said control information and generating an enabling signal when the compare is satisfied;

said program controlling the data processing system for controlling said crypto variable with said control information when said enabling signal has been generated.

19. The method of claim 16 or 17, or the program of claim 18, which further comprises:

said first key expression and said second key expression being symmetric keys, and/or

said first key expression being a public key issued by said receiving node and said second key expression being a private key corresponding to said public key.

20. The method of claim 16, 17, or the program of claim 18 or 19, which further comprises:

said control information includes a received control vector which defines limitations on uses of said crypto variable.

21. The program of claim 20, which further comprises:

said program controlling the data processing system for storing a reference control vector characterizing required uses of said crypto

variable at said receiving node;

said program controlling the data processing system for storing said received control vector extracted from said recovered key block, at said receiving node;

said program controlling the data processing system for comparing said reference control vector with said received control vector, and outputting an acceptance signal if the comparison succeeds, at said receiving node;

said program controlling the data processing system for storing said crypto variable extracted by said extraction means if said acceptance signal is received from said compare means, at said receiving node.

22. The program of claim 21, which further comprises:

said program controlling the data processing system for storing a master key at said receiving node;

said program controlling the data processing system for forming an exclusive OR product of said master key and said received control vector, forming a product key expression, at said receiving node;

said program controlling the data processing system for encrypting said crypto variable under said master key, forming an encrypted crypto variable, at said receiving node;

said program controlling the data processing system for storing said encrypted crypto variable, at said receiving node.

23. The program of claim 22, which further comprises:

said program controlling the data processing system for receiving a request from a user for using said crypto variable, at said receiving node;

said program controlling the data processing system for checking said received control vector to determine if said requested uses are permitted, at said receiving node;

said program controlling the data processing system for outputting an enabling signal if said requested uses are permitted, at said receiving node;

said program controlling the data processing system for receiving said enabling signal and in response thereto, forming an exclusive OR product of said master key and said received control vector, forming a product key expression, at said receiving node;

said program controlling the data processing system for inputting said product key expression, and decrypting said encrypted crypto variable under said master key, recovering said crypto variable, at said receiving node.

24. The program of claim 21, which further comprises:

said program controlling the data processing system for storing a master key at said receiving node;

said program controlling the data processing system for forming an exclusive OR product of said master key and said reference control vector, forming a product key expression, at said receiving node;

said program controlling the data processing system for inputting said product key expression, and encrypting said crypto variable under said master key, forming an encrypted crypto variable, at said receiving node;

said program controlling the data processing system for storing said encrypted crypto variable, at said receiving node.

25. The program of claim 24, which further comprises:

said program controlling the data processing system for receiving a request from a user for using said crypto variable, at said receiving node;

said program controlling the data processing system for checking said reference control vector to determine if said requested uses are permitted, at said receiving node;

said program controlling the data processing system for outputting an enabling signal if said requested uses are permitted, at said receiving node;

said program controlling the data processing system for receiving said enabling signal and in response thereto, forming an exclusive OR

product of said master key and said reference control vector, forming a product key expression, at said receiving node;

said program controlling the data processing system for decrypting said encrypted crypto variable under said master key, recovering said crypto variable, at said receiving node;

wherein said reference control vector is received preferably from said transmitting node.

26. The program of claim 25, which further comprises:

said received control vector is a first hashed product of said reference control vector, received from said transmitting node;

said program controlling the data processing system for forming a second hash product of said reference control vector, at said receiving node;

said program controlling the data processing system for comparing said first hashed product with said second hashed product and outputting a second acceptance signal when the comparison is satisfied, at said receiving node.

27. The program of claim 20, which further comprises:

said control information includes a hashed control vector which represents limitations on uses of said crypto variable.

28. The program of claim 27, wherein said control means further comprises:

said program controlling the data processing system for receiving from said transmitting node and storing a reference control vector characterizing required uses of said crypto variable at said receiving station, at said receiving node;

said program controlling the data processing system for storing said hashed control vector extracted from said recovered key block, at said receiving node;

said program controlling the data processing system for forming a hash product of said reference control vector, at said receiving node;

said program controlling the data processing system for comparing said hash product with said hashed control vector, and outputting an acceptance signal if the comparison succeeds, at said receiving node;

said program controlling the data processing system for storing said crypto variable extracted by said extraction means if said acceptance signal is hashed from said compare means, at said receiving node.

29. The program of claim 28, which further comprises:

said program controlling the data processing system for storing a master key at said receiving node;

said program controlling the data processing system for forming an exclusive OR product of said master key and said hashed control vector, forming a product key expression, at said receiving node;

said program controlling the data processing system for inputting said product key expression, and encrypting said crypto variable under said master key, forming an encrypted crypto variable, at said receiving node;

said program controlling the data processing system for storing said encrypted crypto variable, at said receiving node.

30. The program of claim 29, which further comprises:

said program controlling the data processing system for receiving a request from a user for using said crypto variable, at said receiving node;

said program controlling the data processing system for checking said reference control vector to determine if said requested uses are permitted, at said receiving node;

said program controlling the data processing system for outputting an enabling signal if said requested uses are permitted, at said receiving node;

said program controlling the data processing system for receiving said enabling signal and in response thereto, forming an exclusive OR product of said master key and said hashed control vector, forming a product key expression, at said receiving node;

said program controlling the data processing system for inputting said product key expression, and decrypting said encrypted crypto variable under said master key, recovering said crypto variable, at said receiving node.

31. The method or the program of anyone of the claims 16 to 30, which further comprises:

said control information includes a transmitting node environment identification which characterizes the identity of said transmitting node.

32. The program of claim 31, which further comprises:

said program controlling the data processing system for storing a receiving node environment identification, at said receiving node;

said program controlling the data processing system for storing said transmitting node environment identification extracted from said recovered key block, at said receiving node;

said program controlling the data processing system for comparing said receiving node environment identification and said transmitting node environment identification and outputting an acceptance signal if the comparison fails, at said receiving node;

said program controlling the data processing system for storing said crypto variable extracted by said extraction means if said acceptance signal is received from said compare means, at said receiving node.

FIG. 1

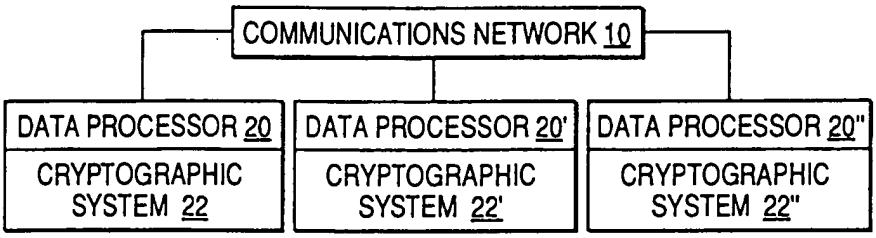


FIG. 2

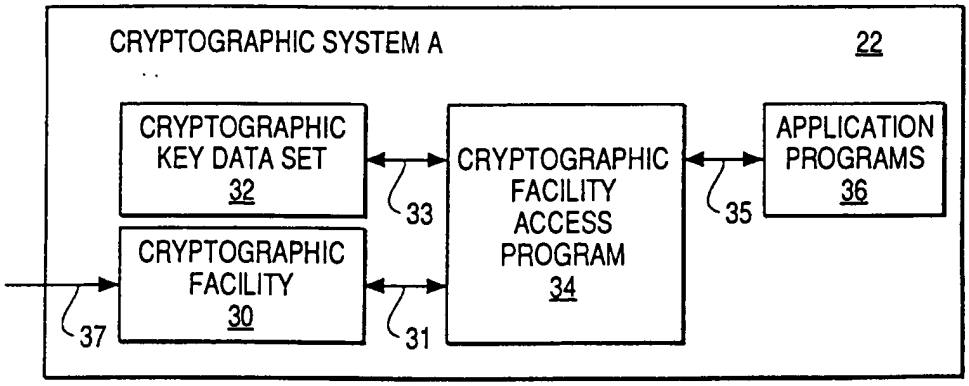


FIG. 3

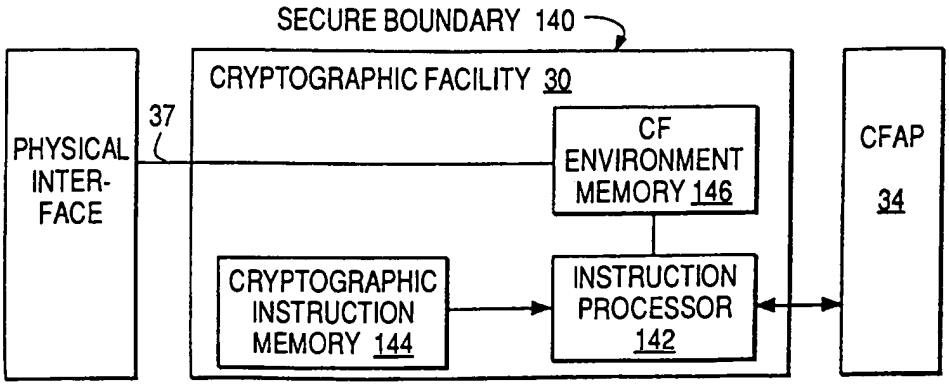


FIG. 4

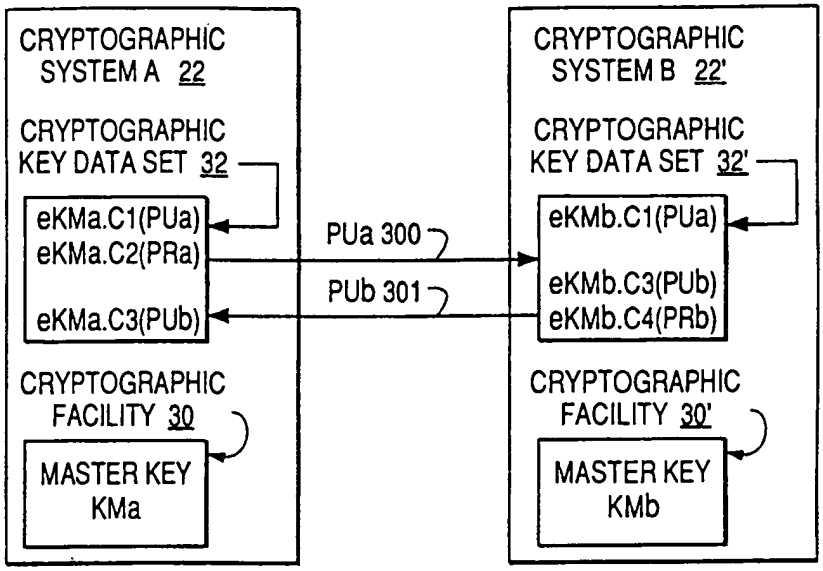


FIG. 5

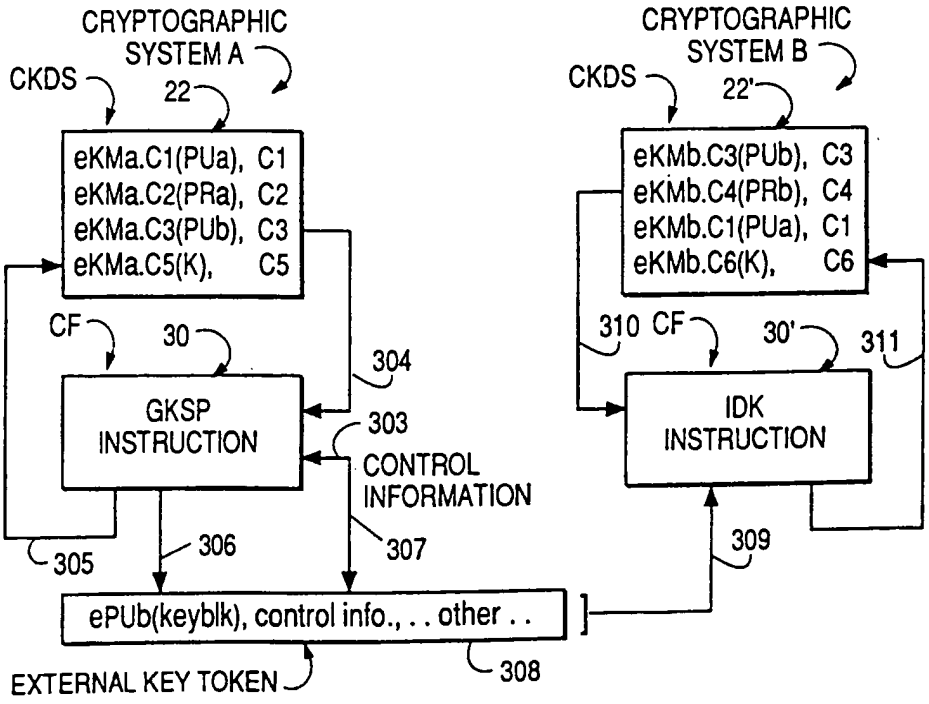


FIG. 6

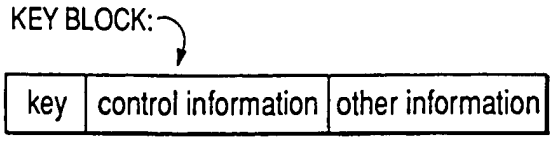


FIG. 7

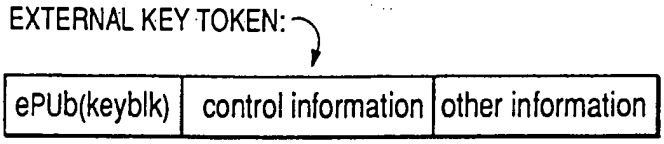


FIG. 8

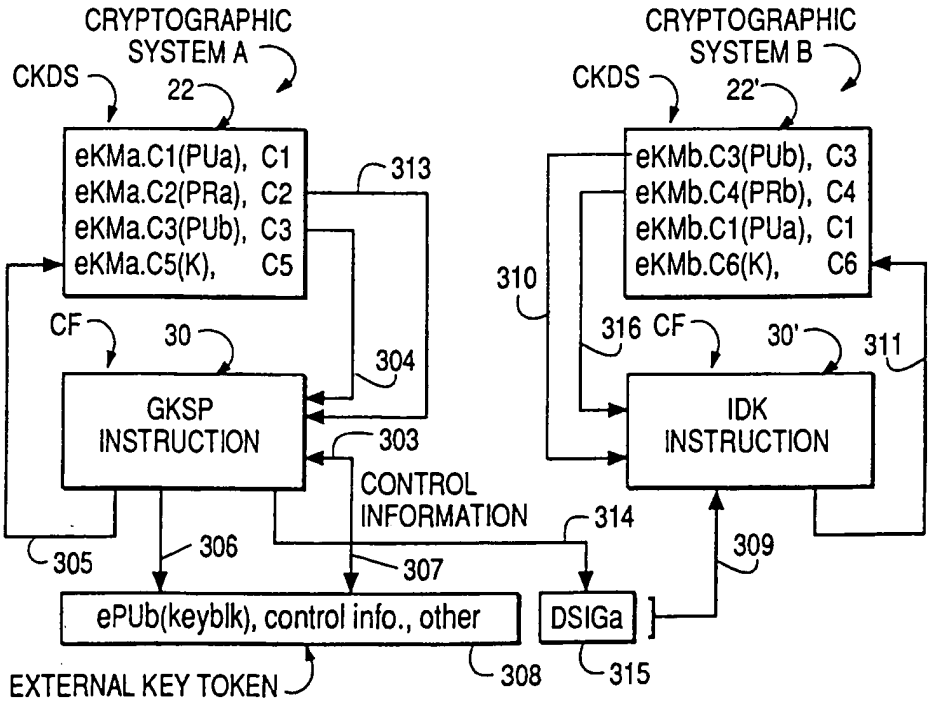


FIG. 9

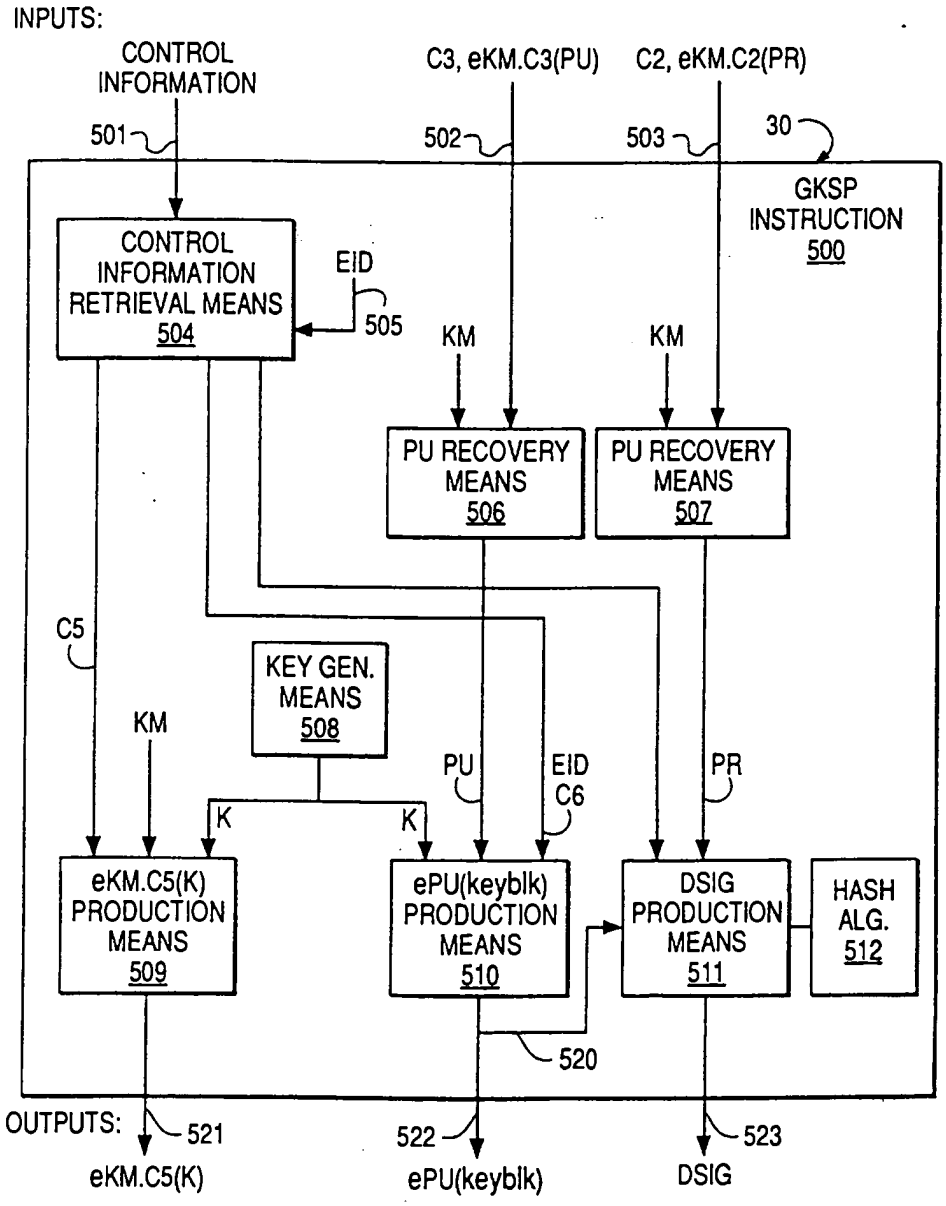


FIG. 10

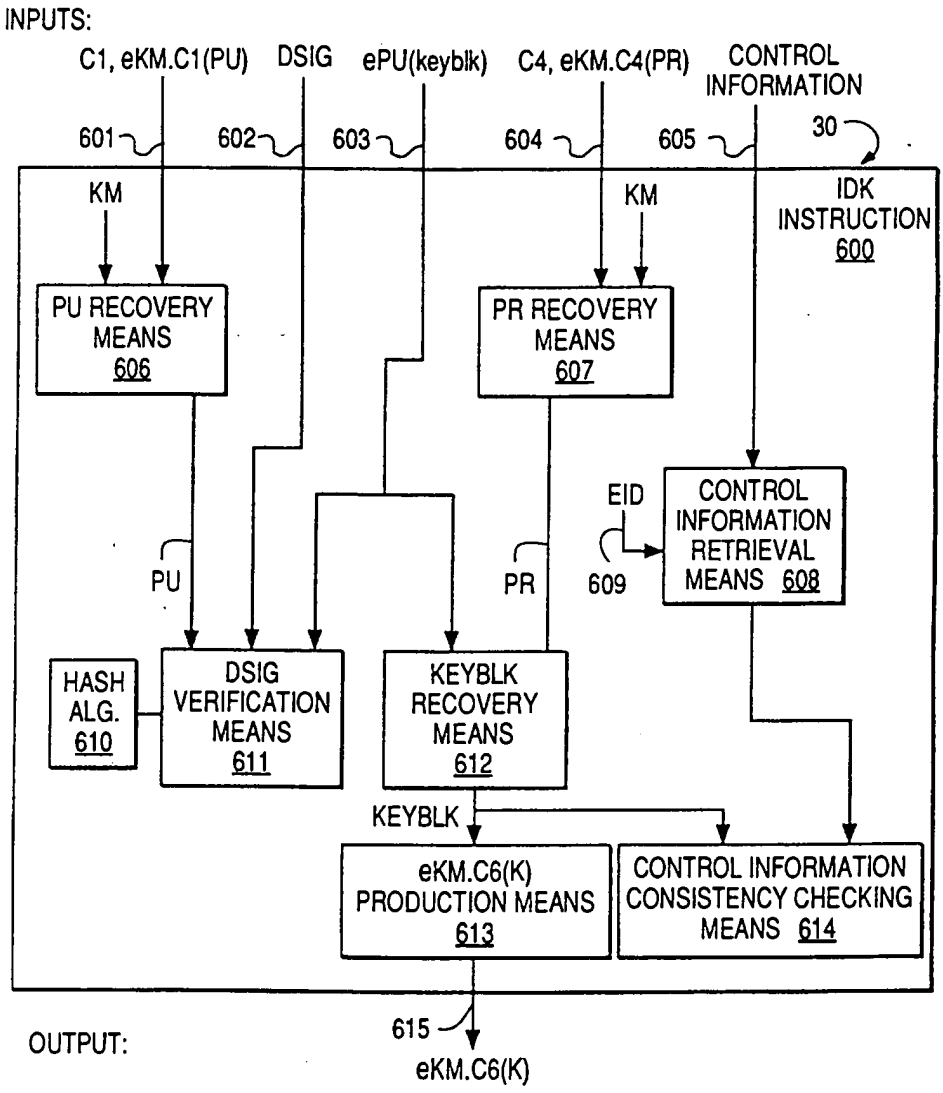


FIG. 11

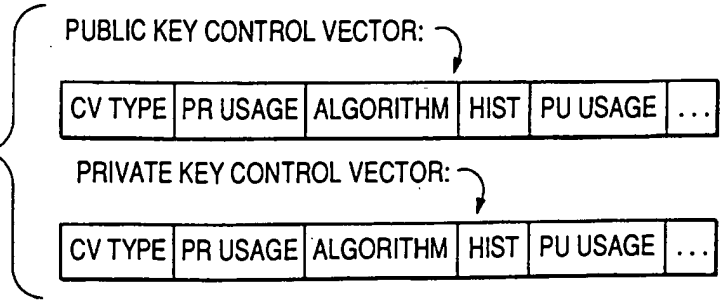


FIG. 12

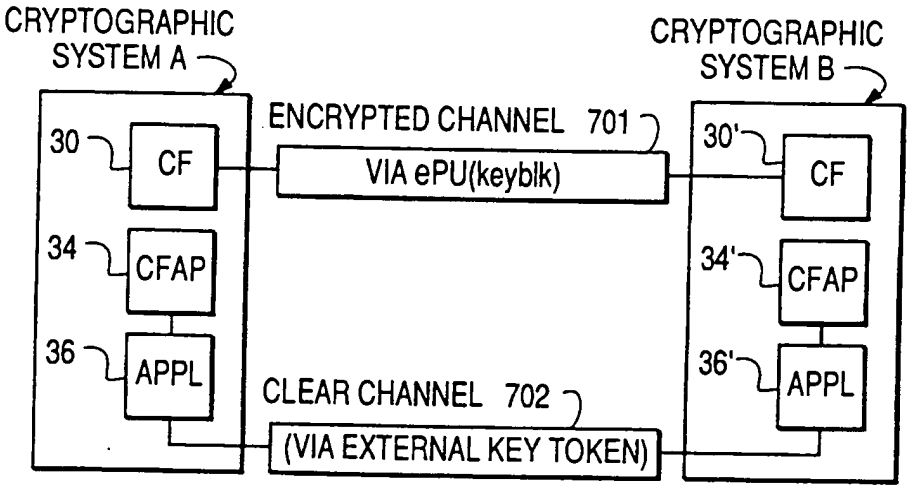


FIG. 13

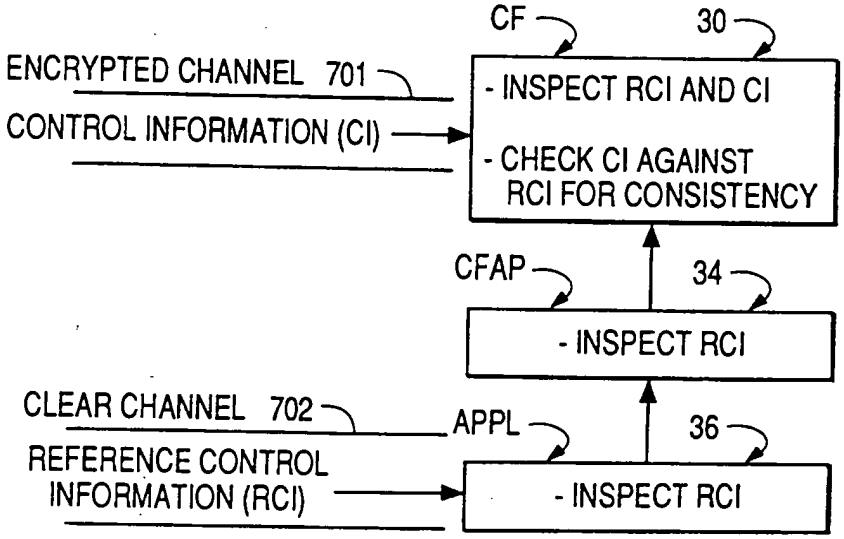


FIG. 14

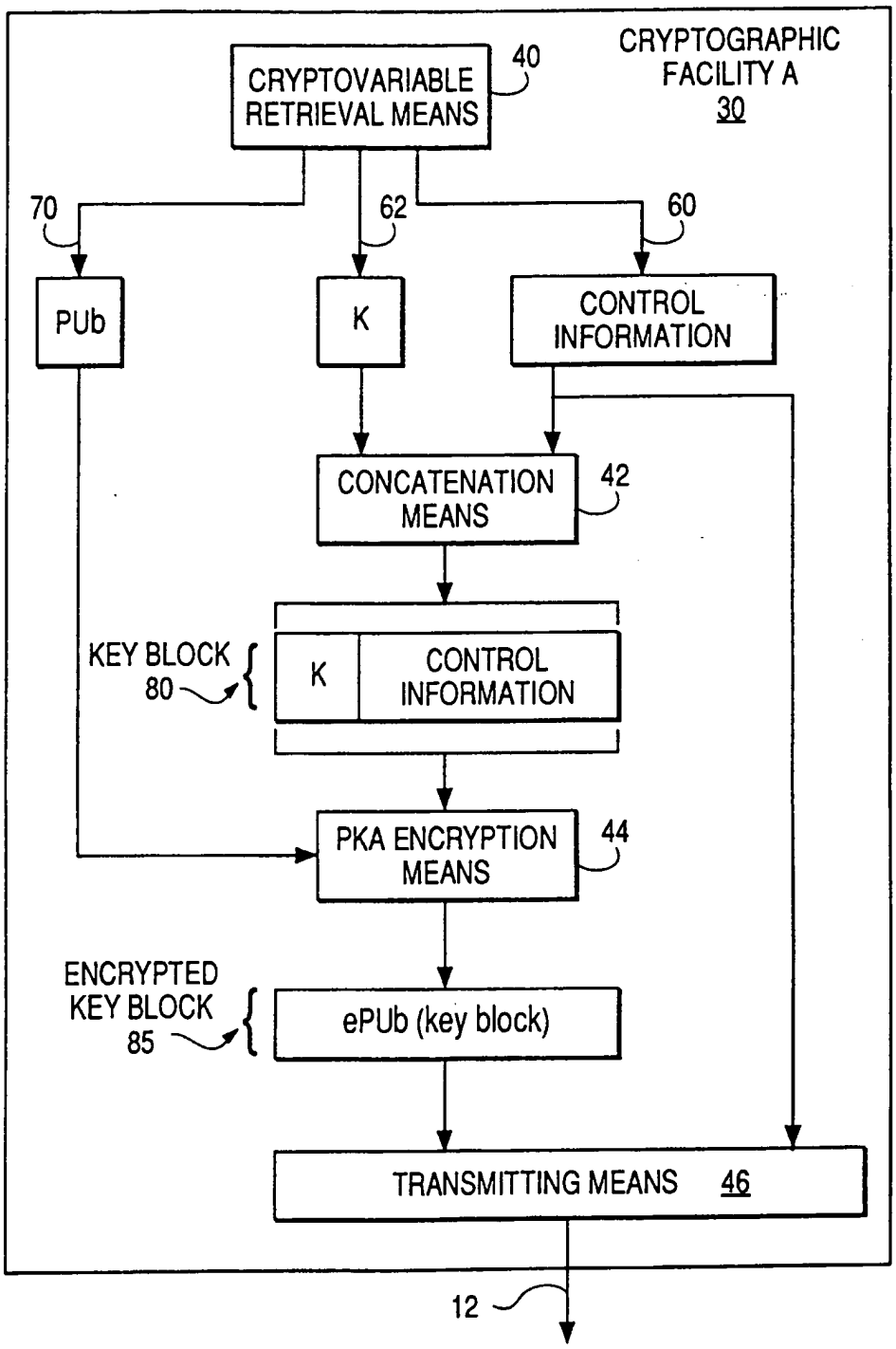


FIG. 15

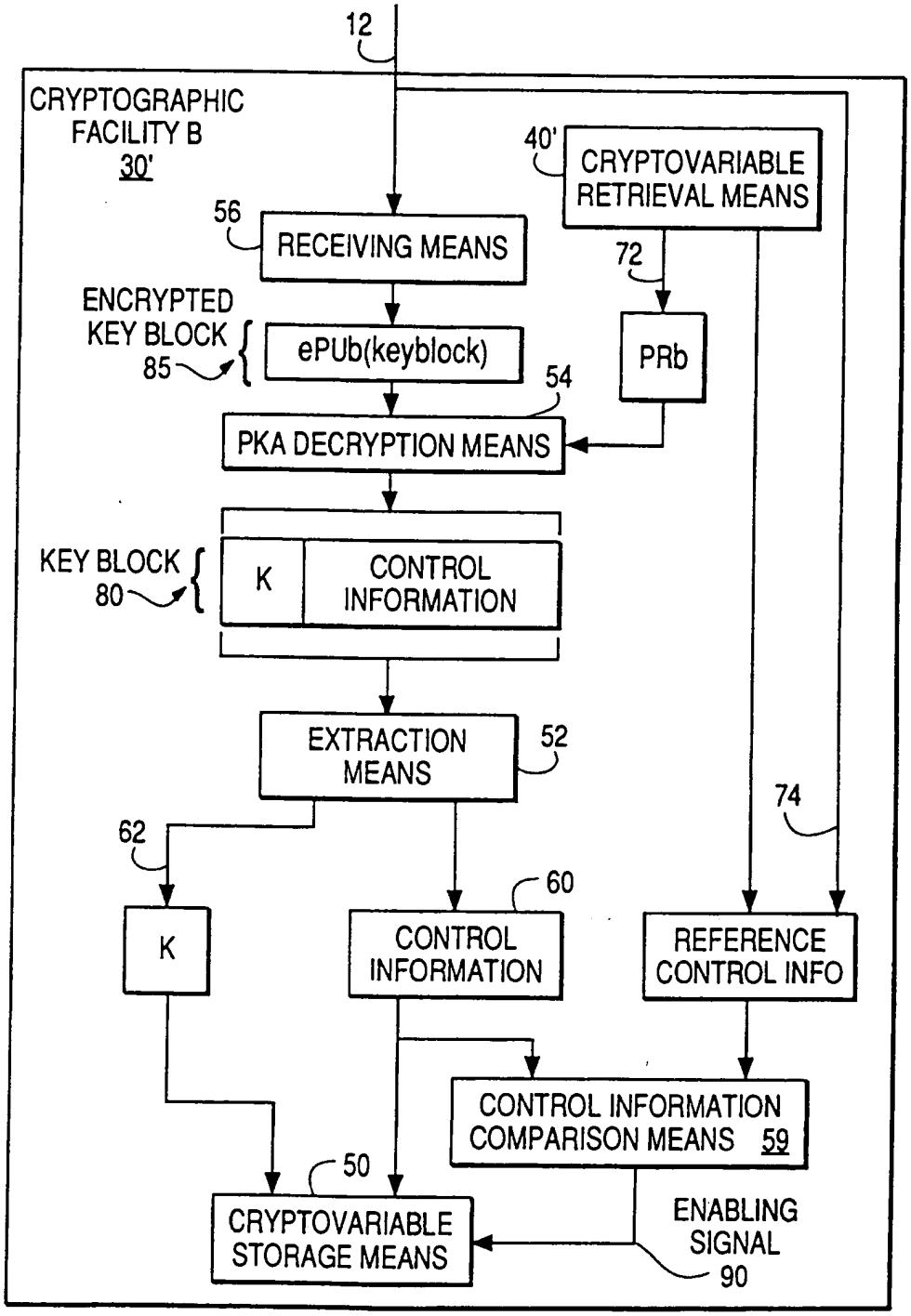
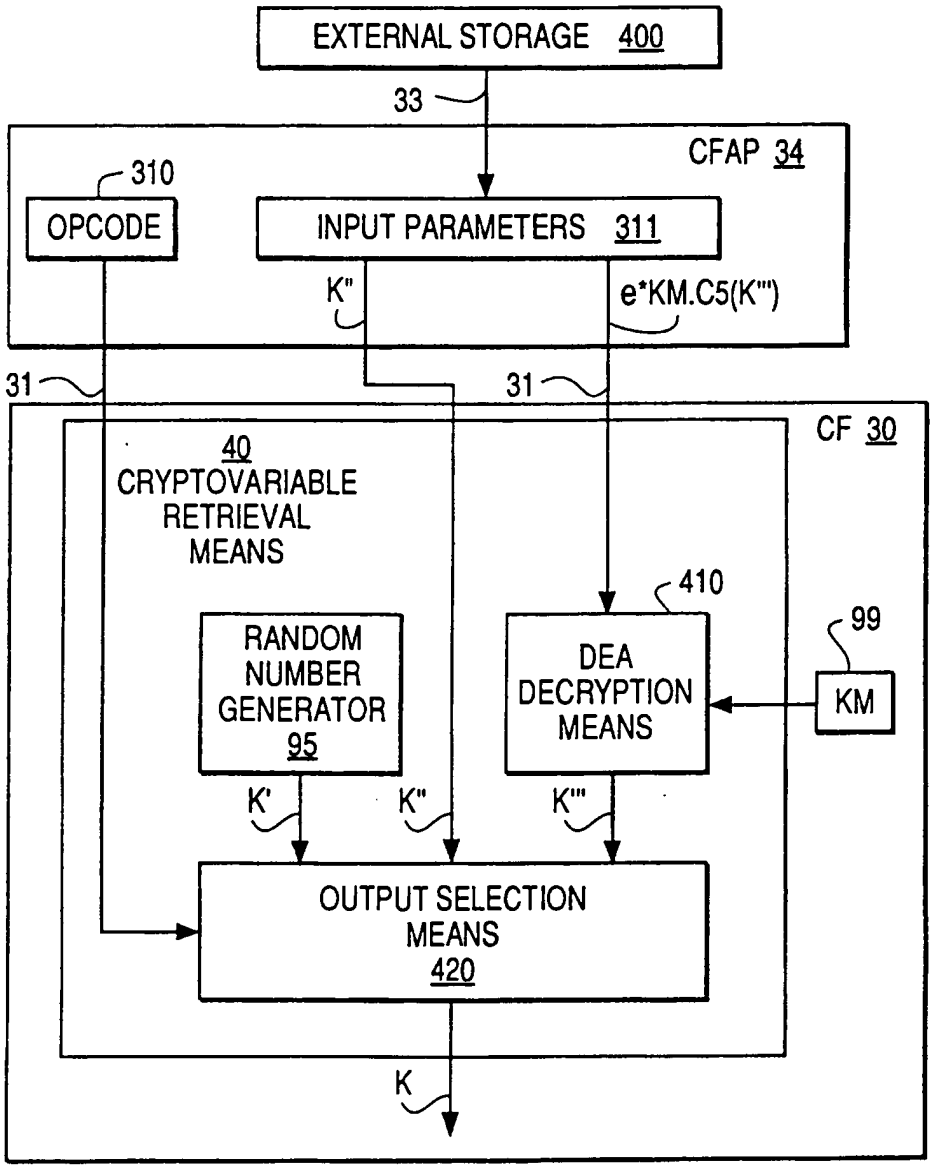


FIG. 16



(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 613 073 A1**

(12) EUROPEAN PATENT APPLICATION

(21) Application number: **93306468.5**

(51) Int. Cl.5: **G06F 1/00**

(22) Date of filing: **17.08.93**

(30) Priority: **23.02.93 GB 9303595**

(43) Date of publication of application:
31.08.94 Bulletin 94/35

(84) Designated Contracting States:
DE FR GB SE

(71) Applicant: **INTERNATIONAL COMPUTERS LIMITED**
ICL House
Putney, London, SW15 1SW (GB)

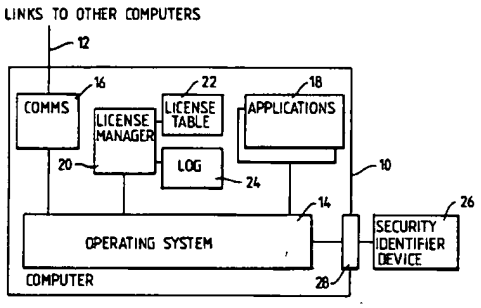
(72) Inventor: **Archer, Barrie**
Lilac Cottage,
Honey Hall
Wokingham, Berkshire RG11 3BA (GB)

(74) Representative: **Guyatt, Derek Charles**
Intellectual Property Department
International Computers Limited et al
Cavendish Road
Stevenage, Herts, SG1 2DY (GB)

(54) Licence management mechanism for a computer system.

(57) A computer system includes a license manager for regulating usage of software items. The license manager checks the host identity of the computer on which it runs and permits usage only if the host identity matches an identity value in a license key. The host identity of the computer is supplied by a security identification device removably coupled to an external port on the computer. Communication of the host identity between the security identifier device and the license manager is protected by encryption.

Fig.1.



EP 0 613 073 A1

Background to the invention

This invention relates to a license management mechanism for a computer system, for controlling use of licensed software.

Software is normally licensed rather than sold in order that restrictions on unauthorised use can be legally enforced. Various schemes have been tried to make the software enforce these restrictions itself, including copy protection, hardware keys, etc., but the current trend is to the use of license keys that are packets of data which permit the software to work only on a particular machine.

One way in which this has been implemented is through the provision of a mechanism referred to as a license manager to which the handling of these license keys is delegated. By centralising the handling of the license keys it is possible to restrict the use of software not just to a single machine but to a network of machines. This provides additional flexibility for the user as well as providing the potential for more sophisticated control over the use of the software within a user organisation.

Central to the use of license managers to control the use of software in this way is the ability to identify which machine the license manager is running. If this were not done it would be possible to obtain license keys for use on one machine and use them on any number of machines. Various schemes have been used to achieve this identification, including serial numbers built into the machine processor, use of Ethernet DTE addresses, etc.

The object of the present invention is to provide a novel way of identifying the machine on which a license manager is running.

Summary of the invention

According to the invention there is provided a computer system including a license manager for regulating usage of software items in accordance with license keys issued to the license manager, the license manager being arranged to check the host identity of the computer on which it runs and to permit usage only if the host identity matches an identity value in the license keys, characterised in that the host identity of the computer is supplied by a security identification device removably coupled to an external port on the computer.

Such identification devices have been used for PC software to permit the software to run only on machines that have the device attached. These devices are usually referred to as dongles. The present invention differs from such known use of dongles in that in the present case the device is used to identify the machine to the license manager, rather than to authorise a particular item of software.

Brief description of the drawings

Figure 1 is a block diagram of a computer system embodying the invention.

Figure 2 is a flow chart showing the operation of a license manager in response to a request to use a feature.

Figure 3 is a flow chart showing a host identity checking function performed by the license manager.

Description of an embodiment of the invention

One embodiment of the invention will now be described by way of example with reference to the accompanying drawing.

Referring to Figure 1, the system comprises a number of computers 10, linked together by means of communications links 12 to form a data processing network.

Each of the computers runs an operating system 14 which controls and coordinates the operation of the computer, and communications software 16 which allows the computer to communicate with the other computers in the system over the links 12. Each computer also runs a number of applications 18 (where an application is any logical software entity).

At least one of the computers runs a program referred to herein as the license manager (LM) 20. The function of the LM is to regulate the applications within a particular domain, so that each application can be used only to the extent permitted by licenses granted to the system owner. The domain comprises those applications that can communicate with the LM. In this example, the domain extends over a multi-computer network, but in other examples it could consist of a single computer.

Each application has a number of features associated with it. A "feature" is defined herein as an aspect of an application that is subject to license control by the LM. A feature may, for example, simply be the invocation of the application by a user. However, more complex features may be defined such as number of users, number of communication links and database size.

Each application also has an application key associated to it, which is unique to the application. As will be described, application keys are used to ensure security of communication between the applications and the LM.

The LM has a private area of memory in which it maintains a license table 22 and a log 24.

The license table holds a number of license keys that have been issued for this system. Each license key contains the following package of information:-

Machine identifier: the identity of the computer on

which the license manager is permitted to run.
Expiry date: the date until which the license key is valid.
Limit: the number of units of a particular feature that are licensed (eg the number of users, number of communication links, or database size).
Application key: the key value of the application to which the license key relates.
Signature: a cryptographic signature which ensures that the license key cannot be changed without detection.

Whenever one of the applications requires to use a feature, it sends a request message to the LM. The request message includes:

- the identity of the feature required
- the number of units of the feature required
- the application key
- a timestamp value.

Referring to Figure 2, when the LM receives this request message, it checks that the timestamp value is current. Assuming the timestamp value is current, the LM then checks whether there is a license key in the license table for the required feature.

If there is a license key in the table, the LM then checks whether the expiry date of the license has passed, and checks the signature of the license key to ensure that it has not been modified. The LM also checks whether the required number of units are available for the feature (ie whether the number of requested units plus the number of units already granted is less than or equal to the limit value in the license key).

If all these checks are satisfactory, the LM returns a "license granted" message to the application, sealed under the application key. The LM keeps a record of the number of units granted for each feature. If, on the other hand, any of the checks fails, the LM returns a "license denied" message to the application. The LM also writes a record in the log 24 to indicate whether a license has been granted or denied.

If the application receives a "license granted" message, it proceeds to use the requested features as required. If, on the other hand, it receives a "license denied" message, it performs one of the following actions, as determined by the designer of the application:

- the application may simply shut itself down.
- in the case where the license was denied because there were not enough units of the requested feature available, the application may display a "call again later" message to the user.
- the application may continue running in a reduced service mode eg a demonstration mode.

When an application terminates, it sends a "license relinquish" message to the LM. The LM will then withdraw any licenses issued to this application, making the units available to other applications.

Each application is required to send a revalidation message periodically to the LM, to re-validate its license. For example, a revalidation message may be required every 5 minutes. If the application does not receive any response to this message, it assumes that it has lost contact with the LM, and shuts down or continues in a reduced service mode.

The LM periodically checks whether it has received revalidation messages from all the application to which it has granted licenses. If a revalidation message has not been received from an application, the LM assumes that the application has failed, and therefore withdraws the license, making the units available to other applications.

In order to ensure that unauthorised copies of the LM cannot be run on other systems, it is necessary to provide a way of identifying the machine on which the LM runs. This is achieved by means of a security identification device (SID) 26, which stores an identifier unique to this device, referred to as the secure host identifier. The SID is attached to the computer 10 by way of an external port 28. In this example, the port is a standard parallel printer port, and the SID is designed so that a printer may be plugged into the back of the SID, so that both the printer and SID share the same port. Messages for the SID are identified by special commands.

In other embodiments of the invention, the SID may be attached to a special dedicated port, or to some other type of standard port. The port may be serial rather than parallel.

Referring to Figure 3, in order to check the host identity, the LM sends a request message to the SID at regular intervals, requesting it to supply the secure host identifier.

The SID responds to this by returning a message encrypted under a key known only to the SID and the LM.

The message contains:

- the secure host identifier
- a sequence number, which is incremented each time the SID returns a message.

When the LM receives this message, it decrypts it, and checks the sequence number to ensure that it is the next expected sequential value. This ensures that it is not possible to replace the SID by a program which intercepts the requests from the LM and returns a copy of the SID's response, or which passes the request to a SID on another system.

The LM then checks whether the returned secure host identifier matches the machine identifiers of the license keys held in the license table 22.

If the LM does not receive any response to a request to the SID, or if the response does not contain the correct sequence number, or if the secure host identifier does not match the machine identifiers in the license keys, the LM closes down. This means that the LM will not issue any more licenses to applications. Also, because the LM will not now respond to the revalidation message from the application, any outstanding licenses are effectively cancelled.

In summary, it can be seen that the LM will issue licenses, permitting applications to operate, only if a security identification device SID is connected to the computer, and if the machine identifiers in the individual license keys issued to the LM match the secure host identifier held in the SID.

It should be noted that the LM can grant licenses to applications running in any of the computers 10 in the network, not just to applications running in the same computer as the LM. The number of licenses that may be granted is restricted by the limit in the license keys. Thus, for example, if a license key sets a limit on the number of users, then the total number of users of a particular application in the network cannot exceed this limit.

The use of the device for the provision of the identifier to the license manager has several very important advantages:

- if the machine to which the device is attached fails, the device can be transferred to another machine (new keys are not required)
- the supplier of the device can retain title to the device, so in the event of the machine being sold the device has to be returned to the supplier. Hence all software on the machine that would only work with a license manager will no longer function as required by the terms of supply of the software which is licensed to a legal entity not to a machine.
- if the user of the software wishes to change the license he has to reduce its capability, the device can be replaced and new keys issued. Current schemes do not provide for the secure revocation of the keys.
- the device can be used to provide secure identification on standard hardware platforms which do not inherently provide such a facility, and hence can enable the use of license management on such hardware.

It should be noted that although the embodiment of the invention described above is a multi-computer system, the invention is equally applicable to single processor systems, or to multi-nodal systems, comprising a plurality of multi-processor nodes.

Claims

1. A computer system including a license manager for regulating usage of software items in accordance with license keys issued to the license manager, the license manager being arranged to check the host identity of the computer on which it runs and to permit usage only if the host identity matches an identity value in the license keys, characterised in that the host identity of the computer is supplied by a security identification device removably coupled to an external port on the computer.

2. A system according to Claim 1 wherein communication of the host identity between the security identifier device and the license manager is protected by encryption.

3. A system according to Claim 2 wherein each host identity returned by the security identifier device is encrypted together with a sequence number which is incremented each time the host identity is returned.

4. A system according to any preceding claim wherein the license manager regulates the usage of software items within a domain comprising software items that can communicate with the license manager.

5. A system according to Claim 4 wherein said domain is distributed over a network of computers.

Fig.1.

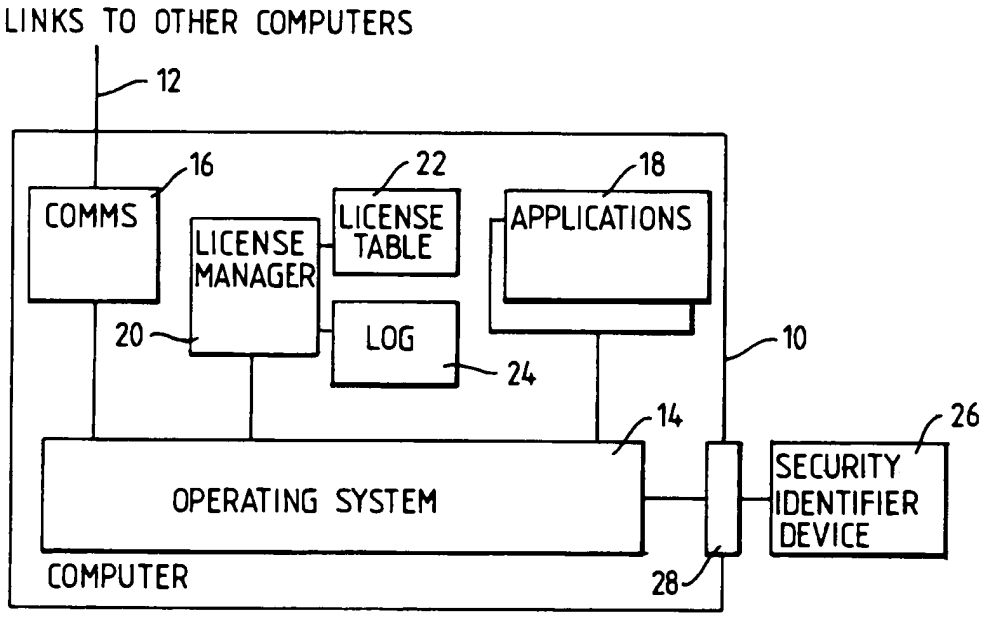


Fig. 2.

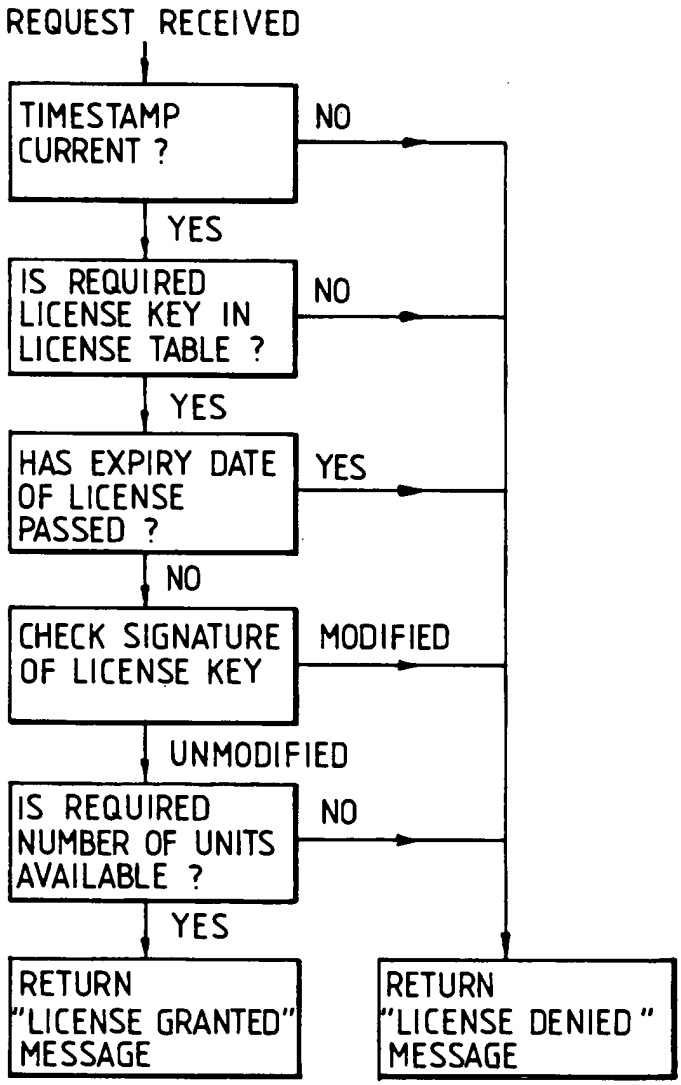
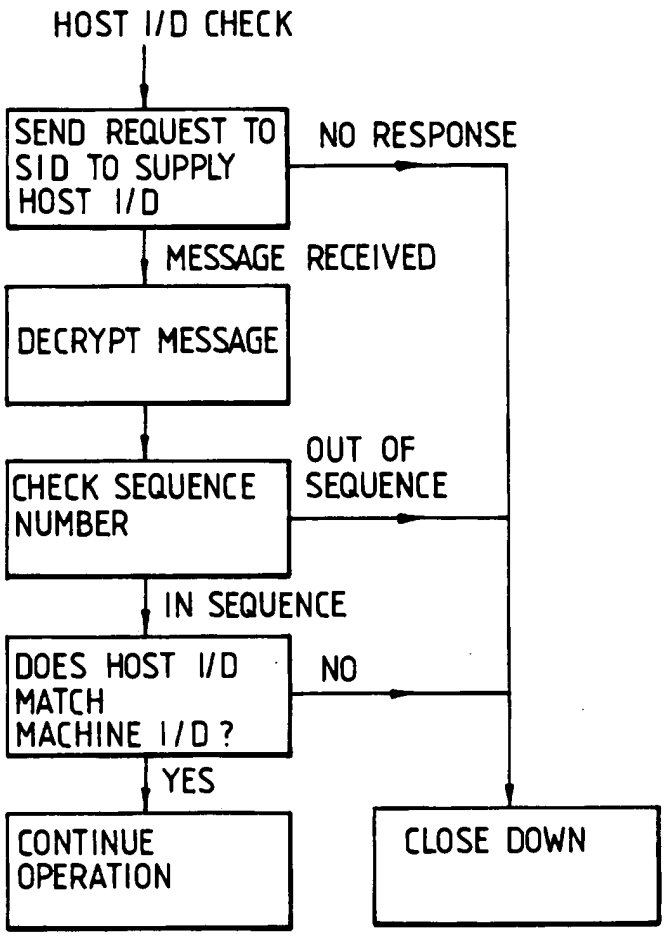


Fig.3.



European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 93 30 6468

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.5)
Y	US-A-4 924 378 (HERSHEY ET AL) * abstract; figures 1,3,5,7 * * column 1, paragraph 2 * * column 2, line 1 - column 3, line 36 * * column 7, line 22 - column 8, line 12 * * column 10, line 27 - line 40 * * claims 1-5,11-23 *	1-5	G06F1/00
Y	PTR PHILIPS TELECOMMUNICATION AND DATA SYSTEMS REVIEW, vol. 47, no. 3 , September 1989 , HILVERSUM, NL; pages 1 - 19 R.C.FERREIRA 'The Smart Card: A High Security Tool in EDP' * summary; figures 4,5 * * page 5, line 6 - page 7, line 5 * * page 9, line 1 - page 11, line 40 * * page 12, line 36 - page 13, line 4 *	1-5	
A	EP-A-0 191 162 (IBM) * abstract; figures 4,9 * * column 6, line 8 - column 7, line 14 * * column 9, line 6 - line 39 * * column 10, line 5 - line 40 * * column 13, line 5 - line 36 *	1,3	TECHNICAL FIELDS SEARCHED (Int.Cl.5) G06F

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	4 May 1994	Powell, D

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 678 836 A1**

(12) EUROPEAN PATENT APPLICATION

(21) Application number: **94105573.3**

(51) Int. Cl.[6]: **G07F 7/10**

(22) Date of filing: **11.04.94**

(43) Date of publication of application:
25.10.95 Bulletin 95/43

(84) Designated Contracting States:
DE FR GB

(71) Applicant: **TANDEM COMPUTERS INCORPORATED**
10435 North Tantau Avenue,
Loc. 200-16
Cupertino,
California 95014-0709 (US)

(72) Inventor: **Atalla, Martin M.**
18 Monte Vista
Atherton
CA 94025 (US)
Inventor: **Hopkins, W. Dale**
2425 Rio Drive
Gilroy
CA 95020 (US)

(74) Representative: **KUHNEN, WACKER & PARTNER**
Alois-Steinecker-Strasse 22
D-85354 Freising (DE)

(54) Method and means for combining and managing personal verification and message authentication encryptions for network transmission.

(57) The method and means of transmitting a user's transaction message to a destination node in a computer-secured network operates on the message, and a sequence number that is unique to the transaction message to form a message authentication code in combination with the user's personal identification number. The message authentication code is encrypted with a generated random number and a single session encryption key which also encrypts the user's personal identification number. An intermediate node may receive the encryptions to reproduce the personal identification number that is then used to encrypt the received message and sequence number to produce the random number and a message authentication code for comparison with a decrypted message authentication code. Upon favorable comparison, the random number and the message authentication code are encrypted with a second session encryption key to produce an output code that is transmitted to the destination node along with an encrypted personal identification number. There, the received encryptions are decrypted using the second session key to provide the personal identification number for use in encrypting the message and sequence number to produce a message authentication code for comparison with a decrypted message authentication code. Upon favorable comparison, the transaction is completed and a selected portion of the decrypted random number is returned to the originating node for comparison with the corresponding portion of the random number that was generated there. Upon unfavorable comparison at the destination node or at an intermediate node, a different portion of the decrypted random number is returned to the originating node for comparison with the corresponding portion of the random number that was generated there. The comparisons at the originating node provide an unambiguous indication of the completion or non-completion of the transaction at the destination node.

EP 0 678 836 A1

Related Cases

The subject matter of this application is related to the subject matter disclosed in U.S. Patents 4,268,715; 4,281,215; 4,283,599; 4,288,659; 4,315,101; 4,357,529; 4,536,647 and pending application for U.S. Patent Serial No. 547,207, entitled POCKET TERMING, METHOD AND SYSTEM FOR SECURED BANKING TRANSACTIONS, filed October 31, 1983 by M.M. Atalla.

Background of the Invention

Conventional data encryption networks commonly encrypt a Personal Identification Number with a particular encryption key for transmission along with data messages, sequence numbers, and the like, from one location node in the data network to the next location or node in the network. There, the encrypted PIN is decrypted using the encryption key, and re-encrypted with another encryption key for transmission to the next node in the network, and so on to the final node destination in the network.

In addition, such conventional data encryption networks also develop a Message Authentication Codes in various ways, and then encrypt such MAC for transmission to the next node using a MAC-encryption key that is different from the encryption key used to encrypt the PIN. At such next node, the MAC is decrypted using the MAC encryption key and then re-encrypted using a new MAC-encryption key for transmission to the next node, and so on to the final destination node in the network.

Further, such conventional networks operate upon the PIN, MAC, data message, sequence number, and the like; received and decrypted at the final destination node to consummate a transaction, or not, and then communicate an ACKnowledgment or Non-ACKnowledgment message back to the originating node of the network. Such ACK or NACK codes may be encrypted and decrypted in the course of transmission node by node through the network back to the originating node to provide an indication there of the status of the intended transaction at the final destination node.

Conventional data encryption networks of this type are impeded from handling greater volumes of messages from end to end by the requirement for separately encrypting and decrypting the PIN and MAC codes at each node using different encryption/decryption keys for each, and by the requirement for encrypting/decrypting at least the ACK code at each node along the return path in the network.

In addition, such conventional data encryption networks are susceptible to unauthorized intrusion and compromise of the security and message authenticity from node to node because of the separated PIN and MAC encryption/decryption techniques involved. For example, the encrypted PIN is vulnerable to being "stripped" away from the associated MAC, message, sequence number, and the like, and to being appended to a different MAC, message, sequence number, and the like, for faithful transmission over the network. Further, the return acknowledgment code may be intercepted and readily converted to a non-acknowledgment code or simply be altered in transmission after the transaction was completed at the destination node. Such a return code condition could, for example, cause the user to suffer the debiting of his account and, at the same time, the denial of completion of a credit purchase at point-of-sale terminal or other originating node.

Summary of the Invention

Accordingly, the method and means for integrating the encryption keys associated with the PIN and MAC codes according to the present invention assure that these codes are sufficiently interrelated and that alteration of one such code will adversely affect the other such code and inhibit message authentication in the network. In addition, the return acknowledgment or non-acknowledgment code may be securely returned from node to node in the network without the need for encryption and decryption at each node, and will still be securely available for proper validation as received at the originating node. This is accomplished according to the present invention by using one session key to encrypt the PIN along with the MAC, a random number, the message, and the sequence number which are also encrypted with the PIN such that re-encryption thereof in the transmission from location to location, or node to node over a network is greatly facilitated and validatable at each node, if desired. In addition, portions of the random number are selected for use as the Acknowledgment or Non-Acknowledgment return codes which can be securely returned and which can then only be used once to unambiguously validate the returned code only at the originating node in the network.

Description of the Drawings

Figure 1 is graphic representation of a typical conventional encryption scheme which operates with two independent session keys;

Figure 2 is a schematic representation of a second network according to the present inventions; and

Figure 3 is a graphic representation of the signal processing involved in the operation of the net-

work of Figure 2.

Description of the Preferred Embodiment

Referring now to Figure 1, there is shown a graphic representation of the encoding scheme commonly used to produce the PIN and MAC codes using two session keys for transmission separately to the next network node. As illustrated, one session key 5 may be used to encrypt the PIN entered 7 by a user (plus a block of filler bits such as the account number, as desired) in a conventional encryption module 9 which may operate according to the Data Encryption Standard (DES) established by the American National Standards Institute (ANSI) to produce the encrypted PIN signal 11 (commonly referred to as the PIN block" according to ANSI standard 9.3) for transmission to the next network node. In addition, the message or transaction data which is entered 13 by the user and which is to be transmitted to another node, is combined with a sequence number 15 that may comprise the date, time, station code, and the like, for encryption by a DES encryption module 17 with another session key 19 to produce a Message Authentication Code (MAC) 21 for that message and sequence number. The MAC may comprise only a selected number of significant bits of the encrypted code. The message and MAC are separately transmitted to the next node along with the encrypted PIN, and these codes are separately decrypted with the respective session keys and then re-encrypted with new separate session keys for transmission to the next network node, and so on, to the destination node. Conventional PIN validation at the destination node, and message authentication procedures may be performed on the received, encrypted PIN and MAC, (not illustrated) and the message is then acted upon to complete a transaction if the PIN is valid and the MAC is unaltered. A return ACKnowledgment (or Non-ACKnowledgment) code may be encrypted and returned to the next node in the network over the return path to the originating node. At each node in the return path, the ACK code is commonly decrypted and re-encrypted for transmission to the next node in the return path, and so on (not illustrated), to the originating node where receipt of the ACK is an indication that the transaction was completed at the destination node. Conventional systems with operating characteristics similar to those described above are more fully described, for example, in U.S. Patent 4,283,599.

One disadvantage associated with such conventional systems is the need to encrypt and decrypt at each node using two separate session keys. Another disadvantage is that such conventional systems are vulnerable to unauthorized manipulation at a network node by which the message and MAC may be "stripped away" from the encrypted PIN associated with such message and replaced with a new message and MAC for transmission with the same encrypted PIN to the next network node. Further, the acknowledgement code that is to be returned to the originating node not only must be decrypted and re-encrypted at each node along the return path, but the return of an acknowledgment code that is altered along the return path may connote non-acknowledgment or non-completion of the intended transaction at the destination node. This condition can result In the account of the user being debited (the PIN and MAC were valid and authentic as received at the destination node), but the user being denied completion of a credit transaction (e.g., transfer of goods) at the originating node.

Referring now to Figures 2 and 3, there are shown schematic and graphic representations, respectively, of network operations according to the present invention. Specifically, there is shown a system for transmitting a message over a network 29 from an originating node 31 to a destination node 33 via an intermediate node 35. At the originating node 31, an authorized user enters his PIN 37 of arbitrary bit length with the aid of a key board, or card reader, or the like, and the entered PIN is then filled or blocked 39 with additional data bits (such as the user's account number in accordance with ANSI standard 9.3) to configure a PIN of standard bit length.

In addition, the transaction data or message 41 entered through a keyboard, or the like, by the user is combined with a sequence number 43 which is generated to include date, time of day, and the like. The combined message and sequence number is encrypted 45 with the PIN (or blocked PIN) in a conventional DES module to produce a multi-bit encrypted output having selected fields of bits, one field of which 51 serves as the Message Authentication Code (MAC). Other schemes may also be used to produce a MAC, provided the PIN (or blocked PIN) is used as the encryption key, and the resulting MAC, typically of 64-bit length, may be segregated into several sectors or fields 51. A random number (R/N) is generated 52 by conventional means and is segregated into several sectors or fields 54, 56, 58. The first sector or field 54 of, say 32-bits length, is then encrypted with the selected MAC field 53 in a conventional DES encryption module 55 (or in DES module 45 in time share operation) using the session key K_1 as the encryption key 50. In addition, the PIN (or blocked PIN) 39 is encrypted in DES encryption module 60 (or in DES module 45 in time share operation) using the session key K1 as the encryption Key 50. The session key 50 may be transmitted to successive

nodes 35, 33 in secured manner, for example, as disclosed in U.S. Patent 4,288,659. The resulting encrypted output codes 62, 64 are then transmitted along with sequence number 43 and the message 41 (in clear or cypher text) over the network 29 to the next node 35 in the path toward the destination node 33. Thus, only a single session key K_1 is used to encrypt the requisite data for transmission over the network, and the residual sectors or fields 56, 58 of the random number from generator 52 remain available to verify successful completion of the transaction at the destination node 33, as later described herein.

At the intermediate node 35, the encrypted PIN 64 received from the originating node 31 is decrypted in conventional DES module 70 using the session key K_1 to produce the blocked PIN 63. In addition, the encrypted MAC and R/N 68 received from the originating node is decrypted in conventional DES module 61 (or in DES module 70 operating in timeshare relationship) using session key K_1 to produce the MAC and the R/N in segregated fields. An initial validation may be performed by encrypting the received message 41 and sequence number 43 in conventional DES module 67 using the decrypted PIN 63 as the encryption key. Of course, the original PIN as entered by the user may be extracted from the decrypted, blocked PIN 63 to use as the encryption key in module 67 if the corresponding scheme was used in node 31. (It should be understood that the PIN or blocked PIN does not appear in clear text outside of such decryption or encryption modules 70, 67 (or 69, later described herein), and that these modules may be the same DES module operated in time-shared relationship.)

The encrypted output of module 67 includes several sectors, or fields, similar to those previously described in connection with the encrypted output of module 45. The selected sector 53 of significant bits that constitutes the MAC is selected for comparison with the MAC 65 that is decrypted in DES module 61. This decryption also provides the R/N having several selected sectors or fields 72. If the comparison of the decrypted and encrypted MAC's in comparator 74 is favorable, gate 76 is enabled and the decrypted MAC and R/N are encrypted in conventional DES module 69 using new session key K_2 as the encryption key, and gate 88 is enabled to encrypt the decrypted PIN in DES module 78 (or in DES module 67 or 69 in time share operating). If comparison is unfavorable, the transaction may be aborted and the gate 80 is enabled to transmit back to the originating node 31 the sector or field 58 of the R/N which constitutes the Non ACKnowledge sector of the decrypted R/N output of module 61. The encrypted PIN output 82 of module 78 and the encrypted MAC and R/N output 84 of the module 69 are thus transmitted along with the message 41 and sequence number 43 over the network 29 to the destination node 35 upon favorable comparison 74 of the encrypted and decrypted MACs.

At the destination node 33, the encrypted PIN output 86 received from the intermediate node 35 is decrypted in conventional DES module 71 using the session key K_2 to produce the PIN 73. An initial validation may be performed by encrypting the received message 41 and sequence number 43 in conventional DES module 77, using the decrypted PIN 73 as the encryption key. As was described in connection with the intermediate node 35, the original PIN as entered by the user may be extracted from the decrypted, blocked PIN 73 to use as the encryption Key in module 77 if the corresponding scheme was used in node 31. And, it should be understood that the PIN or blocked PIN does not appear in clear text outside of the decryption or encryption modules 71, 77, which modules may be the same DES module operated in time-shared relationship. In addition, the encrypted MAC and R/N received at the destination node 33 is decrypted in DES module 92 using the session key K_2 to produce the MAC 75 and the R/N 94 in segregated sectors or fields. The selected sector 53 of significant bits that constitutes the MAC in the encrypted output of module 77 is compared 79 for parity with the decrypted MAC 75. If comparison is favorable, the transaction may be completed in response to the message 41, and gate 81 may be enabled to transmit 29 back to the intermediate node 35 a second selected sector or field 56 which constitutes the ACKnowledge output sector of the R/N decrypted output from module 92. If comparison 79 is unfavorable, the transaction is not completed and gate 83 is enabled to transmit 29 back to the intermediate node 35 a third selected sector or field 58 which constitutes the Non-ACKnowledge sector of the R/N decrypted output from module 92.

In accordance with one aspect of the present invention, the returned ACK or NACK codes do not require decryption and re-encryption when transmitted from node to node along the return path in the network back to the originating node 31. Instead, these codes are already in encoded form and may be transmitted directly from node to node without encumbering a node with additional operational overhead. These codes are therefore secured in transmission over the network and are only cypherable in the originating node 31 which contains the ACK and NACK fields or sectors 56 and 58 of the random number from generator 52. At the originating node 31, the second and third sectors or fields 56 and 58 of the random number are compared 98 with the corresponding sectors of

decrypted R/N outputs received from the destination node 33 (or the sector 58 of the decrypted R/N output received from intermediate node 35) to provide an indication at the originating node that the transaction was either completed 89 or aborted 91. Of course, the ACK and NACK may be encrypted as a network option when returned to the originating node 31. And, it should be understood that the encryption and decryption modules at each node may be the same conventional DES module operated in timeshare relationship.

Therefore, the system and method of combining the management of PIN and MAC codes and the session keys associated therewith from node to node along a data communication network obviates the conventional need for separate session keys for the PIN and the MAC, and also obviates the need for conventional encryption/decryption schemes for an acknowledgment code at each node along the return path back to the originating node. If desired, PIN validations may be performed at each node since the PIN is available within the DES module circuitry. In addition, the present system and method also reduces the vulnerability of a secured transmission system to unauthorized separation of a valid PIN code from its associated message and MAC code for unauthorized attachment to a different message and MAC code. Further, the method and means of the present invention reduces the ambiguity associated with the return or not of only an acknowledgment code in conventional systems by returning either one of the ACK and NACK codes without additional operational overhead at each node.

Claims

1. The method of securing transaction data between two locations in response to a user's message ad personal identification number, the method comprising:

forming a sequence number representative of the user's transaction;

encoding in a first logical combination at the first location the user's message and the sequence number in accordance with the personal identification number received from the user to produce a message authentication code having a plural number of digit sectors;

generating a random number;

establishing a first encoding key;

encoding in a second logical combination at the first location the random number and a selected number of sectors of the message authentication code in accordance with the first encryption key to produce a first coded output;

encoding in a third logical combination at the first location the user's personal identification number in accordance with the first encoding key to produce a second coded output;

transmitting to another location the user's message and the sequence number and the first ad second coded outputs;

establishing the first encoding key at such other location;

decoding the first coded output received at such other location with the first encoding key according to said second logical combination thereof to provide the random number and message authentication code;

decoding the second coded output received at such other location with the first encoding key according to said third logical combination to provide the user's personal identification number;

encoding in the first logical combination at such other location the user's message and sequence number received thereat in accordance with the decoded personal identification number to produce a message authentication code having a plural number of digit sectors; and

comparing selected corresponding digit sectors of the decoded message authentication code and the encoded message authentication code to provide an indication upon favorable comparison of the valid transmission of the user's message between the two locations.

2. The method according to claim 1 comprising the steps of:

establishing a second encoding key at the other location;

encoding in a fourth logical combination at such other location the decoded random number and selected sector of the message authentication code in accordance with the second encoding key to produce a third coded output;

encoding in a fifth logical combination at the other location the decoded user's personal identification number in accordance with the second encoding key to produce a fourth coded output;

transmitting to a remote location the user's message and the sequence number and the third and fourth coded outputs;

establishing the second encoding key at the remote location;

decoding the third coded output as received at the remote location according to the fourth logical combination in accordance with the second encoding key to provide the random number and the message authentication code having a plural number of digit sectors;

decoding the fourth coded output received

at the remote location according to the fifth logical combination to provide the user's personal identification number;

encoding the message and the sequence number received at the remote location according to the first logical combination in accordance with the decoded personal identification number to produce a message authentication code having a plural number of digit sectors; and

comparing corresponding digit sectors of the decoded message authentication code and the encoded message authentication code at the remote location to provide an indication upon favorable comparison of the unaltered transmission of the message, or an indication upon unfavorable comparison of an alteration in the transmission of the message.

3. The method according to claim 1 comprising the steps of:

transmitting a selected sector of the decoded random number from the other location to the one location in response to unfavorable comparison; and

comparing the selected sector of the random number received at the one location from the other location with the corresponding selected sector at the one location to provide an indication of the altered transmission of the message to the other location.

4. The method according to claim 2 comprising the steps of:

completing the transaction and returning a second selected sector of the decoded random number from the remote location to the one location in response to said favorable comparison, and inhibiting completion of the transaction and returning a third selected sector of the decoded random number from the remote location to the one location in response to said unfavorable comparison; and

comparing the selected sector of the random number received at the one location from the remote location with the corresponding selected sector of the number generated at the one location to provide an indication of the completion or non-completion of the transaction at the remote location.

5. Apparatus for securing transaction data between two locations in response to a user's message and personal identification number, the apparatus comprising:

means for generating a sequence number associated with a user's transaction;

means for generating a random number;

first encryption means at one location for encrypting according to a first logical combination of the user's message and the sequence number applied thereto with the personal identification number received from the user for producing a message authentication code therefrom having a plural number of digit sectors;

means at said one location for producing a first session key;

second encryption means coupled to receive the random number from the user and a selected sector of the message identification code for encrypting the same with the first session key according to a second logical combination thereof to produce a first encoded output;

third encryption means coupled to receive the personal identification number from the user for encrypting the same with the first session key according to a third logical combination thereof to produce a second encoded output;

means for transmitting the first and second encoded outputs and message and sequence number from the one location to the next location;

means at the next location for producing the first session key;

first decryption means at the next location coupled to receive the transmitted first encoded output and the first session key for decrypting in accordance with said second logical combination to provide the random number and the message authentication code;

second decryption means at the next location coupled to receive the transmitted second encoded output and the first session key for decrypting in accordance with the third logical combination thereof to produce the user's personal identification number;

third encrpytion means at the next location coupled to receive the transmitted message and sequence number for encoding the same according to said first logical combination with the decrypted personal identification number to produce a message authentication code having a plural number of digit sectors;

comparison means at the next location coupled to receive the corresponding selected sectors of the decrypted message authentication code and of the encrypted message authentication code for producing an output indication of the parity thereof; and

means at the next location responsive to said output indication for operating upon the received message in response to favorable comparison.

6. Apparatus as in claim 5 comprising:

means at the next location responsive to the unfavorable comparison for transmitting to the one location a selected sector of the random number.

7. Apparatus as in claim 5 comprising:

means at the next location for producing a second encoding key;

first encryption means at the next location coupled to receive the decrypted message authentication code and random number for encoding the same with the second encoding key in accordance with a fourth logical combination in response to said favorable comparison for producing a third output code for transmission to a destination location;

second encryption means at the next location coupled to receive the decrypted personal identification number for encoding the same with the second encoding key in accordance with a fifth logical combination in response to said favorable comparison for producing a fourth output code for transmission to a destination location;

means at the destination location for producing the second encoding key;

first decryption means at the destination location for receiving the third output code transmitted from said next location and the second encoding key for decoding the same according to said fourth logical combination to provide the random number and the message authentication code;

second decryption means at the destination location for receiving the fourth output code transmitted from said next location and the second encoding key for decoding the same according to said fifth logical combination to provide the personal identificaton number;

encryption means at the destination location for receiving the message and the sequence number for encoding the same with the decrypted personal identification number in accordance with the first logical combination to produce a message authentication code having a plural number of digit sectors;

means at the destination location for comparing corresponding selected sectors of the encrypted message authentication code and the decrypted message authentication code to produce output indications of favorable and unfavorable comparisons;

means at the destination location responsive to favorable output indication for operating upon the transmitted message and for transmitting a selected sector of the random number to said one location, and responsive to unfavorable comparison for transmitting another selected sector of the random number to said one location; and

comparator means at the one location coupled to receive the corresponding selected sectors of the random number for providing an output indication of the status of operation upon the message at the destination location.

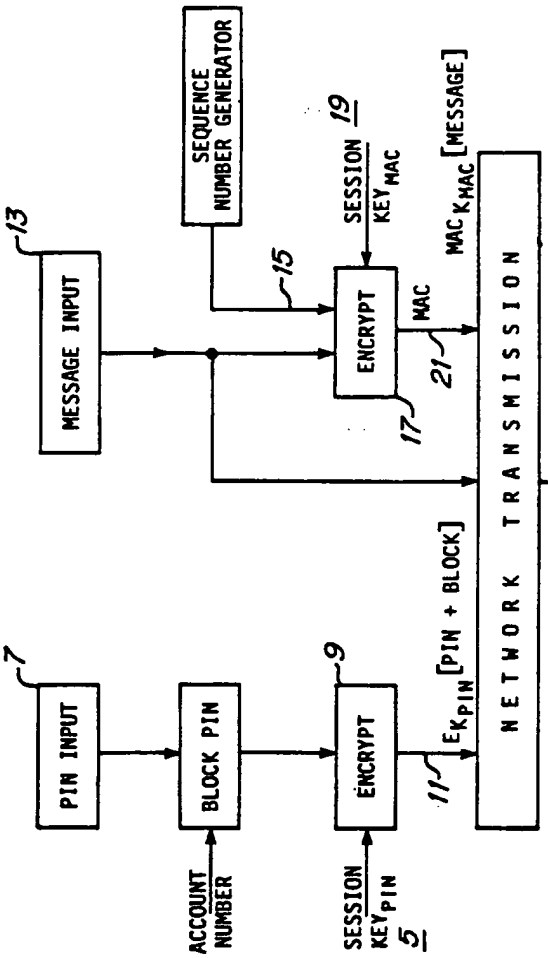


Figure 1
(PRIOR ART)

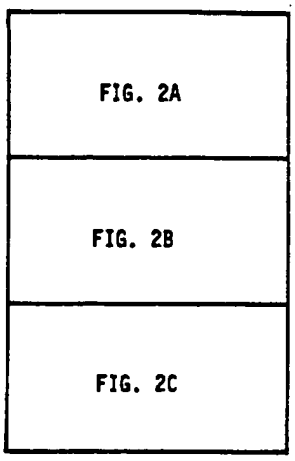


Figure 2

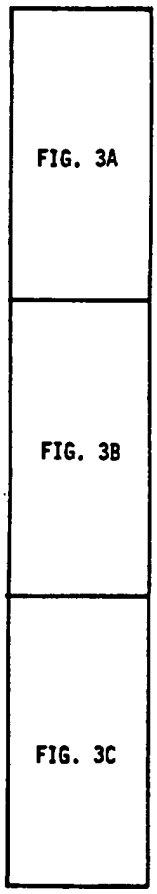


Figure 3

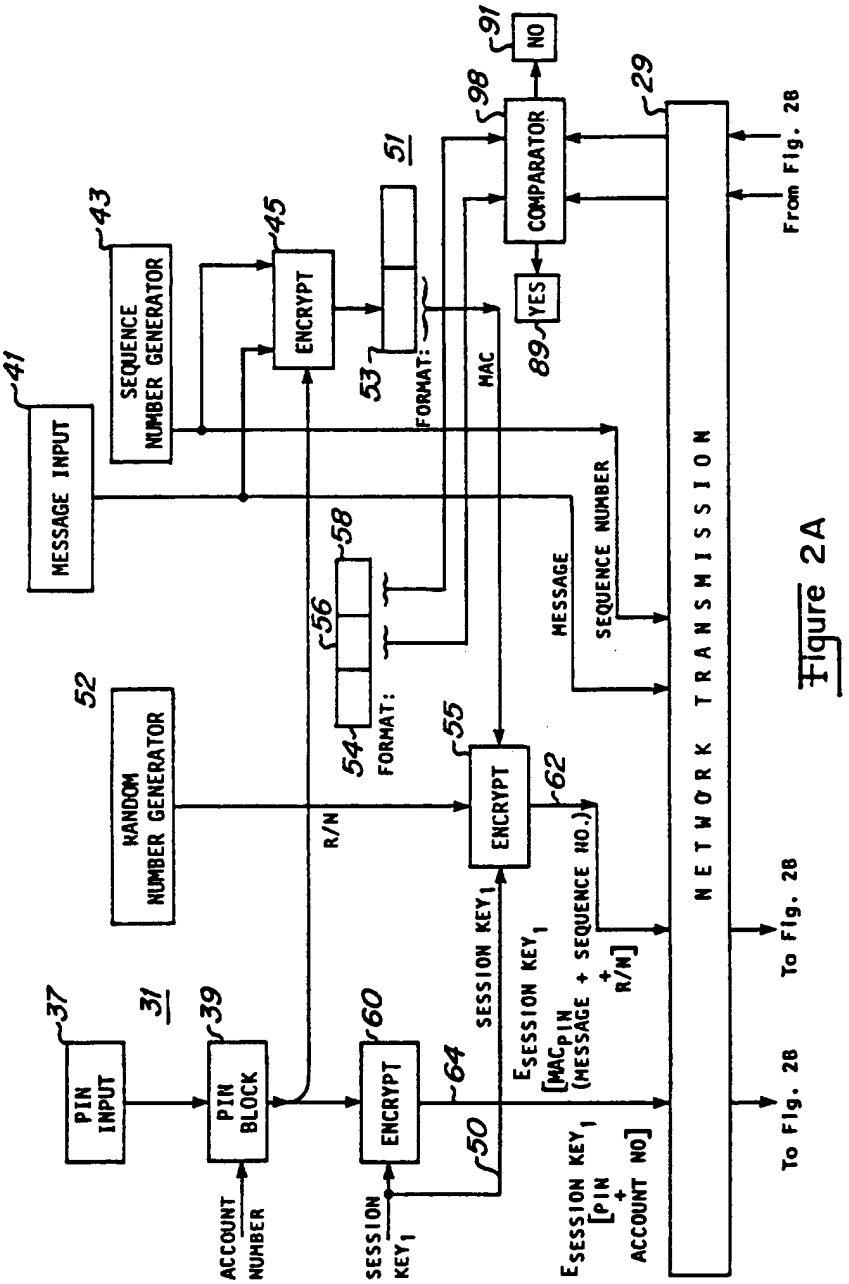


Figure 2A

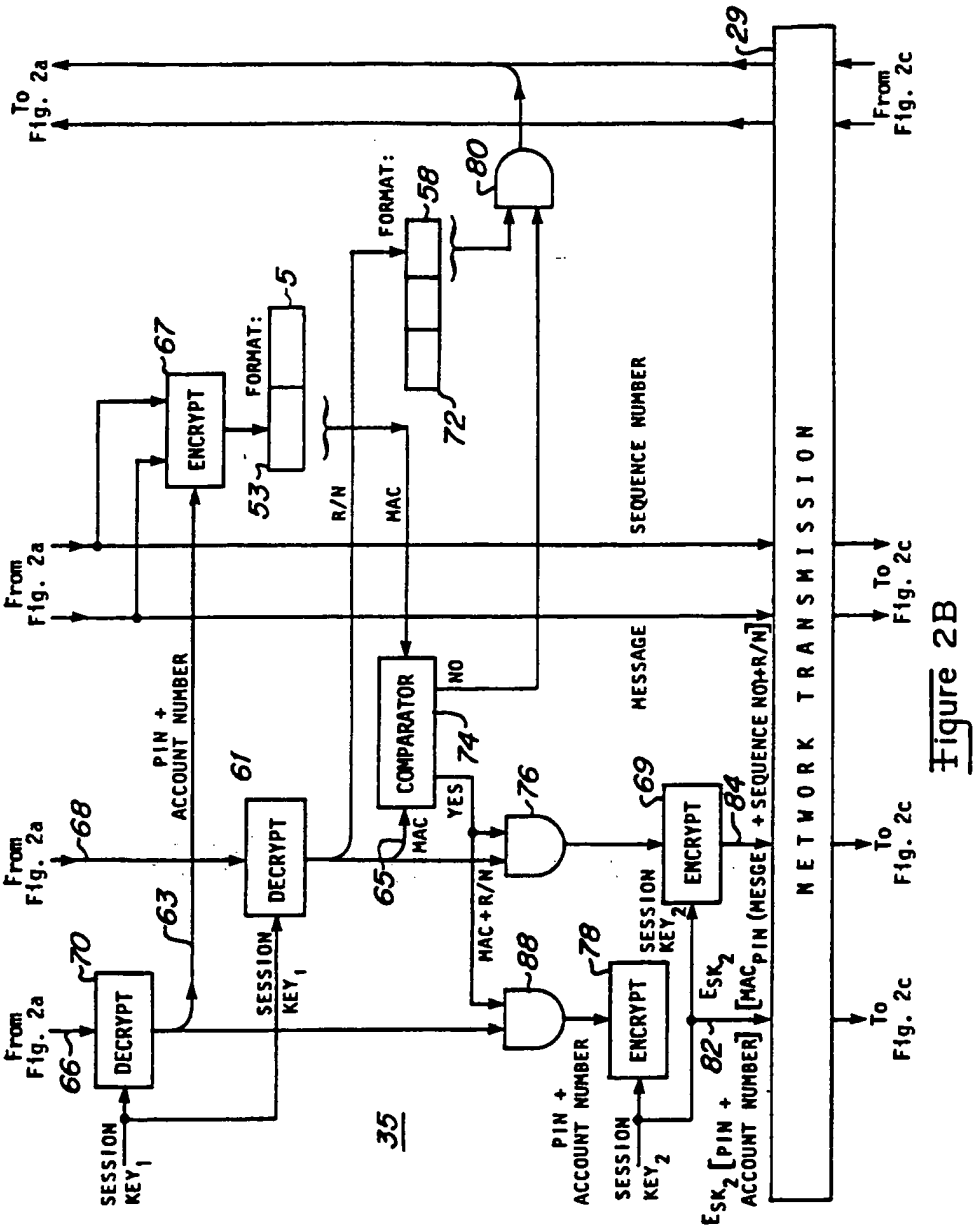


Figure 2B

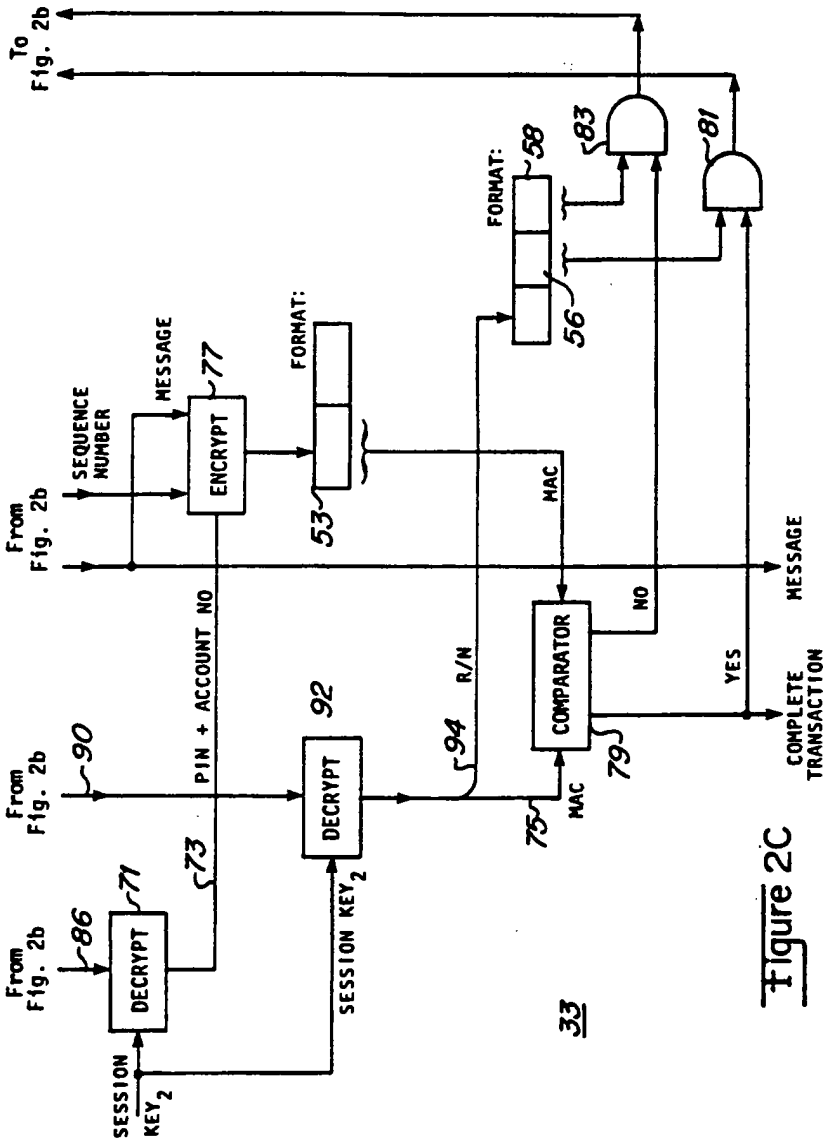


Figure 2C

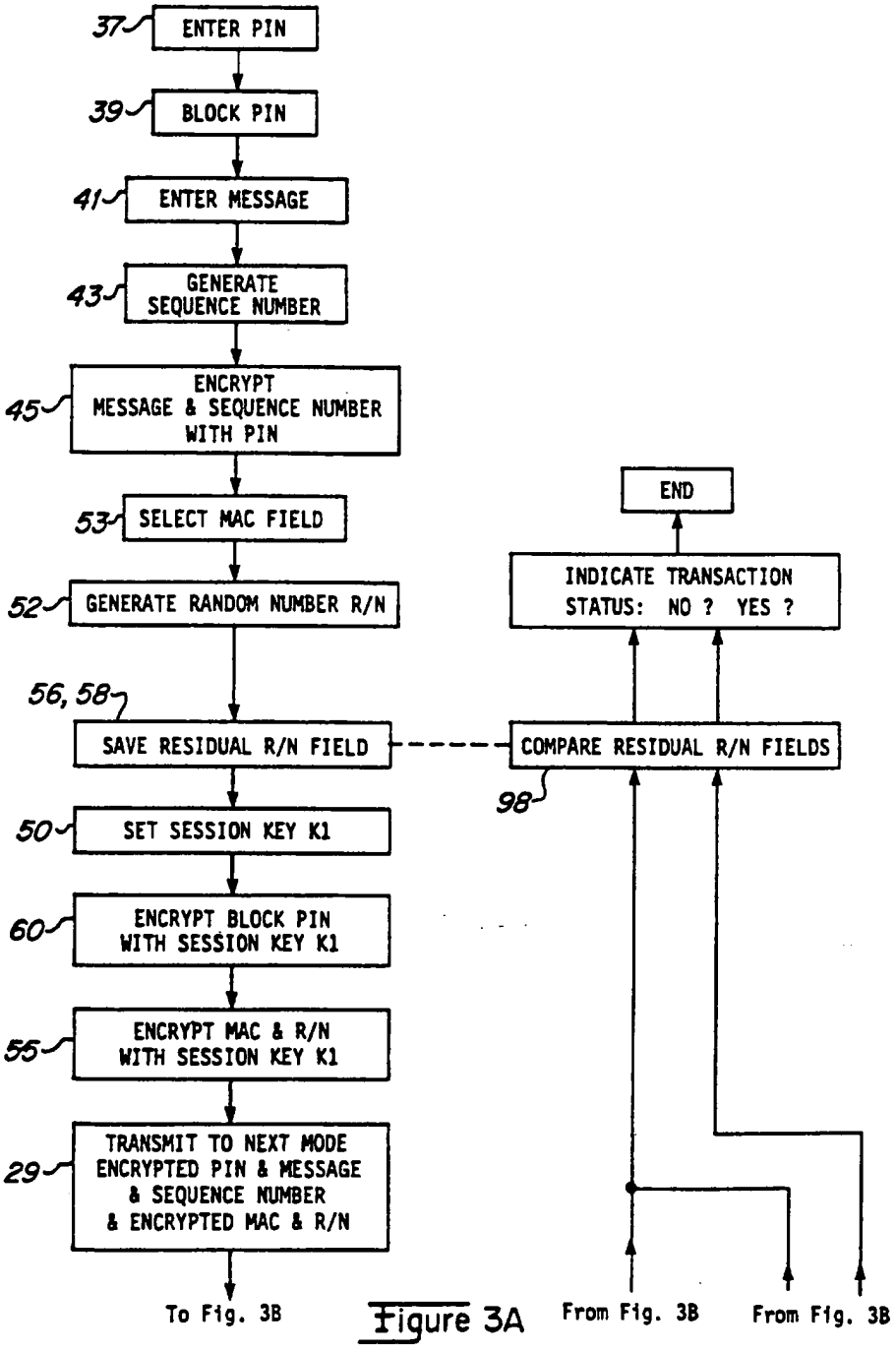


Figure 3A

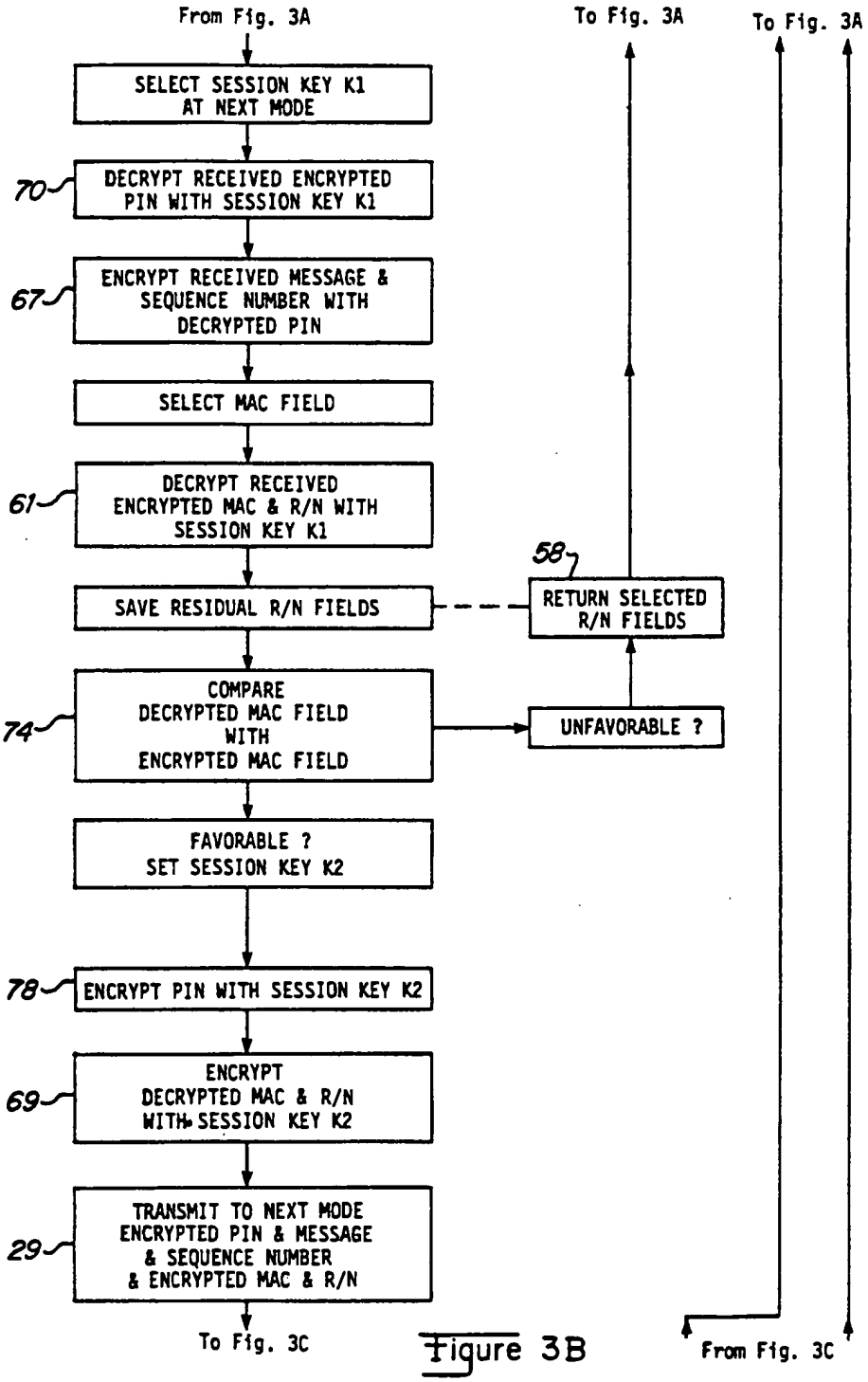


Figure 3B

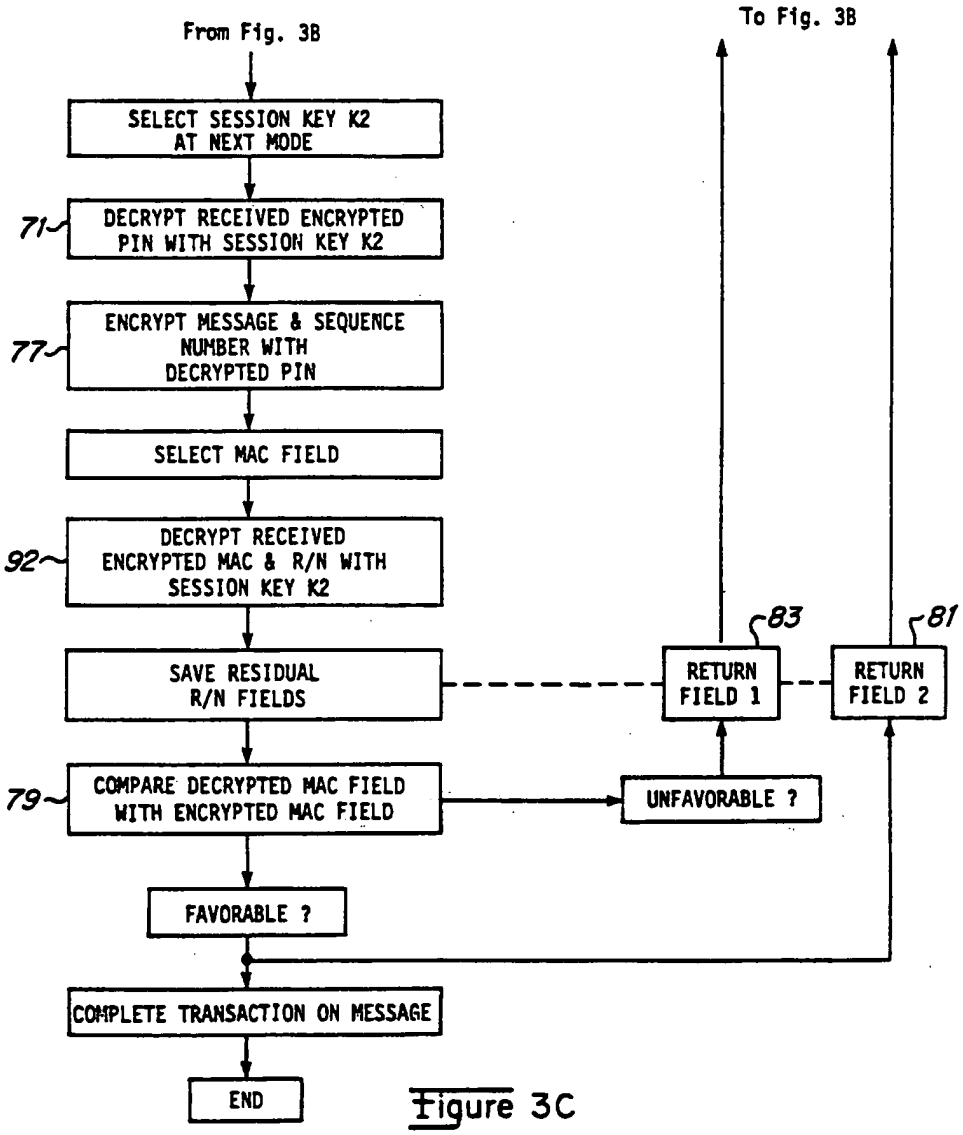


Figure 3C

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 94 10 5573

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
X A	EP-A-0 391 261 (NIPPON TELEGRAPH) * abstract * * page 2, line 19 - line 31 * * page 4, line 31 - page 5, line 12 * * page 6, line 21 - line 25 * * page 7, line 2 - line 11 * * page 9, line 33 - line 54 * * page 16, line 41 - page 17, line 32 * * claim 1; figures 2A,2B *	1 2,5	G07F7/10
X A	US-A-5 101 373 (KATSUAKI) * column 5, line 32 - line 59 * * claims 1,4,5 *	1 2,3,5	
A	US-A-5 016 277 (HAMILTON) * column 16, line 60 - column 17, line 7 *	1,5	
A	EP-A-0 547 975 (BULL CP8) * abstract *	1,5	
A	EP-A-0 500 245 (TOSHIBA) * abstract * * claim 1 *	1,5	TECHNICAL FIELDS SEARCHED (Int.Cl.6) G07F H04L
A	EP-A-0 494 796 (NCR CORPORATION) * abstract *	1,5	

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	14 September 1994	Taccoen, J-F

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 679 977 A1**

(12) EUROPEAN PATENT APPLICATION

(21) Application number: **95105400.6**

(51) Int. Cl.[6]: **G06F 1/00**, G06F 12/14

(22) Date of filing: **10.04.95**

(30) Priority: **25.04.94 US 238418**

(43) Date of publication of application:
02.11.95 Bulletin 95/44

(84) Designated Contracting States:
DE FR GB

(71) Applicant: **INTERNATIONAL BUSINESS MACHINES CORPORATION**
Old Orchard Road
Armonk, N.Y. 10504 (US)

(72) Inventor: **Cooper, Thomas Edward**
858 West Willow Street
Louisville,
Colorado 80027 (US)

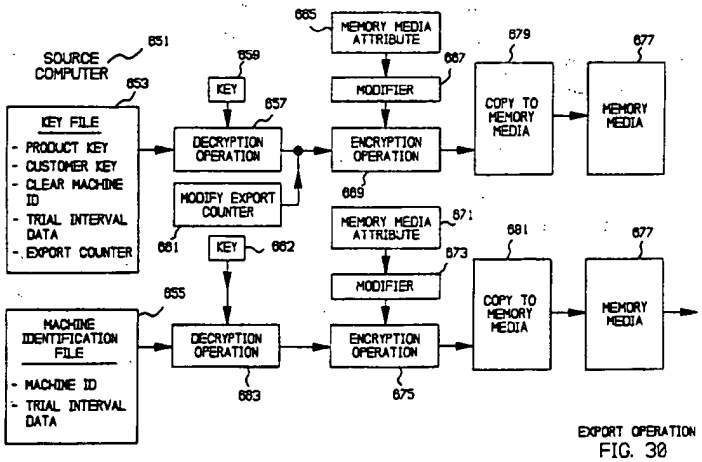
Inventor: **Nagda, Jagdish**
701 Kalmia Avenue
Boulder,
Colorado 80304 (US)

Inventor: **Pryor, Robert Franklin**
7380 Mt. Meeker Road
Lognmont,
Colorado 80503 (US)

(74) Representative: **Schäfer, Wolfgang, Dipl.-Ing.**
IBM Deutschland
Informationssysteme GmbH
Patentwesen und Urheberrecht
D-70548 Stuttgart (DE)

(54) Method and apparatus enabling software trial allowing the distribution of software objects.

(57) A method and apparatus is provided for transferring encrypted files from a source computer to one or more target computers. An export program is provided in the source computer and an import program is provided in the target computer. The export program decrypts the encrypted file and tags the export operation with an export counter value. The clear text file is then encrypted with an encryption operation utilizing a key which is unique to a transfer memory media, such as diskette serial number. The memory media is carried to a target computer which utilizes the import file to decrypt the encrypted file.



EXPORT OPERATION
FIG. 30

EP 0 679 977 A1

CROSS-REFERENCE TO RELATED APPLICATION

The present application is related to U.S. Patent Application Serial No. 08/235,033, entitled "Method and Apparatus for Enabling Trial Period Use of Software Products: Method and Apparatus for Utilizing a Decryption Stub," further identified by Attorney Docket No. BT9-93-070; U.S. Patent Application Serial No. 08/235,035, entitled "Method and Apparatus for Enabling Trial Period Use of Software Products: Method and Apparatus for Allowing a Try-and-Buy User Interaction," further identified by Attorney Docket No. DA9-94-008; U.S. Patent Application Serial No. 08/235,032, entitled "Method and Apparatus for Enabling Trial Period Use of Software Products: Method and Apparatus for Generating a Machine-Dependent Identification," further identified by Attorney Docket No. DA9-94-009; and U.S. Patent Application Serial No. 08/235,418, entitled "Method and Apparatus for Enabling Trial Period Use of Software Products: Method and Apparatus for Utilizing an Encryption Header," further identified by Attorney Docket No. DA9-94-010, all filed of even date herewith by the inventors hereof and assigned to the assignee herein, and incorporated by reference herein.

BACKGROUND OF THE INVENTION

1.Technical Field:

The present invention relates in general to techniques for securing access to software objects, and in particular to techniques for temporarily encrypting and restricting access to software objects.

2. Description of the Related Art:

The creation and sale of software products has created tremendous wealth for companies having innovative products, and this trend will continue particularly since consumers are becoming evermore computer literate as time goes on. Computer software is difficult to market since the potential user has little opportunity to browse the various products that are available. Typically, the products are contained in boxes which are shrink-wrapped closed, and the potential customer has little or no opportunity to actually interact with or experience the software prior to purchasing. This causes considerable consumer dissatisfaction with products, since the consumer is frequently forced to serially purchase a plurality of software products until an acceptable product is discovered. This is perhaps one significant cause of the great amount of software piracy which occurs in our economy. A potential software purchaser will frequently "borrow" a set of diskettes from a friend or business associate, with the stated intention of using the software for a temporary period. Frequently, such temporary use extends for long intervals and the potential customer may never actually purchase a copy of the software product, and may instead rely upon the borrowed copy.

Since no common communication channel exists for the sampling of software products, such as those created in movie theaters by movie trailers, and in television by commercials, software manufacturers are forced to rely upon printed publications and direct mail advertisements in order to advertise new products and solicit new customers. Unfortunately, printed publications frequently fail to provide an accurate description of the product, since the user interaction with the product cannot be simulated in a static printed format. The manufacturers of computer software products and the customers would both be well served if the customers could have access to the products prior to making decisions on whether or not to purchase the product, if this could be accomplished without introducing risk of unlawful utilization of the product.

The distribution of encrypted software products is one mechanism a software vendor can utilize to distribute the product to potential users prior to purchase; however, a key must be distributed which allows the user access to the product. The vendor is then forced to rely entirely upon the honesty and integrity of a potential customer. Unscrupulous or dishonest individuals may pass keys to their friends and business associates to allow unauthorized access. It is also possible that unscrupulous individuals may post keys to publicly-accessible bulletin boards to allow great numbers of individuals to become unauthorized users. Typically, these types of breaches in security cannot be easily prevented, so vendors have been hesitant to distribute software for preview by potential customers.

SUMMARY OF THE INVENTION

It is one object of the present invention to provide a method and apparatus for distributing software objects from a producer to potential users which allows the user a temporary trial period without subjecting the software product to unnecessary risks of piracy or unauthorized utilization beyond the trial interval. Preferably this is accomplished by providing a software object on a computer-accessible memory media along with a file management program. Preferably, the software object is reversibly functionally limited, through one or more particular encryption operations. The computer-accessible memory media is shipped from the producer

to the potential user utilizing conventional mail and delivery services. Upon receipt, the potential user loads the file management program into a user-controlled data processing system and associates it with the operating system for the data processing system. Then, the computer-accessible memory media is read utilizing the user-controlled data processing system. The file management program is executed by the user-controlled data processing system and serves to restrict access to the software object for a predefined and temporary trial period. During the temporary trial mode of operation, the software object is temporarily enabled by reversing the reversible functional limitation of the software object. This is preferably accomplished by decryption of the encrypted software object when the software object is called by the operating system of the user-controlled data processing system. The file management program preferably prevents copying operations, so the encrypted software project is temporarily decrypted when it is called by the operating system. If the potential user elects to purchase the software object, a permanent use mode of operation is entered, wherein the functional limitation of the software object is permanently reversed, allowing unlimited use to the software object by the potential user. This facilitates browsing operations which allow the potential user to review the software and determine whether it suits his or her needs.

The file management program continuously monitors the operating system of the user-controlled data processing system for operating system input calls and output calls. The file management program identifies when the operating system of the user-controlled data processing system calls for a software object which is subject to trial-interval browsing. Then, the file management system fetches a temporary access key associated with the software object, and then examines the temporary access key to determine if it is valid. Next, the file management program reverses the functional limitation of the software object, and passes it to the data processing system for processing.

It is another objective of the present invention to provide a method and apparatus for distributing a software object from a source to a user, wherein a software object is encrypted utilizing a long-lived encryption key, and directed from the source to the user. The encrypted software object is loaded onto a user-controlled data processing system having a particular system configuration. A numerical machine identification based at least in part upon the particular configuration of the user-controlled data processing system is then derived. Next, a temporary key is derived which is based at least in part upon the numerical machine identification and the long-lived encryption key. A long-lived key generator is provided for receiving the temporary key and producing the long-lived encryption key. The temporary key allows the user to generate for a prescribed interval the long-lived encryption key to access the software object. These operations are performed principally by a file management program which is operable in a plurality of modes. These modes include a set up mode of operation, a machine identification mode of operation, and a temporary key derivation mode of operation. During the set up mode of operation, the file management program is loaded onto a user-controlled data processing system and associated with an operating system for the user-controlled data processing system. During the machine identification mode of operation, the file management program is utilized to derive a numerical machine identification based upon at least on attribute of the user-controlled data processing system. During the temporary key derivation mode of operation, a temporary key is derived which is based at least in part upon the numerical machine identification. The file management program also allows a trial mode of operation, wherein the file management program is utilized by executing it with the user-controlled data processing system to restrict access to the software object for an interval defined by the temporary key, during which the long-lived key generator is utilized in the user-controlled data processing system to provide the long-lived key in response to receipt of at least one input including the temporary key.

It is yet another objective of the present invention to provide a method and apparatus in a data processing system for securing access to particular files which are stored in a computer-accessible memory media. A file management program is provided as an operating system component of the data processing system. A plurality of files are stored in the computer-accessible memory media, including at least one encrypted file and at least one unencrypted file. For each encrypted file, a preselected portion is recorded in computer memory, a decryption block is generated which includes information which can be utilized to decrypt the file, and the decryption block is incorporated into the file in lieu of the preselected portion which has been recorded elsewhere in computer memory. The file management program is utilized to monitor data processing operation calls for a called file stored in the computer-accessible memory media. The file management program determines whether the called file has an associated decryption block. The file management program processes the called file in a particular manner dependent upon whether or not the called file has an associated decryption block. The incorporation of the decryption block does not change the size of the encrypted file, thus

preventing certain types of processing errors. During the trial interval, the encrypted file is maintained in an encrypted condition, and cannot be copied. If the potential user opts to purchase the software product, a permanent key is provided which results in replacement of the preselected portion to the file in lieu of the decryption block. Once the decryption block is removed, the encrypted file may be decrypted to allow unrestricted use by the purchaser. Preferably, the file management program is utilized to intercept files as they are called by the operating system, and to utilize the decryption block to derive a name for a key file and read the called file. The decryption block of each encrypted file includes a validation segment which is decrypted by the file management program and compared to a selected segment for the called file to determine whether the key can decrypt the particular file. If the decrypted validation segment matches a known clear text validation segment, the file is then dynamically decrypted as it is passed for further processing.

It is yet another objective of the present invention to provide a method and apparatus in a data processing system for securing access to particular files which are stored in a computer-accessible memory media. A file management program is provided as an operating system component of a data processing system. In a computer-accessible memory media available to the data processing system, at least one encrypted file and one unencrypted file are stored. The encrypted file has associated with it an unencrypted security stub which is at least partially composed of executable code. The file management program is utilized to monitor the data processing system calls for a called file stored in the computer accessible memory media, to determine whether the called file has an associated unencrypted security stub, and to process the called file in a particular manner dependent upon whether or not the called file has an associated unencrypted security stub. More particularly, if it is determined that the called file has no associated unencrypted security stub, the called file is allowed to be processed. However, if it is determined that the called file has an associated unencrypted security stub, it must be examined before a decision can be made about whether or not to allow it to be processed. First, the unencrypted security stub is examined in order to obtain information which allows decryption operations to be performed. Then, the decryption operations are performed. Finally, the called file is allowed to pass for further processing. Preferably, the called file is dynamically decrypted as it is passed to the operating system for processing. Also, the unencrypted security stub is separated from the called file prior to execution of the called file. However, if the unencrypted security stub accidentally remains attached to the called file, processing operations must be stopped, and a message must be posted in order to prevent the processor from becoming locked-up.

It is still another objective of the present invention to provide a method and apparatus for distributing a software object from a source to a user. A computer-accessible memory media is distributed from the source to a potential user. It includes a software object which is encrypted utilizing a predetermined encryption engine and a long-lived and secret key. An interface program is provided which facilitates interaction between the source and the user. The interface program includes machine identification module which generates a machine identification utilizing at least on predetermined attribute of the user-controlled data processing system. It also further includes a long-lived and secret key generator which receives as an input at least a temporary key and produces as an output a long-lived and secret key. A validation module is provided which tests temporary key determined its validity. The source of the software object maintains a temporary key generator which receives as an input at least a machine identification and produces an output of the temporary key. An interface program is loaded onto the user-controlled data processing system. The machine identification module is utilized to examine at least one predetermined attribute of the user-controlled data processing system and to generate the machine identification. During interaction between the source and the user, the machine identification is communicated over an insecure communication channel. At the source of the software object, the temporary key is generated utilizing the machine identification (and other information) as an input to the temporary key generator. During interaction between the source and the user, the temporary key is communicated, typically over an insecure communication channel. Next, the validation module is utilized to determine the validity of the temporary key. The long-lived and secret key generator is then utilized to receive the temporary key and generate the long-lived and secret key in order to decrypt and temporarily gain access to the software object. The user is also provided with an import module and an export module which allow for the utilization of portable memory media to transfer the encrypted software object, a key file, and a machine identification file from one machine in a distributed data processing system to another machine in the distributed data processing system, while allowing the temporary key to allow temporary trial access to the software object.

The above as well as additional objectives, features, and advantages of the present invention

will become apparent in the following detailed written description.

BRIEF DESCRIPTION OF THE DRAWINGS

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself, however, as well as a preferred mode of use, further objectives and advantages thereof, will best be understood by reference to the following detailed description of an illustrative embodiment when read in conjunction with the accompanying drawings, wherein:

Figure 1 is a pictorial representation of a stand-alone data processing system, a telephone, and a variety of computer-accessible memory media all of which may be utilized in the implementation of the preferred technique of enabling trial period use of software products;

Figure 2 is a pictorial representation of a distributed data processing system which may utilize the technique of the present invention of enabling trial period use of software products;

Figure 3 is a block diagram representation of data processing system attributes which may be utilized to generate a machine identification, in accordance with the present invention;

Figure 4 is a block diagram depiction of a routine for encrypting software objects;

Figure 5 is a pictorial representation of the exchange of information between a source (a software vendor) and a user (a customer), in accordance with the teachings of the present invention;

Figure 6 is a flowchart representation of the broad steps employed in building a user interface shell, in accordance with the present invention;

Figure 7 is a flowchart representation of vendor and customer interaction in accordance with the present invention;

Figures 8, 9, 10a, and 10b depict user interface screens which facilitate trial period operations in accordance with the present invention;

Figure 11 depicts a user interface which is used to initiate a temporary access key;

Figure 12 is a block diagram depiction of the preferred technique of generating a machine identification;

Figure 13 is a block diagram depiction of an encryption operation which is utilized to encrypt a machine identification, in accordance with the present invention;

Figure 14 is a block diagram representation of the preferred technique for generating a product key, in accordance with the present invention;

Figure 15 is a block diagram representation of a preferred technique utilizing a temporary product key to generate a real key which can be utilized to decrypt one or more software objects;

Figures 16 and 17 depict a preferred technique of validating the real key which is derived in accordance with the block diagram of Figure 15;

Figure 18 is a block diagram depiction of the preferred routine for encyrpting a key file which contains information including a temporary product key;

Figure 19 is a block diagram depiction of the preferred technique of handling an encryption header in an encrypted file, in accordance with the present invention;

Figure 20 depicts in block diagram form the technique of utilizing a plurality of inputs in the user-controlled data processing system to derive the real key which may be utilized to decrypt an encrypted software object;

Figure 21 depicts a decryption operation utilizing the real key derived in accordance with Figure 20;

Figure 22 is a block diagram depiction of a comparison operation which is utilized to determine the validity of the real key;

Figure 23 depicts a decryption operation utilizing a validated real key;

Figures 24, 25, 26, 27, 28 depict the utilization of an encryption header in accordance with the present invention;

Figure 29 is a flowchart representation of the preferred technique of providing a trial period of use for an encrypted software object;

Figures 30 and 31 depict export and import operations which may be utilized to perform trial period use operations in a distributed data processing system;

Figures 32 and 33 provide an alternative view of the import and export operations which are depicted in Figures 30 and 31;

Figures 34 and 35 provide a block diagram depiction of an alternative technique for performing an export/import operation.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENT

The method and apparatus of the present invention for enabling trail period use of software products can be utilized in stand-alone PCs such as that depicted in Figure 1, or in distributed data processing systems, such as that depicted in Figure 2. In either event, temporary trial period access to one or more software products depends upon utilization of the trial product on a particular data processing system with particular data processing system attributes. This is accomplished by encrypting the trial software product utilizing a temporary access key which is based upon one or more data

processing system attributes. Figure 3 graphically depicts a plurality of system configuration attributes, which may be utilized in developing a temporary access key, as will be described in greater detail herebelow. To begin with, the environment of the stand-alone data processing system of Figure 1, and the distributed data processing system of Figure 2 will be described in detail, followed by a description of particular system configuration attributes which are depicted in Figure 3.

With reference now to the figures and in particular with reference to Figure 1, there is depicted a pictorial representation of data processing system 10 which may be programmed in accordance with the present invention. As may be seen, data processing system 10 includes processor 12 which preferably includes a graphics processor, memory device and central processor (not shown). Coupled to processor 12 is video display 16 which may be implemented utilizing either a color or monochromatic monitor, in a manner well known in the art. Also coupled to processor 12 is keyboard 14. Keyboard 14 preferably comprises a standard computer keyboard which is coupled to the processor by means of a cable.

Also coupled to processor 12 is a graphical pointing device, such as mouse 20. Mouse 20 is coupled to processor 12, in a manner well known in the art, via a cable. As is shown, mouse 20 may include left button 24, and right button 26, each of which may be depressed, or "clicked", to provide command and control signals to data processing system 10. While the disclosed embodiment of the present invention utilizes a mouse, those skilled in the art will appreciate that any graphical pointing device such as a light pen or touch sensitive screen may be utilized to implement the method of the present invention. Upon reference to the foregoing, those skilled in the art will appreciate that data processing system 10 may be implemented utilizing a so-called personal computer, such as the Model 80 PS/2 computer manufactured by International Business Machines Corporation of Armonk, New York.

While the present invention may be utilized in stand-alone data processing systems, it may also be utilized in a distributed data processing system, provided the import and export routines of the present invention are utilized to transfer one or more encrypted files, their encrypted key files, and associated file management programs through a portable memory media (such as diskettes or tapes) between particular data processing units within the distributed data processing system. While the import and export routines of the present invention will be described in greater detail herebelow, it is important that a basic distributed data processing system be described and understood.

Figure 3 provides a block diagram depiction of a plurality of data processing system attributes which may be utilized to uniquely identify a particular data processing system (whether a stand-alone or a node in a distributed data processing system), and which further can be utilized to generate in the machine identification value which is utilized to derive or generate a temporary access product key which may be utilized to gain access to an encrypted product for a particular predefined trial interval. A data processing system may include a particular system bus 60 architecture, a particular memory controller 74, bus controller 76, interrupt controller 78, keyboard mouse controller 80, DMA controller 66, VGA video controller 82, parallel controller 84, serial controller 86, diskette controller 88, and disk controller 82. Additionally, a plurality of empty or occupied slots 106 may be used to identify the particular data processing system. Each particular data processing system may have attributes which may be derived from RAM 70, ROM 68, or CMOS RAM 72. End devices such as printer 96, monitor 94, mouse 92, keyboard 90, diskette 100, or disk drive 104 may be utilized to derive one or more attributes of the data processing system which may be processed in a predetermined manner to derive a machine identification value. The derivation of the machine identification value will be described in greater detail below. The present invention is directed to an efficient method of distributing software programs to users which would provide to them a means to try the program before obtaining (by purchasing) a license for it. In accordance with this concept, complete programs are distributed to potential users on computer-accessible memory media such as diskettes or CD-ROMs. The concept is to generate keys that allow the user to access the programs from the distributed media. In this environment, a file management program provides a plurality of interfaces which allows the user to browse the different products. The interfaces allow ordering and unlocking of the software products contained on the distributed media. Unlocking of the software product is accomplished by the reception, validation, and recording of a temporary access (decryption) key.

The file management program is resident in the user-controlled data processing system and becomes a part of the operating system in the user's computer. An example of such a resident program (in the PC DOS environment) would be a resident program TSR, for "terminate and stay resident" operations, that intercepts and handles DOS file input and output operations. When a temporary access key is provided to a user, system files are checked to see if this file has been used in a trial mode of operation before. If the product has

never been used in a trial mode of operation, the temporary key is saved. Once the trial mode of operation key exists, an encrypted application can only be run if it is initiated by the file management program. The file management program will recognize that the application is encrypted and that a valid trial mode of operation key exists for the particular operation. A valid trial mode of application key is one that has not expired. The trial mode of operation may be defined by either a timer, or a counter. A timer can be used to count down a particular predefined period (such as thirty days); alternatively, the counter can be used to decrement through a predefined number of trial "sessions" which are allowed during the trial mode of operation. If the key is valid, the file management program communicates directly with the TSR and enables the trial mode of operation for a particular encrypted application. The file management program then kicks off the encrypted application. The code which is resident in the operating system of the user-controlled data processing system maintains control over the operating system. It monitors the use of the trial mode of operation keys to allow files to be decrypted and loaded into memory, but prevents the encrypted files from being decrypted and copied to media. This is done by using the operating system to determine which applications are trying to access the data and only allowing the applications that have permission to access the data to do so.

Figure 4 is a block diagram depiction of a routine for encrypting software objects. The binary characters which make up software object 201 are supplied as an input to encryption engine 205. Real key 203 is utilized as an encryption key in encryption engine 205. The output of encryption engine 205 is an encrypted software object 207. Encryption engine 205 may be any conventional encryption operation such as the published and well known DES algorithm; alternatively, the encryption engine 205 may be an exclusive-OR operation which randomizes software object 201.

Figure 5 is a pictorial representation of the exchange of information between a source 209 (a software vendor) and a user 211 (a potential customer, in accordance with the teachings of the present invention. The arrows between source 209 and user 211 represent exchanges of objects or information between vendor 209 and 211. In the exchange of flow 203, computer-accessible memory media is directed from source 209 to user 211. This transfer may occur by US mail delivery, courier delivery, express service delivery, or by delivery through printed publications such as books and magazines. Alternatively, an electronic document may be transferred from source 209 to user 211 utilizing electronic mail or other transmission techniques. In flow 215, user-specific information, preferably including a unique machine identification number which identifies the data processing system of user 211, is transferred from user 211 to source 209 via an insecure communication channel; typically, this information is exchanged over the telephone, but may be passed utilizing electronic mail or other communication techniques. In flow 217, source 209 provides a product key to user 211. The product key allows the product contained in the memory media to be temporarily accessed for a prescribed and predefined interval. This interval is considered to be a "trial" interval during which user 211 may become familiar with the software and make a determination on whether or not he or she wishes to purchase the software product. User 211 must communicate additionally with source 209 in order to obtain permanent access to the software product. The product key allows user 211 to obtain access to the software product for a particular predefined time interval, or for a particular number of predefined "sessions." As time passes, the user's clock or counter runs down. At the termination of the trial period, further access is denied. Therefore, the user 211 must take affirmative steps to contact source 209 and purchase a permanent key which is communicated to user 211 and which permanently unlocks a product to allow unrestricted access to the software product.

The communication between source 209 and user 211 is facilitated by a user interface. The creation of the interface is depicted in flowchart form in Figure 6. The process begins at software block 219, and continues at software block 221, wherein source 209 makes language and locale selections which will determine the language and currencies utilized in the interface which facilitates implementation of the trial period use of the software products. A plurality of software products may be bundled together and delivered to user 211 on a single computer-accessible memory media. Therefore, in accordance with software block 223, source 209 must make a determination as to the programs which will be made available on a trial basis on the computer-accessible memory media, and the appropriate fields are completed, in accordance with software block 223. Next, in accordance with software block 225, the programs are functionally limited or encrypted. Then, in accordance with software block 227, the shell is loaded along with the computer program products onto a computer-accessible memory media such as a diskette or CD ROM. The process ends at software block 229.

Figure 7 is a flowchart representation of vendor and customer interaction in accordance with the present invention. The flow begins at software block 231, and continues at step 233, wherein

computer-accessible memory media are distributed to users for a try-and-buy trial interval. Then, in accordance with step 235, the file management program is loaded from the computer-accessible memory media onto a user-controlled data processing system for execution. The file management program includes a plurality of interface screens which facilitate interaction between the vendor and the customer, which and which set forth the options available to the customer. Thus, in accordance with step 237, the file management program allows browsing and displays appropriate user interfaces. Next, in accordance with step 239, the customer and the vendor interact, typically over the telephone or electronic mail, to allow the vendor to gather information about the customer and to distribute a temporary key which allows access to one or more software products which are contained on the computer-accessible memory media for a predefined trial interval. Typically, the interval will be defined by an internal clock, or by a counter which keeps track of the number of sessions the potential purchaser has with a particular software product or products. Step 241 represents the allowance of the trial interval use. Then, in accordance with software block 243, the file management program monitors and oversees all input and output calls in the data processing system to prevent unauthorized use of the encrypted software products contained on the computer-accessible memory media. In the preferred embodiment of the present invention, the file management program monitors for calls to encrypted files, and then determines whether access should be allowed or denied before the file is passed for further processing. The customer can assess the software product and determine whether he or she desires to purchase it. If a decision is made to purchase the product, the customer must interact once again with the vendor, and the vendor must deliver to the customer a permanent key, as is set forth in step 245. The process ends when the customer receives the permanent key, decrypts the one or more software products that he or she has purchased, and is then allowed ordinary and unrestricted access to the software products.

Figures 8, 9, 10a, and 10b depict user interface screens which facilitate trial period operations in accordance with the present invention. Figure 8 depicts an order form user interface 249 which is displayed when the customer selects a "view order" option from another window. The order form user interface 249 includes a title bar 251 which identifies the software vendor and provides a telephone number to facilitate interaction between the potential customer and the vendor. An order form field 255 is provided which identifies one or more software products which may be examined during a trial interval period of operation. A plurality of subfields are provided including quantity subfield 259, item subfield 257, description subfield 260, and price subfield 253. Delete button 261 allows the potential customer to delete items from the order form field. Subtotal field 263 provides a subtotal of the prices for the ordered software. Payment method icons 265 identify the acceptable forms of payment. Of course, a potential user may utilize the telephone number to directly contact the vendor and purchase one or more software products; alternatively, the user may select one or more software products for a trial period mode of operation, during which a software product is examined to determine its adequacy. A plurality of function icons 267 are provided at the lowermost portion of order form interface 249. These include a close icon, fax icon, mail icon, print icon, unlock icon, and help icon. The user may utilize a graphical pointing device in a conventional point-and-click operation to select one or more of these operations. The fax icon facilitates interaction with the vendor utilizing a facsimile machine or facsimile board. The print icon allows the user to generate a paper archival copy of the interaction with the software vendor.

The customer, the computer-accessible memory media, and the computer system utilized by the customer are identified by media identification 269, customer identification 273, and machine identification 271. The media identification is assigned to the computer-accessible memory media prior to shipping to the potential customer. It is fixed, and cannot be altered. The customer identification 273 is derived from interaction between the potential customer and the vendor. Preferably, the customer provides answers to selected questions in a telephone dialogue, and the vendor supplies a customer identification 273, which is unique to the particular customer. The machine identification 271 is automatically derived utilizing the file management program which is resident on the computer-accessible memory media, and which is unique to the particular data processing system being utilized by the potential customer. The potential customer will provide the machine identification to the vendor, typically through telephone interaction, although fax interaction and regular mail interaction is also possible.

Figure 9 is a representation of an order form dialog interface 275. This interface facilitates the acquisition of information which uniquely identifies the potential customer, and includes name field 277, address field 279, phone number field 281, facsimile number field 283, payment method field 285, shipping method field 287, account number field 289, expiration date field 291, value added tax ID field 293. Order information dialog interface 275

further includes print button 295 and cancel button 297 which allow the potential user to delete information from these fields, or to print a paper copy of the interface screen.

Figures 10a and 10b depict unlock dialog interface screens 301, 303. The user utilizes a graphical pointing device to select one or more items which are identified by the content item number field 307 and description field 309 which are components of unlock list 305. The interface further includes customer ID field 313 and machine ID field 315. Preferably, the vendor provides the customer identification to the customer in an interaction via phone, fax, or mail. Preferably, the customer provides to the vendor the machine identification within machine identification field 315 during interaction via phone, fax, or mail. Once the information is exchanged, along with an identification of the products which are requested for a trial interval period of operation, a temporary access key is provided which is located within key field 311. The key will serve to temporarily unlock the products identified and selected by the customer. Close button 319, save button 317, and help button 321 are also provided in this interface screen to facilitate user interaction.

Figure 10b depicts a single-product unlock interface screen 303. This interface screen includes only machine identification field 315, customer identification field 315, and key field 311. The product which is being unlocked need not be identified in this interface, since the dialog pertains only to a single product, and it is assumed that the user knows the product for which a temporary trial period of operation is being requested. Save button 317, cancel button 319, and help button 321 are also provided in this interface to facilitate operator interaction.

Figure 11 depicts a user interface screen which is utilized in unlocking the one or more encrypted products for the commencement of a trial interval mode of operation. The starting date dialog of Figure 11 is displayed after the "SAVE" push button is selected in the unlock dialog of either Figure 10a or Figure 10b. The user will be prompted to verify the correct starting date which is provided in date field 310. The user responds to the query by pointing and clicking to either the "continue" button 312, the "cancel" button 314, or the "help" button 316. The date displayed in field 310 is derived from the system clock of the user-controlled data processing system. The user may have to modify the system clock to make the date correspond to the official or stated date of commencement of the trial period of operation.

A trial interval operation can take two forms: one form is a functionally disabled product that allows a user to try all the features, but may not allow a critical function like printing or saving of data files. Another type of trial interval is a fully functional product that may be used for a limited time. This requires access protection, and allows a customer to try all the functions of a product for free or for a nominal fee. Typically, in accordance with the present invention, access to the product is controlled through a "timed" key. The trial period for using the product is a fixed duration determined by the vendor. The trial period begins when the key is issued. In accordance with the present invention, the products being previewed during the trial interval of operation can only be run from within a customer shell. A decryption driver will not allow the encrypted products to be copied in the clear, nor will it allow the product to be run outside the customer's shell. In an alternative embodiment, the trial interval is defined by a counter which is incremented or decremented with each "session" the customer has with the product. This may allow the customer a predefined number of uses of the product before decryption is no longer allowed with the temporary key.

The limits of the temporary access key are built into a "control vector" of the key. Typically, a control vector will include a short description of the key, a machine identification number, and a formatted text string that includes the trial interval data (such as a clock value or a counter value). The control vector cannot be altered without breaking the key. When a protected software product is run, the usage data must be updated to enforce the limits of the trial interval period of operation. In order to protect the clock or counter from tampering, its value is recorded in a multiple number of locations, typically in encrypted files. In the preferred embodiment of the present invention, the trial interval information (clock value and/or counter value) is copied to a "key file" which will be described in further detail herebelow, to a machine identification file, which will also be discussed herebelow, and to a system file. When access to an encrypted program is requested, all of these locations are checked to determine if the value for the clock and/or counter is the same. It is unlikely that an average user has the sophistication to tamper successfully with all three files. In the preferred embodiment, a combination of a clock and a counter is utilized to prevent extended use of backup and restore operations to reset the system clock. Although it is possible to reset a PC's clock each time a trial use is requested, this can also be detected by tracking the date/time stamps of certain files on the system and using the most recent date between file date/time stamps and the system clock. As stated above, one of the three locations the timer and/or counter information is stored is a system file. When operating in an OS/2 operating

system, the time and usage data can be stored in the system data files, such as the OS2.INI in the OS/2 operating system. The user will have to continuously backup and restore these files to reset the trial and usage data. These files contain other data that is significant to the operation of the user system. The casual user can accidentally lose important data for other applications by restoring these files to an older version. In the present invention, these protection techniques greatly hinder a dishonest user's attempts to extend the trial interval use beyond the authorized interval.

In broad overview, in the present invention, the vendor loads a plurality of encrypted software products onto a computer-accessible memory media, such as a CD ROM or magnetic media diskette. Also loaded onto the computer-accessible memory media is a file management program which performs a plurality of functions, including the function of providing a plurality of user interface screens which facilitate interaction between the software vendor and the software customer. The computer-accessible memory media is loaded onto a user-controlled data processing system, and the file management program is loaded for execution. The file management program provides a plurality of user-interface screens to the software customer which gathers information about the customer (name, address, telephone number, and billing information) and receives the customer selections of the software products for which a trial interval is desired. Information is exchanged between the software vendor card customer, including: a customer identification number, a product identification number, a media identification number, and a machine identification number. The vendor generates the customer identification number in accordance with its own internal record keeping. Preferably, the representative of the software vendor gathers information from the software customer and types this information into a established blank form in order to identify the potential software customer. Alternatively, the software vendor may receive a facsimile or mail transmission of the completed order information dialog interface screen 275 (of Figure 9). The distributed memory media (such as CDs and diskettes) also include a file management program which is used to generate a unique machine identification based at least in part upon one attribute of the user-controlled data processing system. This machine identification is preferably a random eight-bit number which is created during a one-time setup process. Preferably, eight random bits are generated from a basic random number generator using the system time as the "seed" for the random number generator. Preferably, check bits are added in the final result. Those check bits are critical to the order system because persons taking orders must key in the machine ID that the customer reads over the phone. The check bits allow for instant verification of the machine ID without requiring the customer to repeat the number. Preferably, a master file is maintained on the user-controlled data processing system which contains the clear text of the machine identification and an encrypted version of the machine identification.

When the software customer places an order for a temporary trial use of the software products, he or she verbally gives to the telephone representative of the software vendor the machine identification. In return, the telephone representative gives the software customer a product key which serves as a temporary access key to the encrypted software products on the computer-accessible memory media, as well as a customer identification number. Preferably, the product key is a function of the machine identification, the customer number, the real encryption key for the programs or programs ordered, and a block of control data. The software customer may verify the product key by combining it with the customer number, and an identical block of control data to produce the real encryption key. This key is then used to decrypt an encrypted validation segment, to allow a compare operation. If the encrypted validation segment is identical to known clear text for the validation segment, then the user's file management program has determined that the product key is a good product key and can be utilized for temporary access to the software products. Therefore, if the compare matches, the key is stored on the user-controlled data processing system in a key file. Preferably, the key file contains the product key, a customer key (which is generated from the customer number and an internal key generating key) and a clear ASCII string containing the machine identification. All three items must remain unchanged in order for the decryption tool to derive the real encryption key. To further tie the key file to this particular user-controlled data processing system, the same key file is encrypted with a key that is derived from system parameters. These system parameters may be derived from the configuration of the data processing system.

Stated broadly, in the present invention the temporary key (which is given verbally over the phone, typically) is created from an algorithm that utilizes encryption to combine the real key with a customer number, the machine identification number, and other predefined clear text. Thus, the key is only effective for a single machine: even if the key were to be given to another person, it would not unlock the program on that other person's machine. This allows the software vendor to market software programs by distributing complete programs on computer-accessible memory media

such as diskettes or CD ROMs, without significant risk of the loss of licensing revenue.

Some of the preferred unique attributes of the system which may be utilized for encryption operations include the hard disk serial number, the size and format of the hard disk, the system model number, the hardware interface cards, the hardware serial number, and other configuration parameters. The result of this technique is that a machine identification file can only be decrypted on a system which is an identical clone of the user-controlled data processing system. This is very difficult to obtain, since most data processing systems have different configurations, and the configurations can only be matched through considerable effort. These features will be described in detail in the following written description.

Turning now to Figure 12, the file management program receives the distributed computer-accessible memory media with encrypted software products and a file management program contained therein. The file management program assesses the configuration of the user-controlled data processing system, as represented in step 351 of Figure 12. The user-specific attributes of the data processing system are derived in step 353, and provided as an input to machine identification generator 355, which is preferably a random number generator which receives a plurality of binary characters as an input, and generates a pseudo-random output which is representative of machine identification 357. The process employed by machine identification generator 355 is any conventional pseudo-random number generator which receives as an input of binary characters, and produces as an output a plurality of pseudo-random binary characters, in accordance with a predefined algorithm.

With reference now to Figure 13, machine identification 357 is also maintained within the file management program in an encrypted form. Machine identification 357 is supplied as an input to encryption engine 359 to produce as an output the encrypted machine identification 361. Encryption engine 359 may comprise any convention encryption routine, such as the DES algorithm. A key 363 is provided also as an input to encryption engine 359, and impacts the encryption operation in a conventional manner. Key 363 is derived from system attribute selector 365. The types of system attributes which are candidates for selection include system attribute listing 367 which includes: the hard disk serial number, the size of the hard disk, the format of the hard disk, the system model number, the hardware interface card, the hardware serial number, or other configuration parameters.

In accordance with the present invention, the clear text machine identification 357 and the encrypted machine identification 361 are maintained in memory. Also, in accordance with the present invention, the file management program automatically posts the clear text machine identification 357 to the appropriate user interface screens. The user then communicates the machine identification to the software vendor where it is utilized in accordance with the block diagram of Figure 14. As is shown, product key encryption engine 375 is maintained within the control of the software vendor. This product key encryption engine 375 receives as an input: the machine identification 357, a customer number 369 (which is assigned to the customer in accordance with the internal record keeping of this software vendor), the real encryption key 371 (which is utilized to decrypt the software products maintained on the computer-accessible memory media within the custody of the software customer), a control block text 373 (which can be any predefined textural portion), and trial interval data 374 (such as clock and/or counter value which defines the trial interval of use). Product key encryption engine produces as an output a product key 377. Product key 377 may be communicated to the software customer via an insecure communication channel, without risk of revealing real key 371. Real key 371 is masked by the encryption operation, and since the product key 377 can only be utilized on a data processing system having a configuration identical to that from which machine identification 357 has been derived, access to the encrypted software product is maintained in a secure condition.

Upon delivery of product key 377, the file management program resident in the user-controlled data processing system utilizes real key generator 379 to receive a plurality of inputs, including product key 377, customer number 369, control block text 373, machine identification 357 and trial interval data 374. Real key generator 379 produces as an output the derived real key 381.

Encryption and decryption algorithm utilized to perform the operations of the product key encryption engine 375 and the real key generator 379 (of Figures 14 and 15) is described and claimed in copending U.S. Patent Application Serial No. 07/964,324, filed October 21, 1992, entitled "Method and System for Multimedia Access Control Enablement", which is incorporated herein as if fully set forth.

Next, as is depicted in Figures 16 and 17, the derived real key 381 is tested to determine the validity and authenticity of the product key 377 which has been provided by the software vendor. As is shown, the derived real key 381 is supplied as an input to encryption engine 385. A predetermined encrypted validation data segment 383 is supplied as the other input to encryption engine 385. Encryption engine supplies as an output de-

rived clear validation text 387. Then, in accordance with Figure 17, the derived clear validation text 387 is compared to the known clear validation text 391 in comparator 389. Comparator 389 simply performs a bit-by-bit comparison of the derived clear validation text 387 with the known clear validation text 391. If the derived clear validation text 387 matches the known clear validation text 391, a key file is created in accordance with step 393; however, if the derived clear validation text 387 does not match the known clear validation text 391, a warning is posted to the user-controlled data processing system in accordance with step 395.

Turning now to Figure 18, key file 397 is depicted as including the temporary product key, the customer key (which is an encrypted version of the customer number), the machine identification number in clear text and the trial interval data (such as a clock and/or counter value). This key file is supplied as an input to encryption engine 399. Key 401 is also provided as an input to encryption engine 399. Key 401 is derived from unique system attributes 403, such as those system attributes utilized in deriving the machine identification number. Encryption engine 399 provides as an output the encrypted key file 405.

Figures 19, 20, 21, 22, and 23 depict operations of the file management program after a temporary access key has been received, and validated, and recorded in key file 397 (of Figure 18).

Figure 19 is a block diagram representation of the steps which are performed when an encrypted software product is called for processing by the user-control data processing system. The encrypted file 405 is fetched, and a "header" portion 407 is read by the user-controlled data processing system. The header has a number of components including the location of the key file. The location of the key file is utilized to fetch the key file in accordance with step 409. The header further includes an encrypted validation text 411. The encrypted validation text 411 is also read by the user-controlled data processing system. As is stated above (and depicted in Figure 18) the key file includes the product key 419, a customer key 417, and the machine identification 415. These are applied as inputs to decryption engine 413. Decryption engine 413 provides as an output real key 421. Before real key 421 is utilized to decrypt encrypted software products on the distributed memory media, it is tested to determine its validity. Figure 21 is a block diagram of the validation testing. Encrypted validation text 423, which is contained in the "header", is provided as an input to decryption engine 425. Real key 421 (which was derived in the operation of Figure 20) is also supplied as an input to decryption engine 425. Decryption engine 425 provides as an output clear validation text 427.

As is set forth in block diagram form in Figure 22, clear validation text 427 is supplied as an input to comparator 429. The known clear validation text 431 is also supplied as an input to comparator 429. Comparator 429 determines whether the derived clear validation text 427 matches the known clear validation text 431. If the texts match, the software object is decrypted in accordance with step 433; however, if the validation text portions do not match, a warning is post in accordance with step 435. Figure 23 is a block diagram depiction of the decryption operation of step 433 of Figure 22. The encrypted software object 437 is applied as an input to decryption engine 439. The validated real key 441 is also supplied as an input to decryption engine 439. Decryption engine 439 supplies as an output the decrypted software object 443.

The encryption header is provided to allow for the determination of whether or not a file is encrypted when that file is stored with clear-text files. In providing the encryption header for the encrypted file, it is important that the file size not be altered because the size may be checked as part of a validation step (unrelated in any way to the concept of the present invention) during installation. Therefore, making the file larger than it is suppose to be can create operational difficulties during installation of the software. The encryption header is further necessary since the file names associated with the encrypted software products cannot be modified to reflect the fact that the file is encrypted, because the other software applications that may be accessing the encrypted product will be accessing those files utilizing the original file names. Thus, altering the file name to indicate that the file is encrypted would prevent beneficial and desired communication between the encrypted software product and other, perhaps related, software products. For example, spreadsheet applications can usually port portions of the spreadsheet to a related word processing program to allow the integration of financial information into printed documents. Changing the hard-coded original file name for the word processing program would prevent the beneficial communication between these software products. The encryption header of the present invention resolves these problems by maintaining the encrypted file at its nominal file length, and by maintaining the file name for the software product in an unmodified form.

Figure 24 graphically depicts an encrypted file with encryption header 451. The encryption header 451 includes a plurality of code segments, including: unique identifier portion 453, the name of the key file portion 455, encrypted validation segment 457, encryption type 459, offset to side file 461, and encrypted file data 463. Of course, in this view, the encrypted file data 463 is representative of the

encrypted software product, such as a word processing program or spreadsheet. The encryption header 451 is provided in place of encrypted data which ordinarily would comprise part of the encrypted software product. The encryption header is substituted in the place of the first portion of the encrypted software product. In order to place the encryption header 451 at the front of the encrypted software product of encrypted file data 463, a portion of the encrypted file data must be copied to another location. Offset to side file 461 identifies that side file location where the displaced file data is contained.

Figure 25 graphically depicts the relationship between the directory of encrypted files and the side files. As is shown, the directory of encrypted files 465 includes file aaa, file bbb, file ccc, file ddd, through file nnn. Each of these files is representative of a directory name for a particular encrypted software product. Each encrypted software product has associated with it a side file which contains the front portion of the file which has been displaced to accommodate encryption header 451 without altering the size of the file, and without altering the file name. File aaa has associated with it a side file AAA. Software product file bbb has associated with it a side file BBB. Encrypted software product ccc has associated with it a side file CCC. Encrypted software product ddd has associated with it a side file DDD. Encrypted software product nnn has associated with it a side file NNN. In Figure 25, directory names 467, 469, 471, 473, 475 are depicted as being associated with side files 477, 479, 481, 483, and 485. The purpose of the side files is to allow each of the encrypted software products to be tagged with an encryption header without changing the file size.

Encryption type segment 459 of the encryption header 451 identifies the type of encryption utilized to encrypt the encrypted software product. Any one of a number of conventional encryption techniques can be utilized to encrypt the product, and different encryption types can be utilized to encrypt different software products contained on the same memory media. Encryption type segment 459 ensures that the appropriate encryption/decryption routine is called so that the encrypted software product may be decrypted, provided the temporary access keys are valid and not expired. The name of key file segment 455 of encryption header 451 provides an address (typically a disk drive location) of the key file. As is stated above (in connection with Figure 18) the key file includes the product key, a customer key, and the clear machine ID. All three of these pieces of information are required in order to generate the real key (in accordance with Figure 20). Encrypted validation segment 457 includes the encrypted validation text which is utilized in the routine depicted in Figure 21 which generates a derived clear validation text which may be compared utilizing the routine of Figure 22 to the known clear validation text. Only if the derived clear validation text exactly matches the known clear validation text can the process continue by utilizing the derived and validated real key to decrypt the encrypted software product in accordance with the routine of Figure 23. However, prior to performing the decryption operations of Figure 23, the contents of the corresponding side file must be substituted back into the encrypted software product in lieu of encryption header 451. This ensures that the encrypted software product is complete prior to the commencement of decryption operations.

Each time a file is called for processing by the operating system of the user-controlled data processing system, the file management program which is resident in the operating system intercepts the input/output requests and examines the front portion of the file to determine if a decryption block identifier, such as unique identifier 453, exists at a particular known location. For best performance, as is depicted in Figure 24, this location will generally be at the beginning of the file. If the file management program determines that the file has the decryption block, the TSR will read the block into memory. The block is then parsed in order to build a fully qualified key file name by copying an environment variable that specifies the drive and directory containing the key files and concatenating the key file name from the encryption block. The TSR then attempts to open the key file. If the key file does not exist, the TSR returns an "access denied" response to the application which is attempting to open the encrypted file. If the key file is determined to exist, the TSR opens the key file and reads in the keys (the product key, the customer key, and the machine identification) and generates the real key. This real key is in use to decrypt the decryption block validation data. As is stated above, a comparison operation determines whether this decryption operation was successful. If the compare fails, the key file is determined to be "invalid", and the TSR returns an "access denied message" to the application which is attempting to open the encrypted software product. However, if the compare is successful, the file management program prepares to decrypt the file according to the encryption type found in the encryption header. The TSR then returns a valid file handle to the calling application to indicate that the file has been opened. When the application reads data from the encrypted file, the TSR reads and decrypts this data before passing it back to the application. If the data requested is part of the displaced data that is stored in the side file, the TSR will read the side

file and return the appropriate decrypted block to the calling application without the calling application being aware that the data came from a separate file.

While the broad concepts of the encryption header are depicted in Figures 24 and 25, the more particular aspects of creating the encrypted files are depicted in Figures 26, 27, and 28. Figures 27 and 28 depict two types of data files. Figure 27 depicts a non-executing data file, while Figure 28 depicts an executing data file. Figure 26 depicts a header 499 which includes signature segment 501, header LEN 503, side file index 505, side file LEN 507, decryption type identifier 509, verification data 511, and key file name 518. As is shown in Figure 27, a software product begins as a clear file 521, and is encrypted in accordance with a particular encryption routine into encrypted file 523. Encryption type segment 509 of header 499 identifies the type of encryption utilized to change clear file 521 to encrypted file 523. Next, the front portion of encrypted file 523 is copied to side file 527 which is identified by side file index 505 and side file LEN 507 of header 499. Additionally, a copy of the clear text of the verification data is also included in side file 527. Then, header 499 is copied to the front portion of encrypted file 523 to form modified encrypted files 525. A similar process is employed for executing files, as depicted in Figure 28. The clear text copy of the software product (represented as clear file 531) is encrypted in accordance with a conventional routine, to form encrypted file 533. The front portion of encrypted file 533 is copied to side file 539 so that the overlaid data of encrypted file 533 is preserved. Furthermore, side file 539 includes a copy of the clear text of the verification data. Then, the encrypted file 533 is modified by overlaying and executable stub 537 and header 599 onto the first portion of encrypted file 553.

The purpose of executable stub 537 of Figure 28 will now be described. The DOS operating system for a personal computer will try to execute an encrypted application. This can result in a system "hang" or unfavorable action. The executable stub 357 of the executing file of Figure 28 is utilized to protect the user from attempting to execute applications that are encrypted: there would be considerable risk that a user would hang his system or format a drive if he or she try to run an encrypted file. The executable stub is attached to the front portion of the encrypted software product so that this stub is executed whenever the application is run without the installed TSR or run from a drive the TSR is not "watching". This stub will post a message to the user that explains why the application cannot run. In addition to providing a message, this executable stub can be used to perform sophisticated actions, such as:

(1) it can duplicate the functionality of the TSR and install dynamic encryption before kicking off the application a second time;

(2) it can turn on a temporary access key and kick off the application a second time;

(3) it can communicate with the TSR and inform it to look at the drive the application is being run from.

The executable stub is saved or copied into the encrypted program as follows:

(1) the application is encrypted;

(2) a decryption block is created for this program;

(3) a pre-built executable stub is attached to the front end of the decryption block;

(4) the length of the combined decryption header and executable stub is determined;

(5) the bytes at the front of the executable file equal to this length are then read into memory, preferably into a predefined side file location; and

(6) the encryption header and executable stub are then written over the leading bytes in the executable code.

The TSR can determine if an executable is encrypted by searching beyond the "known size" of the executable stub for the decryption block portion. When the TSR decrypts the executable stub it accesses the side file to read in the bytes that were displaced by the stub and header block.

Figure 29 provides a flowchart representation of operation during a trial period interval, which begins at software block 601. In accordance with software block 603, the file management program located in the operating system of the user-controlled data processing system continually monitors for input/output calls to the memory media. Then, in accordance with software block 605, for each input/output call, the called file is intercepted, and in accordance with software block 607 the operating system is denied access to the called file, until the file management program can determine whether access should be allowed or not. A portion of the called file is read where the decryption block should be located. This portion of the called file is then read, in accordance with software block 609, to derive a key file address in accordance with software block 611. The address which is derived is utilized to fetch the key file, in accordance with software block 613. In accordance with decision block 615, if the key file cannot be located, the process ends at software block 617; however, if it is determined in decision block 615 that the key file can be located, the key is derived in accordance with software block 619. The derived key is then utilized to decrypt the validation segment which is located within the encryption header, in

accordance with software block 621. In decision block 623, the decryption validation segment is compared to the clear text for the decryption validation segment; if it is determined that the decrypted segment does not match the known clear text segment, the process continues at software block 625 by ending; however, if it is determined in decision block 623 that the decrypted validation segment does match the known clear text validation segment, the process continues as software block 627, wherein access to the called file is allowed. Then, the decryption type is read from the decryption header in accordance with software block 629, and the called file is dynamically decrypted in accordance with software block 631 as it is passed for processing by the operating system of the user-controlled data processing system, in accordance with software block 633. The process terminates at software block 635.

If unauthorized execution of an encrypted file is attempted, the executable stub will at least temporarily deny access and post a message to the system, but may handle the problem in a number of sophisticated ways which were enumerated above.

In accordance with the preferred embodiment of the present invention, during the trial interval, or at the conclusion of the trial interval, the prospective purchaser may contact the vendor to make arrangements for the purchase of a copy of the one or more software products on the computer-accessible memory media. Preferably, CD ROMs or floppy disks have been utilized to ship the product to the potential user. Preferably, the computer-accessible memory media includes the two encrypted copies of each of the products which are offered for a trial interval of use. One encrypted copy may be decrypted utilizing the file management program and the temporary key which is communicated from the vendor to the purchaser. The other encrypted copy is not provided for use in the trial interval mode of operation, but instead is provided as the permanent copy which may be decrypted and utilized once the software product has been purchased. In broad overview, the user selects a software product for a trial interval mode of operation, and obtains from the vendor temporary access keys, which allow the user access to the product (through the file management program) for a predefined trial interval. Before or after the conclusion of the trial interval, the user may purchase a permanent copy of the software product from the vendor by contacting the vendor by facsimile, electronic mail, or telephone. Once payment is received, the vendor communicates to the user a permanent access key which is utilized to decrypt the second encrypted copy of the software product. This encrypted product may be encrypted utilizing any conventional encryption routine, such as the DES algorithm. The permanent key allows the software product to be decrypted for unrestricted use. Since multiple copies of the product may be purchased in one transaction, the present invention is equipped with a technique for providing movable access keys, which will be discussed below in connection with Figures 30 through 35. In the preferred embodiment of the present invention, the encryption algorithm employed to encrypt and decrypt the second copy of the software product is similar to that employed in the trial interval mode of operation.

The present invention includes an export/import function which allows for the distribution of permanent access keys, after the conclusion of a trial interval period. Typically, an office administrator or data processing system manager will purchase a selected number of "copies" of the encrypted product after termination of a trial interval period. Certain individuals within the organization will then be issued permanent keys which allow for the unrestricted and permanent access to the encrypted product. In an office or work environment where the computing devices are not connected in a distributed data processing network, the permanent access keys must be communicated from the office administrator or data processing manager to the selected individuals within an organization who are going to receive copies of the encrypted software product. The permanent keys allow for permanent access to the product. Since not all employees within an organization may be issued copies of the particular encrypted product, the vendor would like to have the distribution occur in a manner which minimizes or prevents the distribution beyond the sales agreement or license agreement. Since the products are encrypted, they may be liberally distributed in their encrypted form. It is the keys which allow unrestricted access to the product which are to be protected in the current invention. To prevent the distribution of keys on electronic mail or printed communications, the present invention includes an export program which is resident in a source computer and an import program which is resident in a target computer which allow for the distribution of the access keys via a removable memory media, such as a floppy diskette. This ensures that the access keys are not subject to inadvertent or accidental distribution or disclosure. There are two principal embodiments which accomplish this goal.

In the first embodiment, one or more encrypted files which are maintained in the source computer are first decrypted, and then encrypted utilizing an encryption algorithm and an encryption key which is unique to the transportable memory media (such as a diskette serial number). The key file may then

be physically carried via the diskette to a target computer, where it is decrypted utilizing a key which is derived by the target computer from interaction with the transferable memory media. Immediately, the key file or files are then encrypted utilizing an encryption operation which is keyed with a key which is derived from a unique system attribute of the target computer.

In the alternative embodiment, the transferrable memory media is loaded onto the target computer to obtain from the target computer import file a transfer key which is uniquely associated with the target computer, and which may be derived from one or more unique system attributes of the target computer. The memory media is then transferred to the source computer, where the one or more key files are decrypted, and then encrypted utilizing the transfer key. The memory media is then carried to the target computer where the transfer key is generated and utilized in a decryption operation to decrypt the one or more key files. Preferably, immediately the key files are encrypted utilizing an encryption operation which is keyed with a key which is uniquely associated with the target computer, and which may be derived from one or more unique computer configuration attributes. The first embodiment is discussed herein in connection with Figures 30, 31, 32, and 33. The second embodiment is discussed in connection with Figures 34 and 35.

Figures 30 and 31 depict in block diagram form export and import operations which allow an authorized user to move his permanent key to another data processing system using an "export" facility that produces a unique diskette image of the access key that has been enabled for import into another system. In accordance with the present invention, the access keys which are delivered over the telephone by the software vendor to the customer are less than 40 bytes in length. The key file that is produced is over 2,000 bytes in length. An export facility is provided for copying the key file and the machine identification file to a diskette. Both files are then encrypted with a modified diskette serial number to inhibit these files from being copied to a public forum where anyone could use them. An import facility provided in another system decrypts these files and adds the product key and machine identification from the diskette to a list of import product keys and machine identifications in the import systems master file, and copies the key file to the import system hard disk. The key file is encrypted on the import system as is disclosed above.

Figure 30 is a block diagram depiction of an export operation in accordance with the preferred embodiment of the present invention. As is shown, source computer 651 includes a key file 653 and a machine identification file 655. Key file 653 includes the product key, the customer key, the clear text of the machine identification for source computer 653, trial interval data (such as a clock and/or counter which define the trial interval period), and an export counter which performs the dual functions of defining the maximum number of export operations allowed for the particular protected software products and keeping track of the total number of export operations which have been accomplished. The machine identification file includes the machine identification number and trial interval data (such as a clock and/or counter which defines the trial interval period). Both key file 653 and machine identification file 655 are encrypted with any conventional encryption operation (such as the DES algorithm), which is keyed with a key which is derived from a unique system attribute of source computer 651. At the commencement of an export operation, key file 653 and machine identification file 655 are decrypted. Key file 653 is supplied as an input to decryption operation 657 which is keyed with key 659. Likewise, machine identification file 655 is supplied as an input to decryption operation 663 which is keyed with key 661. Decryption operations 657, 663 generate a clear text version of key file 653 and machine identification file 655. Once the clear text is obtained, the export counter which is contained within key file 653 is modified in accordance with block 661. For example, if this is the seventh permitted export operation out of ten permissible operations, the counter might read "7:10". The clear text version of key file 653 is supplied as an input to encryption operation 669. Encryption operation 669 may be any conventional encryption operation (such as the DES algorithm), which is keyed with a memory media attribute 665 which is unique to a memory media which is coupled to source computer 651, which has been subjected to modification of modifier 667. For example, a unique diskette serial number may be supplied as the "memory media attribute" which is unique to memory media 677. The diskette serial number is modified in accordance with modifier 667 to alter it slightly, and supply it as an input to encryption operations 669. The same operation is performed for the clear text of machine identification file 655. A unique memory media attribute 671 is modified by modifier 673 and utilized as a key for encryption operation 675, which may comprise any conventional encryption operation, such as the DES operation. Finally, the output of encryption operations 669 and 675 are supplied as inputs to copy operations 679, 681 which copy the encrypted key file 653 and machine identification file 655 to memory media 677.

Figure 31 is a block diagram depiction of an import operation. Memory media 677 (of Figure 30)

is physically removed from source computer 651 (of Figure 30) and physically carried over to computer 707 (of Figure 31); alternatively, in a distributed data processing system, this transfer may occur without the physical removal of memory media 677. With reference now to Figure 31, in accordance with block 683, the machine identification of the target machine is copied to memory media 677 to maintain a record of which particular target computer received the key file and machine identification file. Then, in accordance with blocks 685, 693 the encrypted key file 653 and machine identification file 655 are copied from the memory media to target computer 707. The encrypted key file 653 is supplied as an input to decryption operation 689 which is keyed with key 687. Decryption operation 689 reverses the encryption operation of block 669, and provides as an output a clear text version of key file 653. Likewise, machine identification file 655 is supplied as an input to decryption operation 697, which is keyed with key 695. Decryption operation 697 reverses the encryption of encryption operation 675 and provides as an output the clear text of machine identification file 655. In accordance with block 691, the machine identification of the source computer 651 is retrieved and recorded in memory in the clear text of key file 653. Next, the clear text of key file 653 is supplied as an input to encryption operation 699. Encryption operation 699 is a conventional encryption operation, such as the DES operation, which is keyed with a target computer unique attribute, such as the machine identification or modified machine identification for the target computer 707. The clear text of machine identification file 655 is supplied as an input to encryption operation 703. Encryption operation 703 is any conventional encryption operation, such as the DES encryption operation, which is keyed with a unique target computer attribute 705, such as machine identification or modified machine identification of target computer 707. The output of encryption operation 699 produces an encrypted key file 709 which includes a product key (which is the same temporary product key of key file 653 of source computer 651), a customer number (which is the same customer number of key file 653 of source computer 651), and clear machine identification (which is the machine identification for target computer 707, and not that of source computer 651), trial interval data (which is identical to the trail interval data of key file 653 of source 651), and an identification of the machine identification of the source computer 651. The output of encryption operation 703 defines machine identification file 711, which includes the machine identification of the target computer 707 (and not that of the source computer 651), and the trial interval data (which is identical to that of machine identification file 655 of source computer 651).

Figures 32 and 33 provide alternative views of the import and export operations which are depicted in Figures 30 and 31, and emphasize several of the important features of the present invention. As is shown, source computer 801 includes machine identification file 803 which is encrypted with a system attribute key which is unique to the source computer 801. The machine identification file includes machine identification file number as well as count of the number of exports allowed for each protected software product, and a count of the total number of exports which have been utilized. For example, the first export operation carries a count of "1:10", which signifies that one export operation of ten permitted export operations has occurred. In the next export operation, the counter is incremented to "2:20" which signifies that two of the total number of ten permitted export operations has occurred. Each target computer which receives the results of the export operation is tagged with this particular counter value, to identify that it is the recipient of a particular export operation. For example, one source computer system may carry a counter value of "1:10", which signifies that it is the recipient of the first export operation of ten permitted export operations. Yet another target computer may carry the counter value of "7:10", which signifies that this particular target computer received the seventh export operation of a total of ten permitted export operations. In this fashion, the target computer maintains a count of a total number of used export operations, while the source computers each carry a different counter value which identifies it a the recipient of the machine identification file and key file from the source computer from particular ones of the plurality of permitted export operations.

Note that in source computer 801 machine identification file 803 and key file 805 are encrypted with an encryption algorithm which utilizes as a key a system attribute which is unique to source computer 801; however, once machine identification file 803 and key file 805 are transferred to a memory media, such as export key diskette 807, machine identification file 809 and key file 811 are encrypted in any conventional encryption operation which utilizes as an encryption key a unique diskette attribute, such as the diskette's serial number. This minimizes the possibility that the content of the machine ID file 809 and/or key file 811 can be copied to another diskette or other memory media and then utilized to obtain unauthorized access to the software products. This is so because for an effective transfer of the content of machine ID file 809 and key file 811 to a target computer to occur, the target computer must be able to read and utilize the unique diskette attribute from the export

key diskette 807. Only when the machine ID file 809 and key file 811 are presented to a target computer on the diskette onto which these items were copied can an effective transfer occur. The presentation of the machine ID file 809 and key file 811 on a diskette other than export key diskette 807 to a potential target computer will result in the transfer of meaningless information, since the unique attribute of export key diskette 807 (such as the diskette serial number) is required by the target computer in order to successfully accomplish the decryption operation.

As is shown in Figure 33, export key diskette 807 is presented to target computer 813. Of course, the machine identification file 809 and key file 811 are in encrypted form. In the transfer from export key diskette 807 to target computer 813, the content of machine ID file 809 is updated with the machine identification of the target computer 813, and the count of imports utilized. In accomplishing the transfer to target computer 813, a machine identification file 815 is constructed which includes a number of items such as machine identification for the target computer 813, customer information, as well as a list of the machine identification number of the source computer 801. Both machine identification file 815 and the key file 817 are encrypted utilizing a conventional encryption operation which uses as a key a unique attribute of target computer 813. This ties machine identification file 815 and key file 817 to the particular target computer 813.

By using an export/import counter to keep track of the total number of authorized export/import operations, and the total number of used export/import operations, the present invention creates an audit trail which can be utilized to keep track of the distribution of software products during the trial interval. Each source computer will carry a record of the total number of export operations which have been performed. Each source computer will carry a record of which particular export/import operation was utilized to transfer one or more protected software products to the target computer. The memory media utilized to accomplish the transfer (such as a diskette, or group of diskettes) will carry a permanent record of the machine identification numbers of both the source computer and the target computer's utilized in all export/import operations.

The procedure for implementing export and import operations ensures that the protected software products are never exposed to unnecessary risks. When the machine identification file and key file are passed from the source computer to the export diskette, they are encrypted with the unique attribute of the export diskette which prevents or inhibits copying of the export diskette or posting of its contents to a bulletin board as a means for illegally distributing the keys. During the import operations, the machine identification and key files are encrypted with system attributes which are unique to the target computer to ensure that the software products are maintained in a manner which is consistent with the security of the source computer, except that those software products are encrypted with attributes which are unique to the target computer, thus preventing illegal copying and posting of the keys.

The second embodiment of the export/import function is depicted in block diagram form in Figures 34 and 35. In broad overview, memory media 1677 is first utilized to interact with target computer 1707 to obtain from target computer 1707 a transfer key which is unique to target computer 1707, and which is preferably derived from one or more unique system attributes of target computer 1707. The transfer key may he a modification of the machine identification for target computer 1707. Next, the memory media 1677 is utilized to interact with source computer 1651 in an export mode of operation, wherein key file 1653 and machine identification file 1655 are first decrypted, and then encrypted utilizing the transfer key.

Figure 34 is a block diagram depiction of an export operation in accordance with the preferred embodiment of the present invention. As is shown, source computer 1651 includes a key file 1653 and a machine identification file 1655. Key file 1653 includes the product key, the customer key, the clear text of the machine identification for source computer 1653, trial interval data (such as a clock and/or counter which define the trial interval period), and an export counter which performs the dual functions of defining the maximum number of export operations allowed for the particular protected software products and keeping track of the total number of export operations which have been accomplished. The machine identification file includes the machine identification number and trial interval data (such as a clock and/or counter which defines the trial interval period). Both key file 1653 and machine identification file 1655 are encrypted with any conventional encryption operation (such as the DES algorithm), which is keyed with a key which is derived from a unique system attribute of source computer 1651. At the commencement of an export operation, key file 1653 and machine identification file 1655 are decrypted. Key file 1653 is supplied as an input to decryption operation 1657 which is keyed with key 1659. Likewise, machine identification file 1655 is supplied as an input to decryption operation 1663 which is keyed with key 1661. Decryption operations 1657, 1663 generate a clear text version of key file 1653 and machine identification file 1655. Once the clear text

is obtained, the export counter which is contained within key file 1653 is modified in accordance with block 1661. For example, if this is the seventh permitted export operation out of ten permissible operations, the counter might read "7:10". The clear text version of key file 1653 is supplied as an input to encryption operation 1669. Encryption operation 1669 may be any conventional encryption operation (such as the DES algorithm), which is keyed with the transfer key 1665 which was previously obtained. The same operation is performed for the clear text of machine identification file 1655. Transfer key 1671 is utilized as a key for encryption operation 1675, which may comprise any conventional encryption operation, such as the DES operation. Finally, the output of encryption operations 1669 and 1675 are supplied as inputs to copy operations 1679, 1681 which copy the encrypted key file 1653 and machine identification file 1655 to memory media 1677.

Figure 35 is a block diagram depiction of an import operation. Memory media 1677 (of Figure 34) is physically removed from source computer 1651 (of Figure 34) and physically carried over to computer 1707 (of Figure 35); alternatively, in a distributed data processing system, this transfer may occur without the physical removal of memory media 1677. With reference now to Figure 35, in accordance with block 1683, the machine identification of the target machine is copied to memory media 1677 to maintain a record of which particular target computer received the key file and machine identification file. Then, in accordance with blocks 1685, 1693 the encrypted key file 1653 and machine identification file 1655 are copied from the memory media to target computer 1707. The encrypted key file 1653 is supplied as an input to decryption operation 1689 which is keyed with key 1687. Decryption operation 1689 reverses the encryption operation of block 1669, and provides as an output a clear text version of key file 1653. Likewise, machine identification file 1655 is supplied as an input to decryption operation 1697, which is keyed with key 1695. Decryption operation 1697 reverses the encryption of encryption operation 1675 and provides as an output the clear text of machine identification file 1655. In accordance with block 1691, the machine identification of the source computer 1651 is retrieved and recorded in memory in the clear text of key file 1653. Next, the clear text of key file 1653 is supplied as an input to encryption operation 1699. Encryption operation 1699 is a conventional encryption operation, such as the DES operation, which is keyed with a target computer unique attribute, such as the machine identification or modified machine identification for the target computer 1707. The clear text of machine identification file 1655 is supplied as an input to encryption operation 1703. Encryption operation 1703 is any conventional encryption operation, such as the DES encryption operation, which is keyed with a unique target computer attribute 1705, such as machine identification or modified machine identification of target computer 1707. The output of encryption operation 1699 produces an encrypted key file 1709 which includes a product key (which is the same temporary product key of key file 1653 of source computer 1651), a customer number (which is the same customer number of key file 1653 of source computer 1651), and clear machine identification (which is the machine identification for target computer 1707, and not that of source computer 1651), trial interval data (which is identical to the trail interval data of key file 1653 of source 1651), and an identification of the machine identification of the source computer 1651. The output of encryption operation 1703 defines machine identification file 1711, which includes the machine identification of the target computer 1707 (and not that of the source computer 1651), and the trial interval data (which is identical to that of machine identification file 1655 of source computer 1651).

While the invention has been particularly shown and described with reference to a preferred embodiment, it will be understood by those skilled in the art that various changes in form and detail may be made therein without departing from the spirit and scope of the invention.

Claims

1. A method of passing encrypted files between data processing systems, comprising:
   at a source computer providing at least one file which is encrypted with a key which is at least partially derived from at least one unique source computer system attribute;
   providing a transfer memory medium;
   at said source computer, decrypting said at least one file;
   at said source computer, encrypting said at least one file with a key which is derived from at least one unique transfer memory media attribute;
   at said source computer, copying said encrypted file to said transfer memory media;
   at a target computer, decrypting said at least one file;
   at said target computer, encrypting said at least one file with a key which is at least partially derived from at least one target computer system attribute.

2. A method of passing encrypted files between data processing systems, comprising:

at a source computer providing at least one file which is encrypted with a key which is at least partially derived from at least one unique source computer system attribute;
providing a transfer memory medium;
at a target computer copying a transfer encryption key which is unique to said target computer to said transfer memory media;
at said source computer, decrypting said at least one file;
at said source computer, encrypting said at least one file with said transfer encryption key;
at said source computer, copying said encrypted file to said transfer memory media;
at a target computer, decrypting said at least one file;
at said target computer, encrypting said at least one file with a key which is at least partially derived from at least one target computer system attribute.

3. A method of passing encrypted files according to Claims 1 or 2, further comprising:
providing a export counter in said source computer which defines a maximum number of permissible transfer operations; and
actuating said export counter for each transfer operation.

4. A method of passing encrypted files according to one of Claims 1 to 3, further comprising:
identifying each one of said permissible transfer operations to a particular target computer.

5. A method of passing encrypted files according to one of Claims 1 to 4, further comprising:
recording the occurrence of all transfer operations involving said transfer memory medium by obtaining identifying information from each target computer.

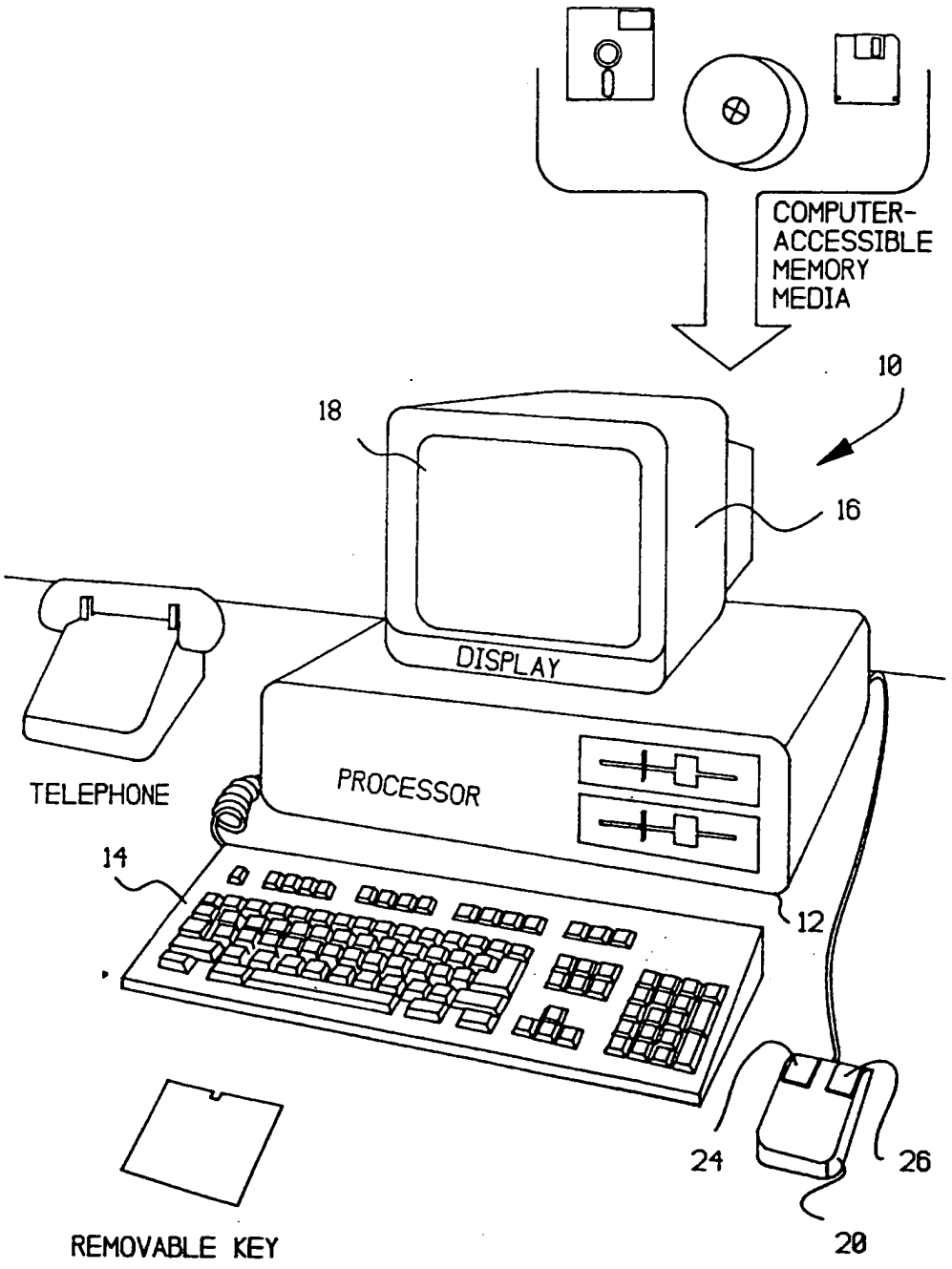
6. A method of passing encrypted files between data processing systems, comprising:
at a source computer providing at least one file which is encrypted with a key which is at least partially derived from at least one unique source computer system attribute;
providing a transfer memory medium;
initiating a particular transfer operation;
at said source computer, decrypting said at least one file;
including in said at least one file a transfer identifier which uniquely identifies said particular transfer operation;
at said source computer, encrypting said at least one file with a key which is derived from at least one unique transfer memory media attribute;
at said source computer, copying said encrypted file to said transfer memory media;
at a target computer, decrypting said at least one file;
at said target computer, encrypting said at least one file with a key which is at least partially derived from at least one target computer system attribute.

7. A method of passing encrypted files according to Claim 6, further comprising:
at said target computer, passing a unique target computer identification to said transfer memory media.

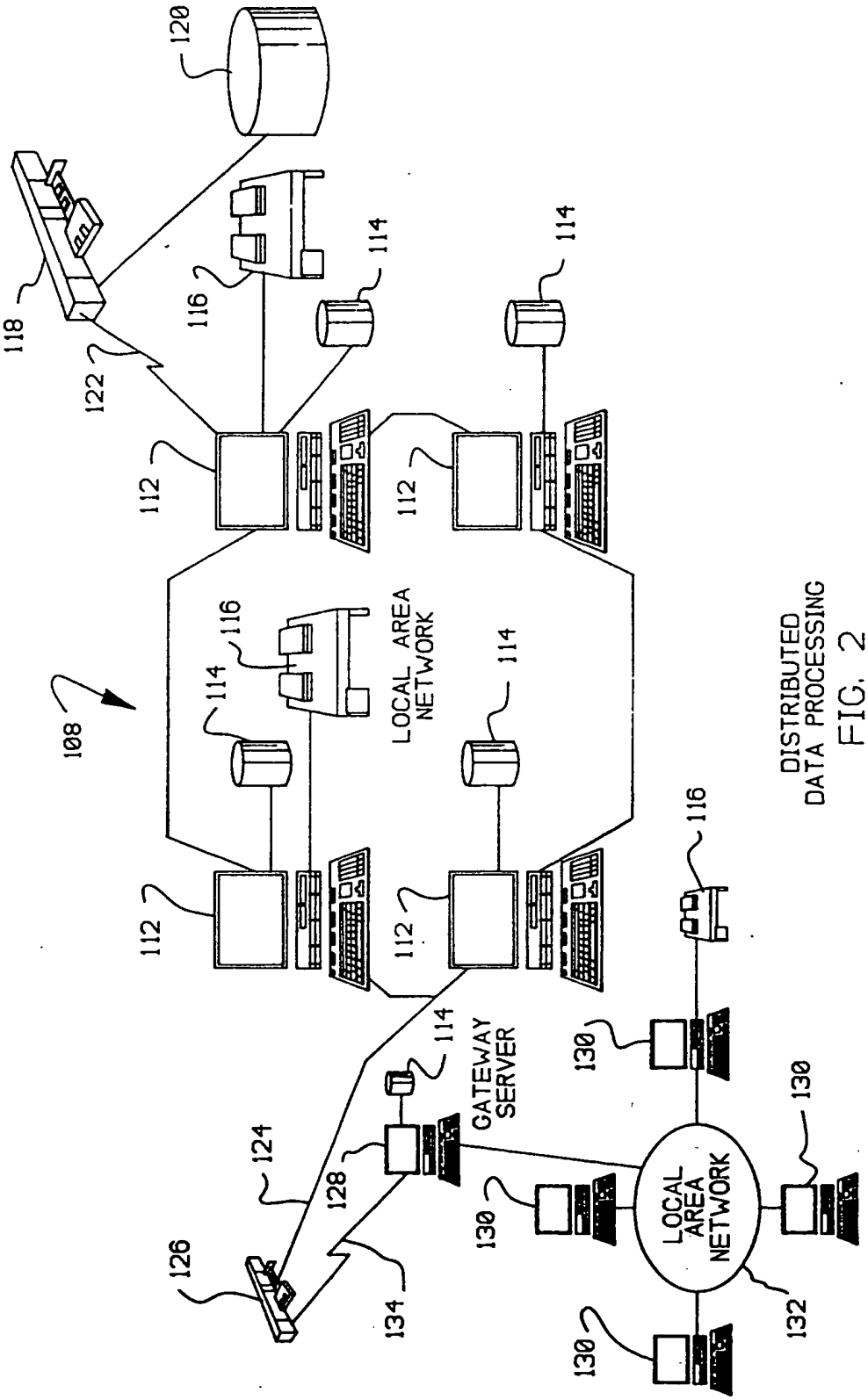
8. A method of passing encrypted files according to Claim 6 or 7, further comprising:
at said target computer, updating said at least one file to provide an identification of said source computer.

9. An apparatus passing encrypted files between data processing systems, comprising:
at least one file in a source computer which is encrypted with a key which is at least partially derived from at least one unique source computer system attribute;
a removable transfer memory medium having a unique attribute;
an export program for decrypting said at least one file and encrypting said at least one file with a key which is derived from said unique attribute and copying said encrypted file to said transfer memory media;
an import program at a target computer for decrypting said at least one file, and encrypting said at least one file with a key which is at least partially derived from at least one target computer system attribute.

10. An apparatus for passing encrypted files according to Claim 9, further comprising:
a export counter in said export program in said source computer which defines a maximum number of permissible transfer operations, and for counting each transfer operation.



STAND ALONE PC
FIG. 1



DISTRIBUTED DATA PROCESSING

FIG. 2

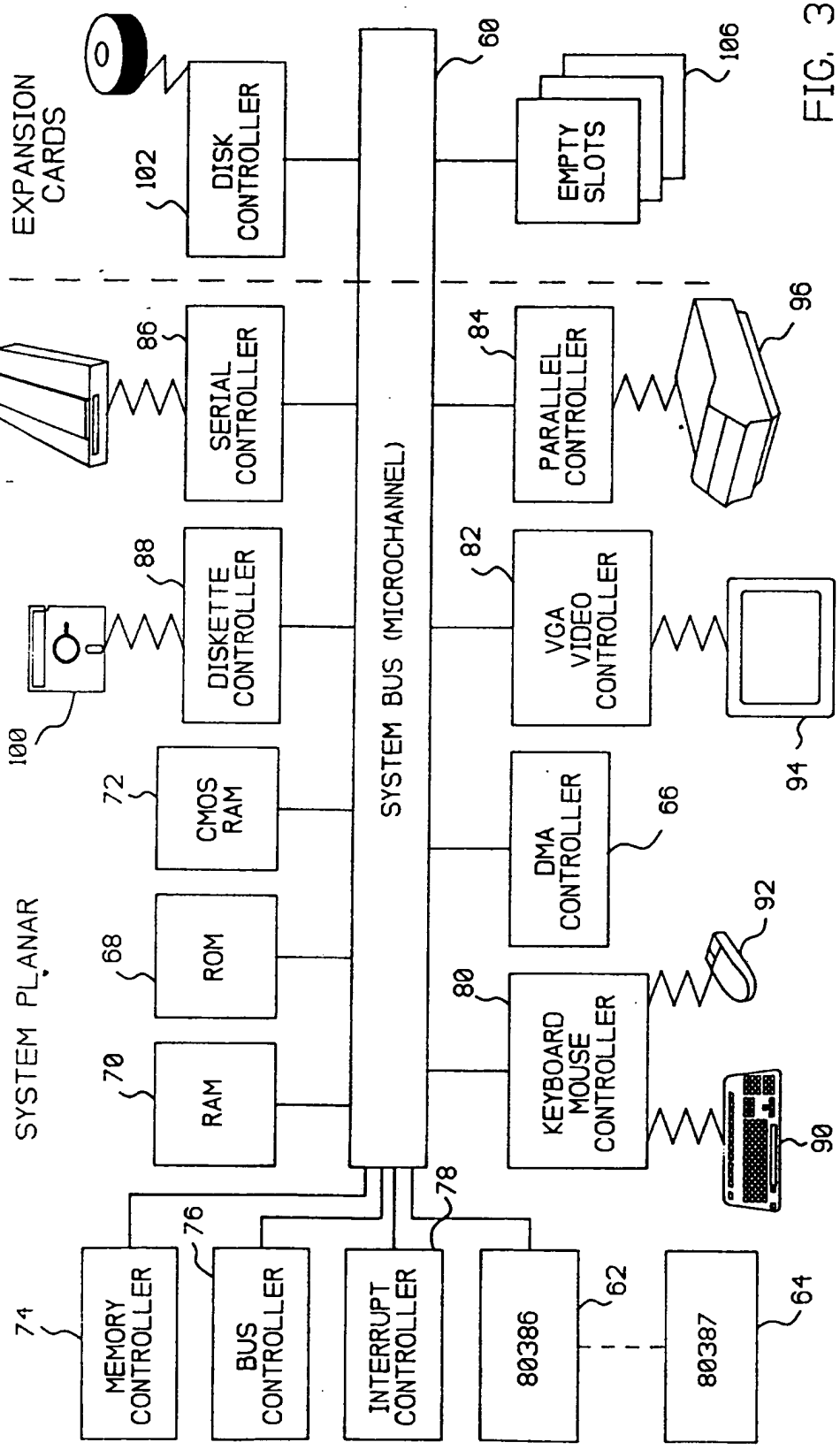


FIG. 3

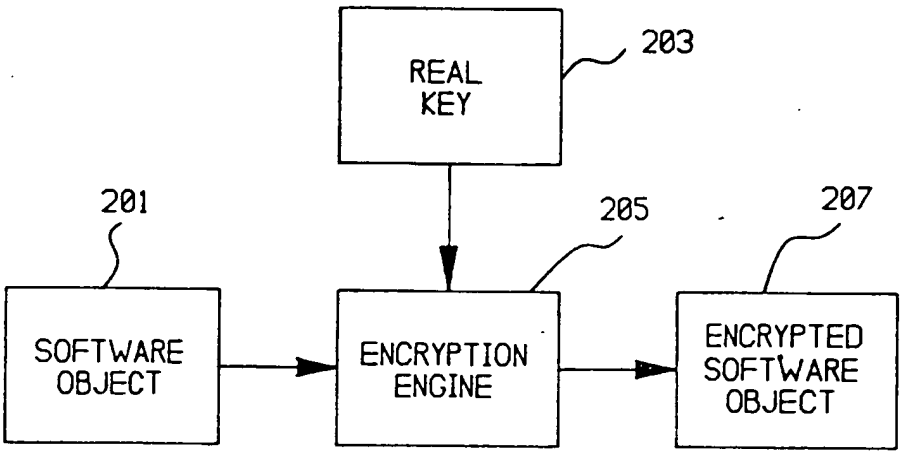


FIG. 4

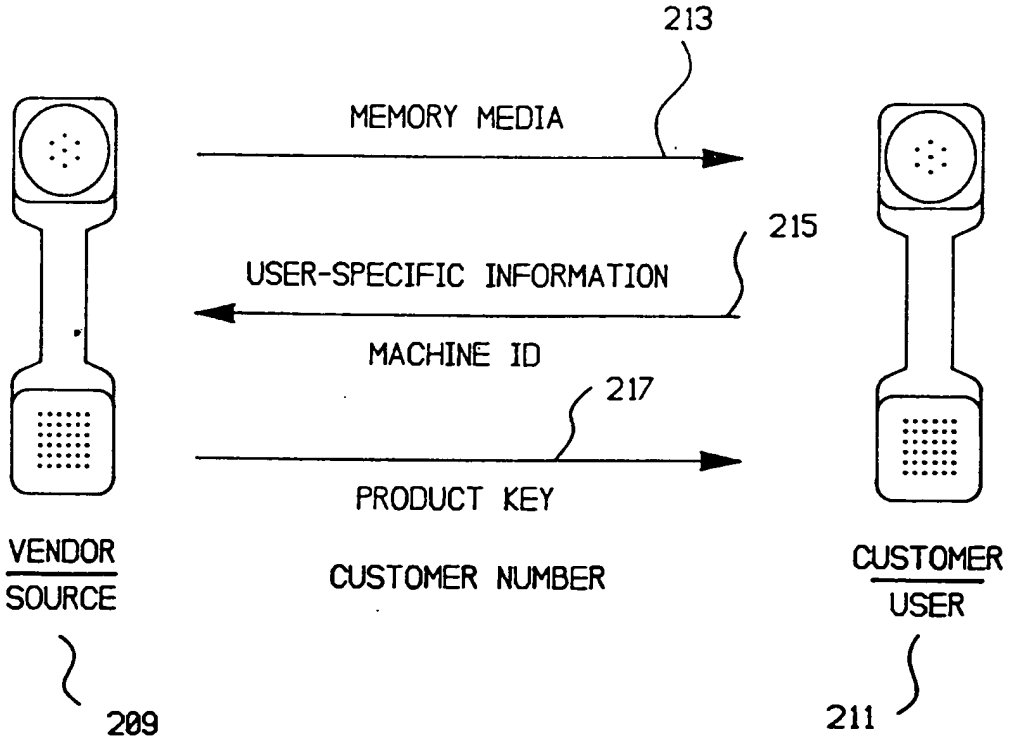
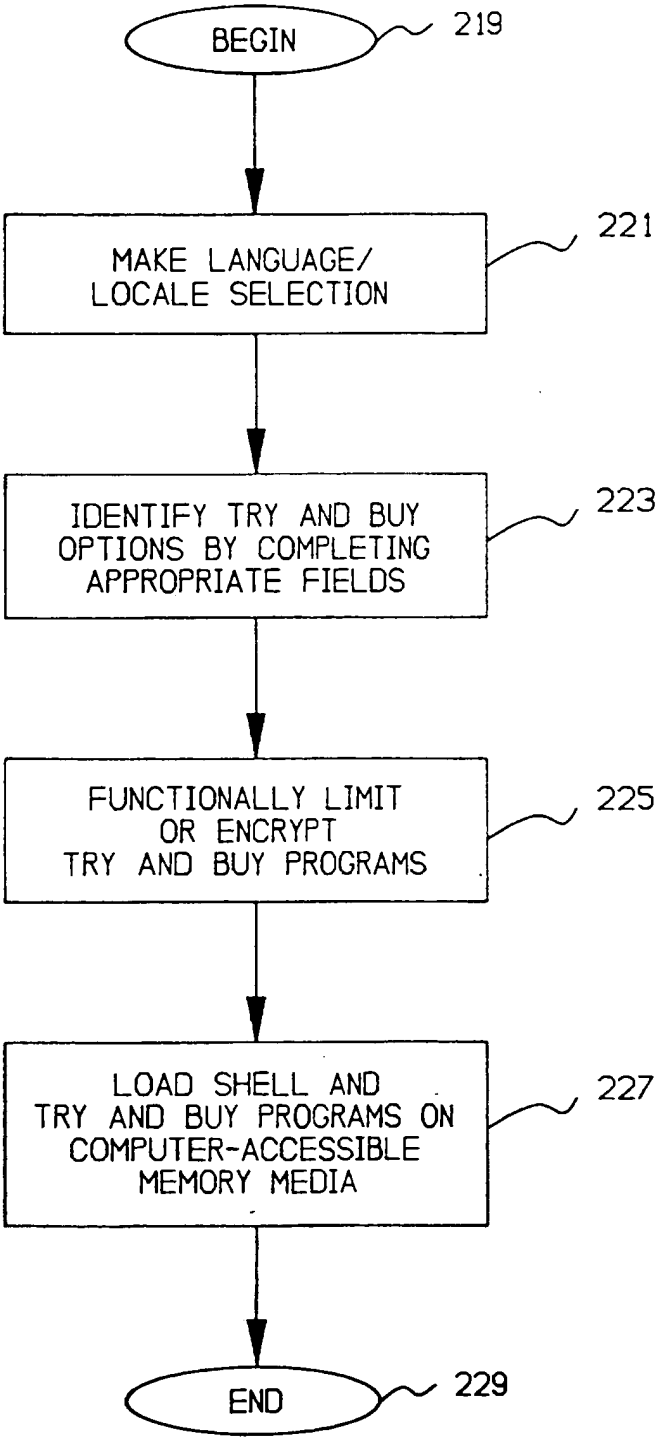


FIG. 5



BUILDING THE SHELL

FIG. 6

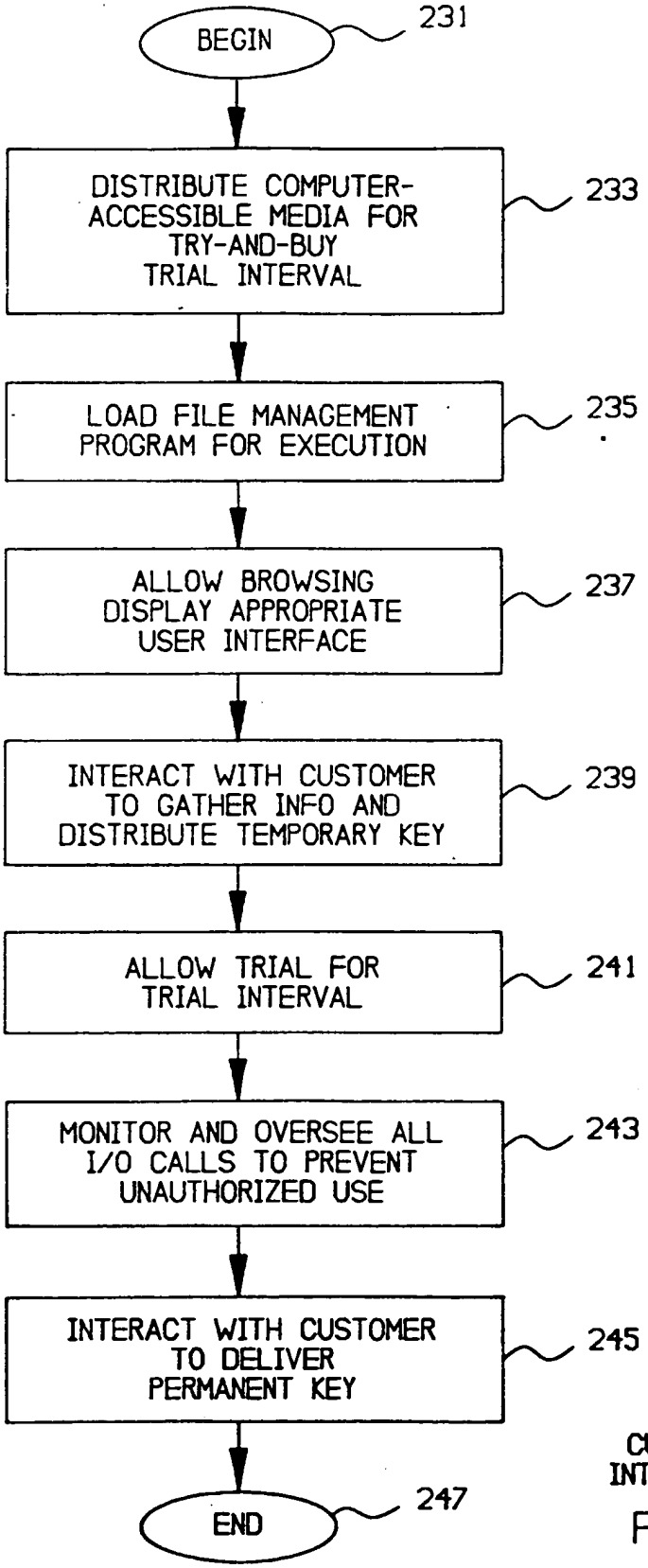


FIG. 7

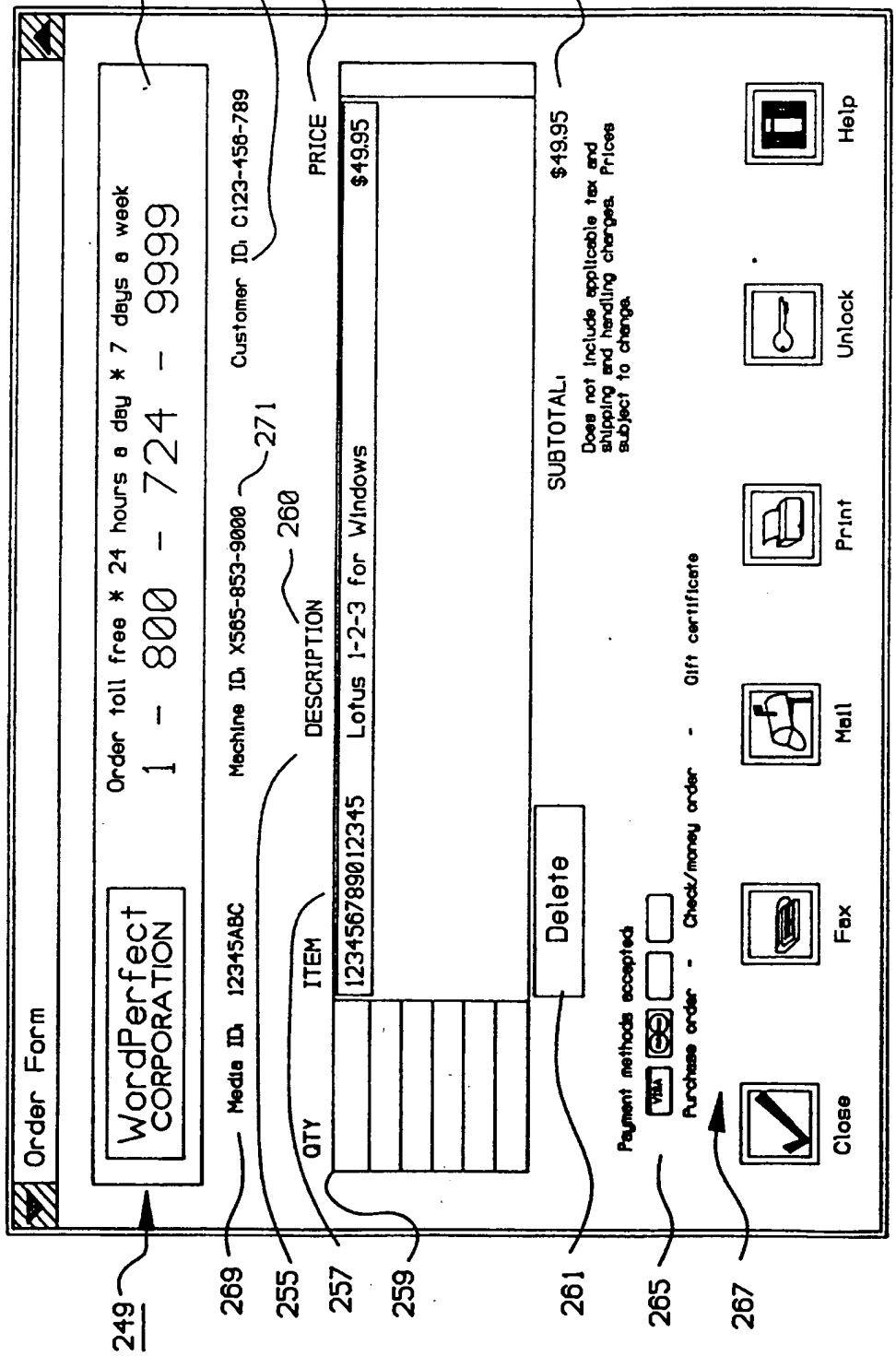


FIG. 8

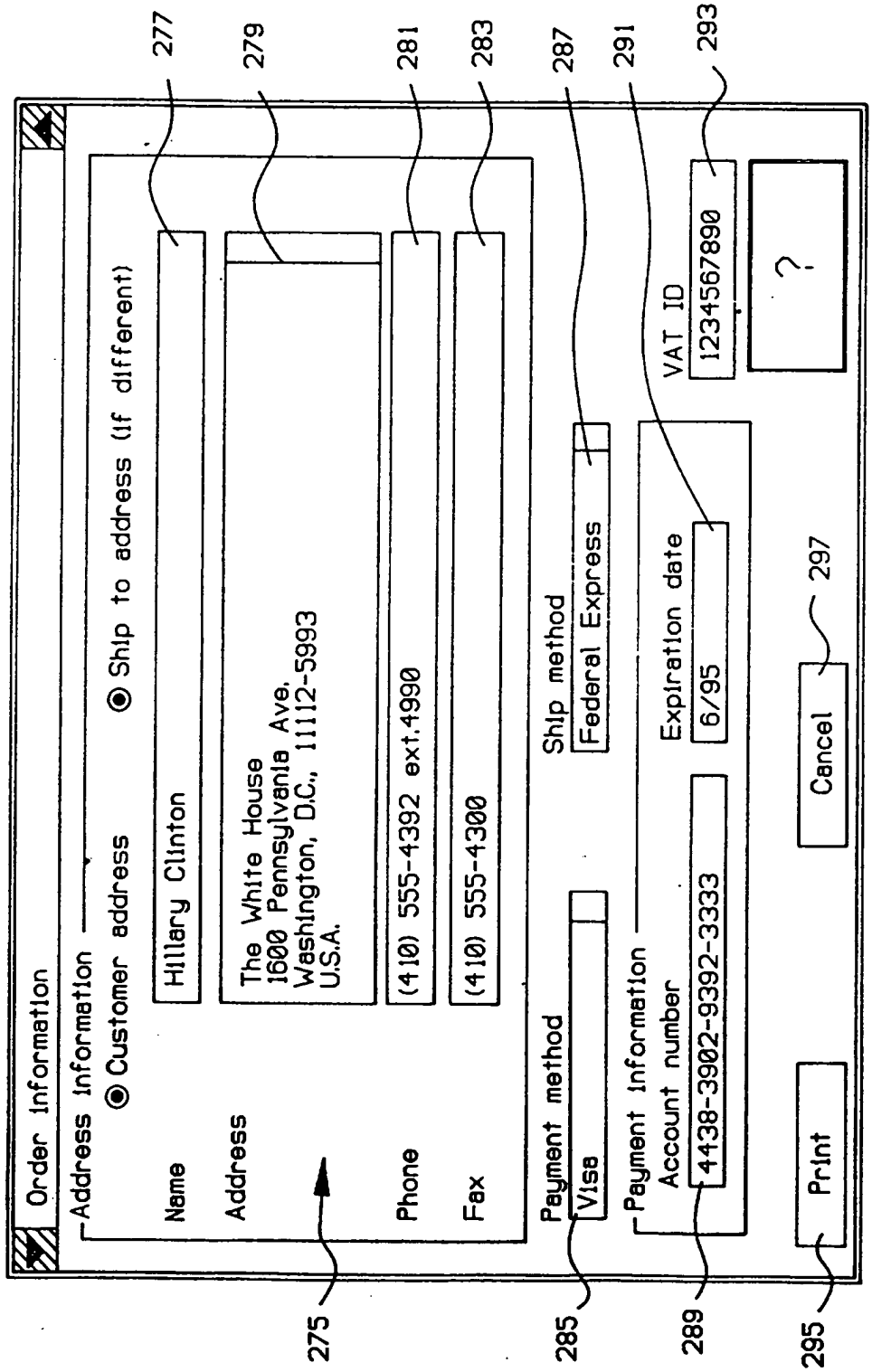


FIG. 9

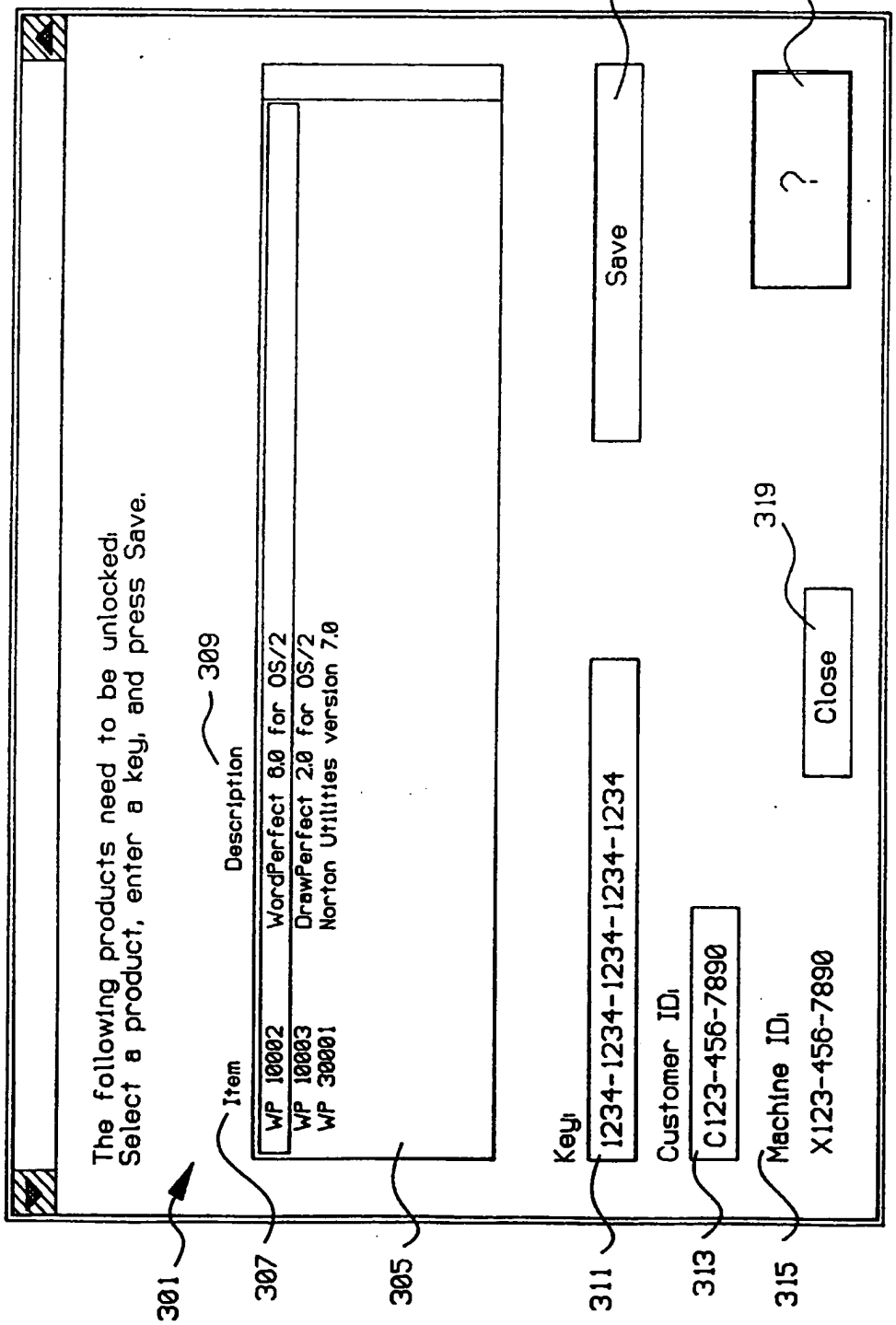


FIG. 10A

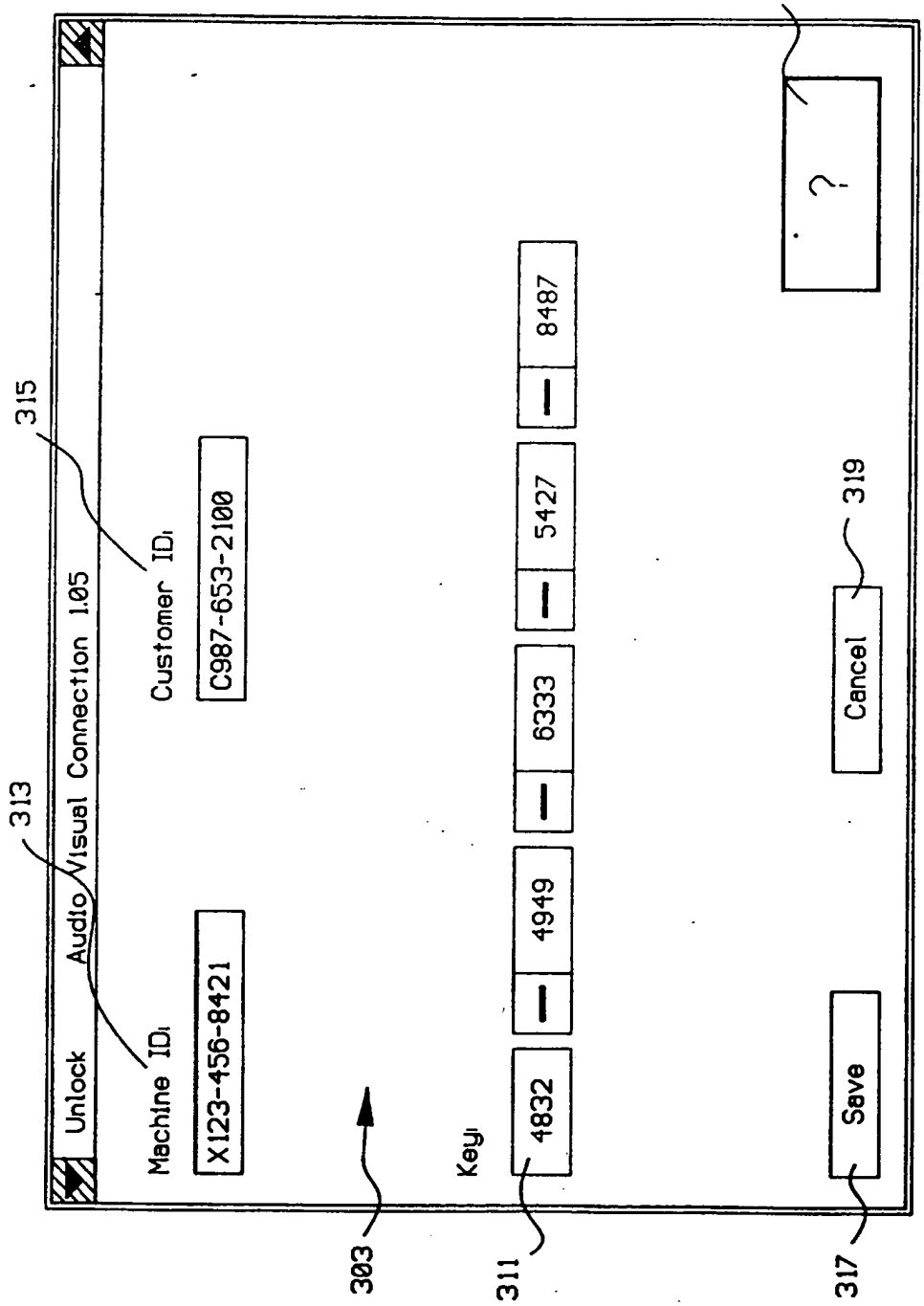


FIG. 10B

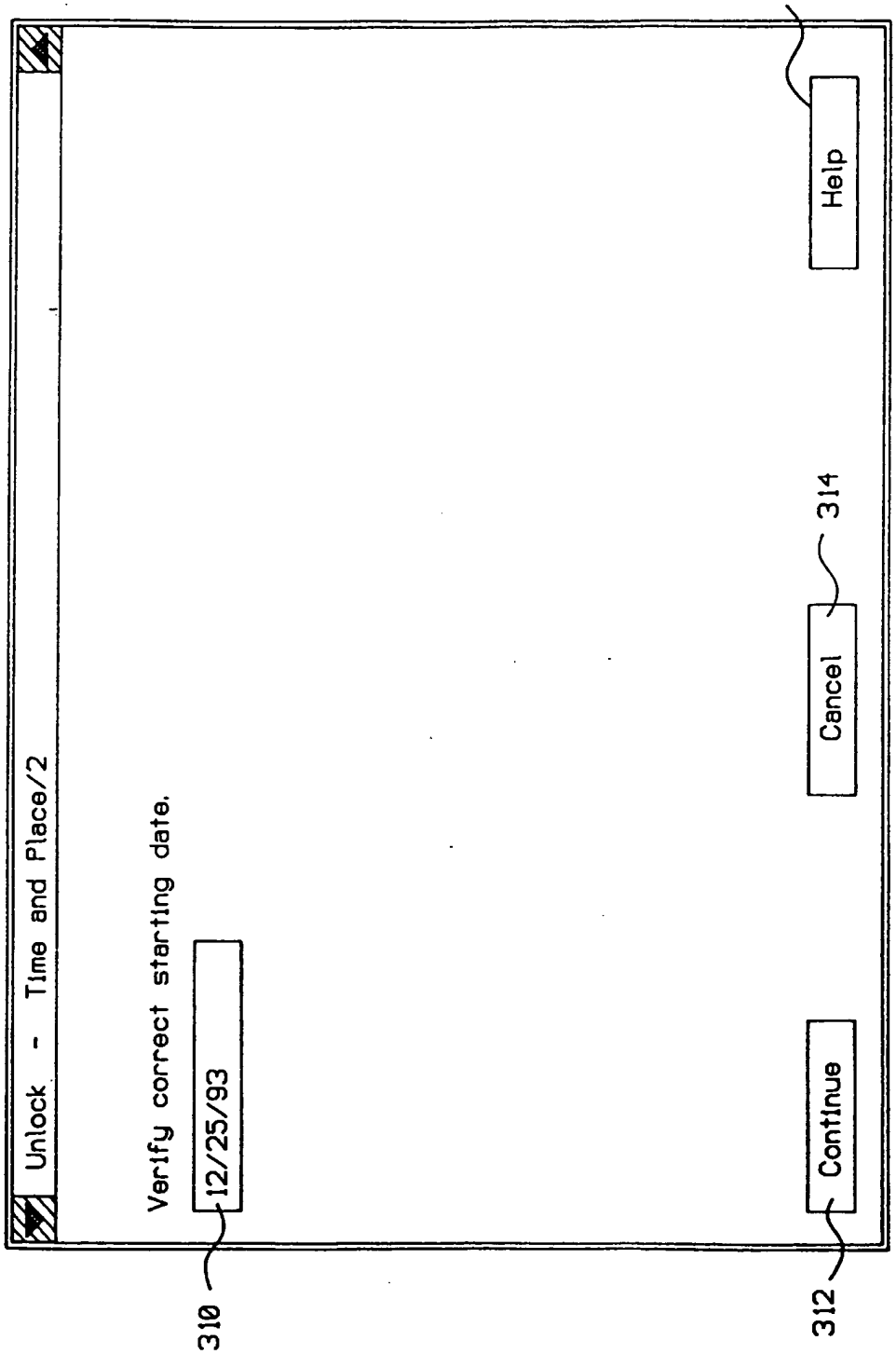


FIG. 11

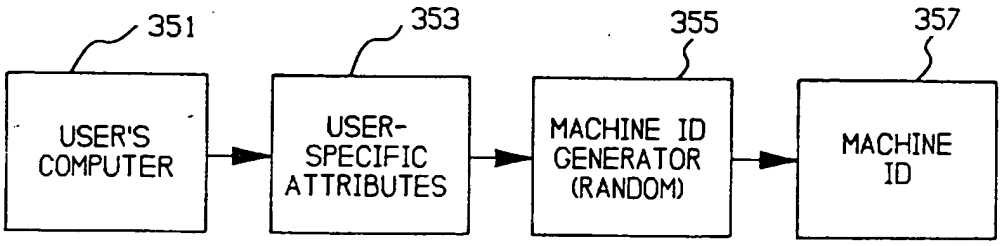
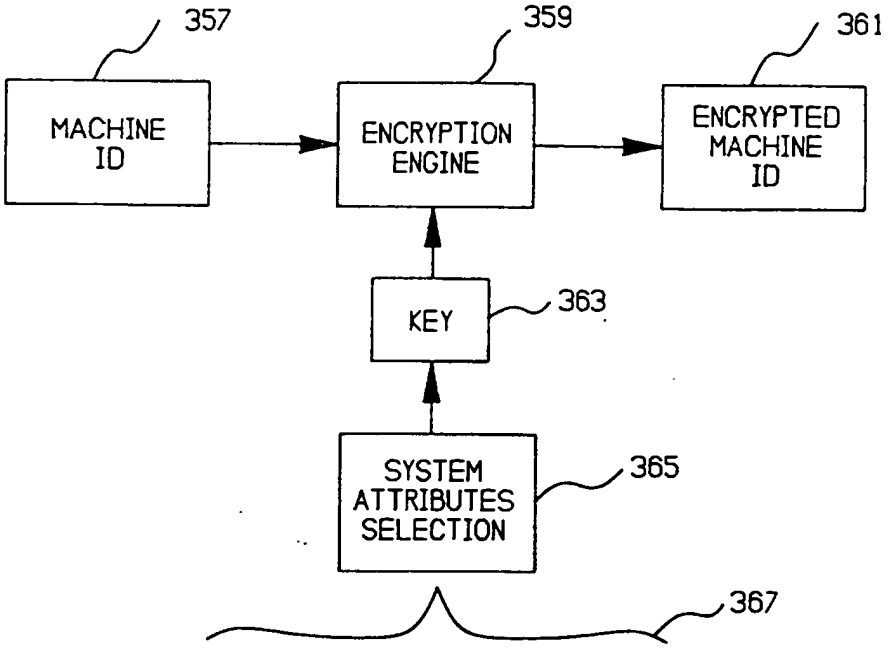
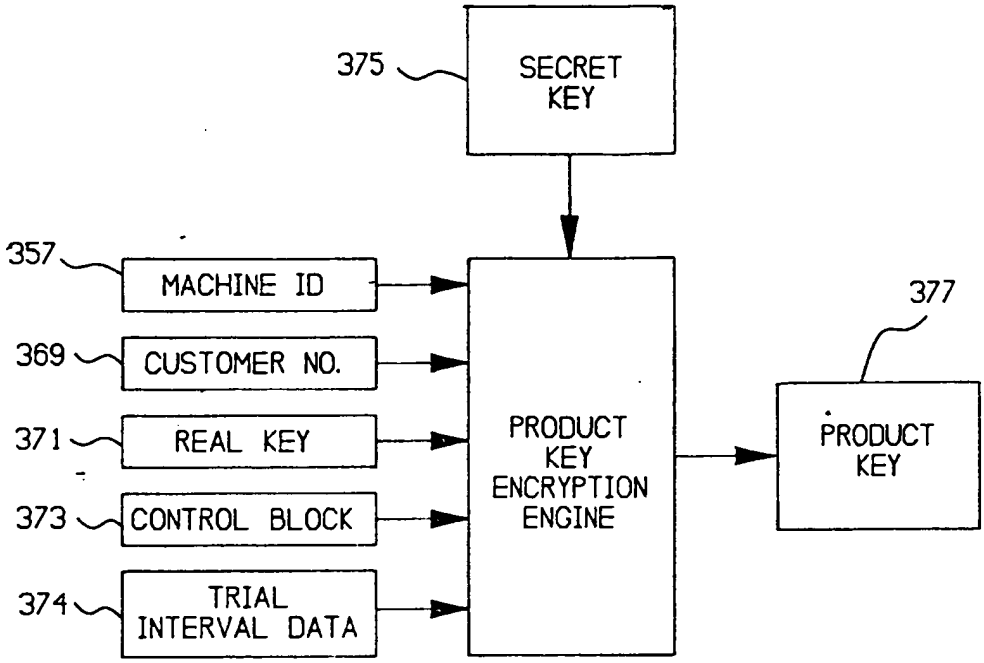


FIG. 12

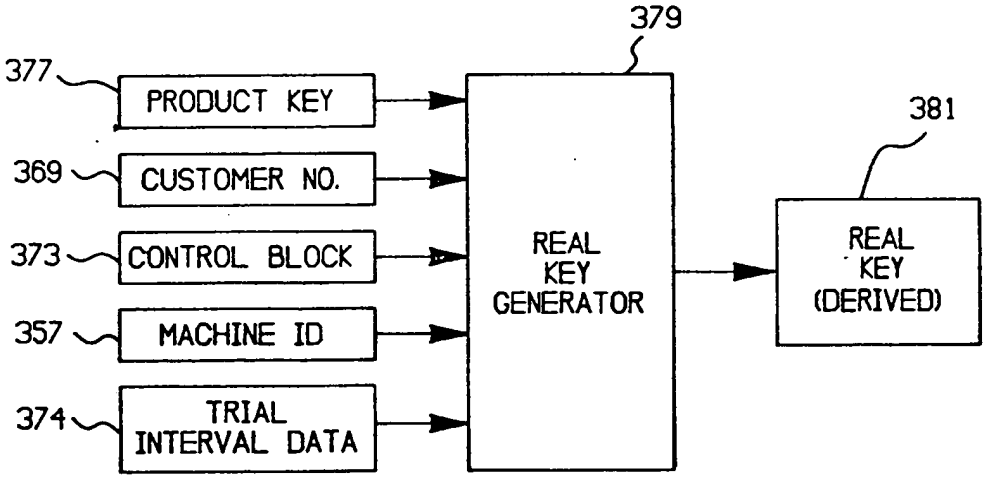


1. HARD DISK SERIAL NO.
2. SIZE OF HARD DISK
3. FORMAT OF HARD DISK
4. SYSTEM MODEL NO.
5. HARDWARE INTERFACE CARD
6. HARDWARE SERIAL NO.
7. CONFIGURATION PARAMETERS

FIG. 13



GENERATION OF PRODUCT KEY
FIG. 14



VALIDATION OF PRODUCT KEY
FIG. 15

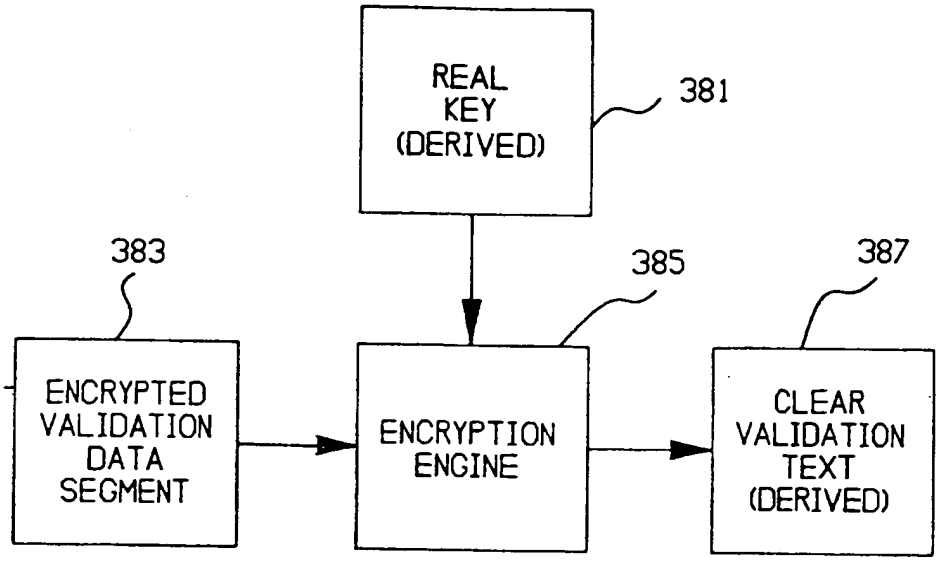


FIG. 16

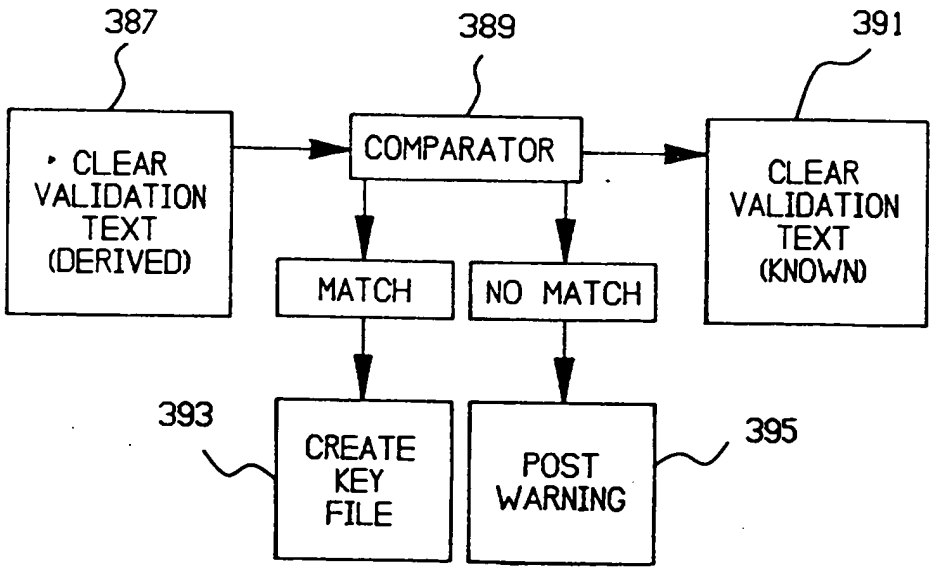


FIG. 17

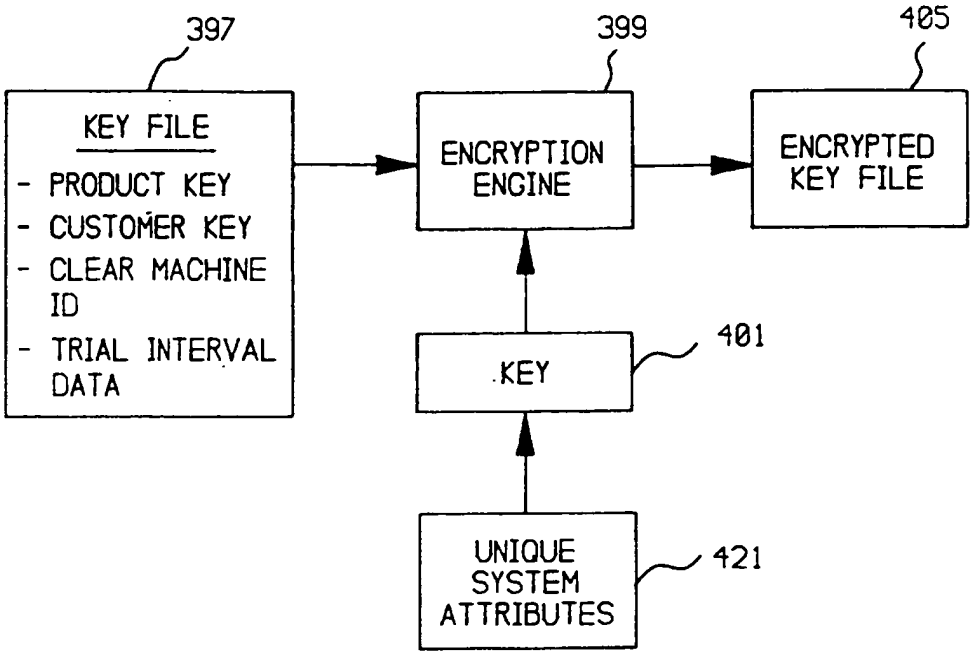


FIG. 18

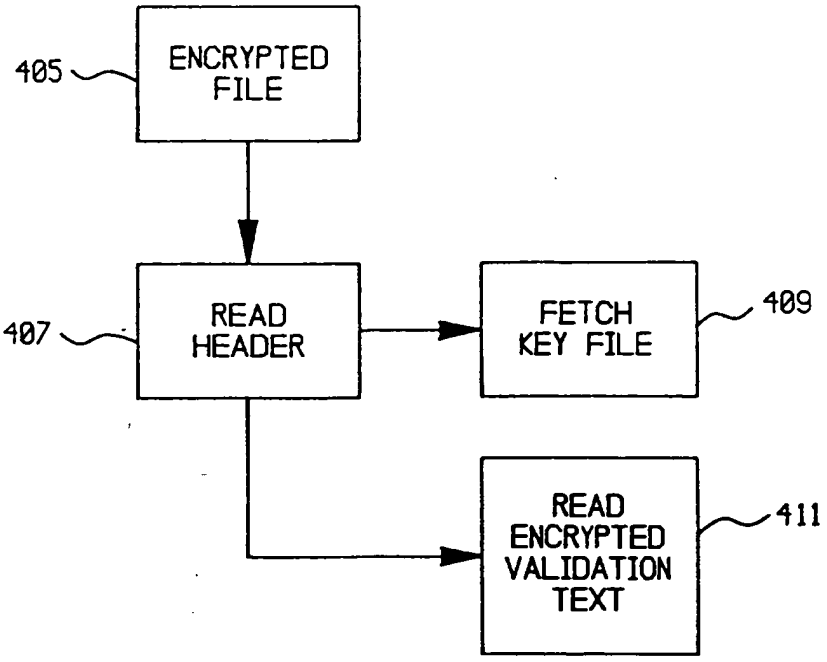


FIG. 19

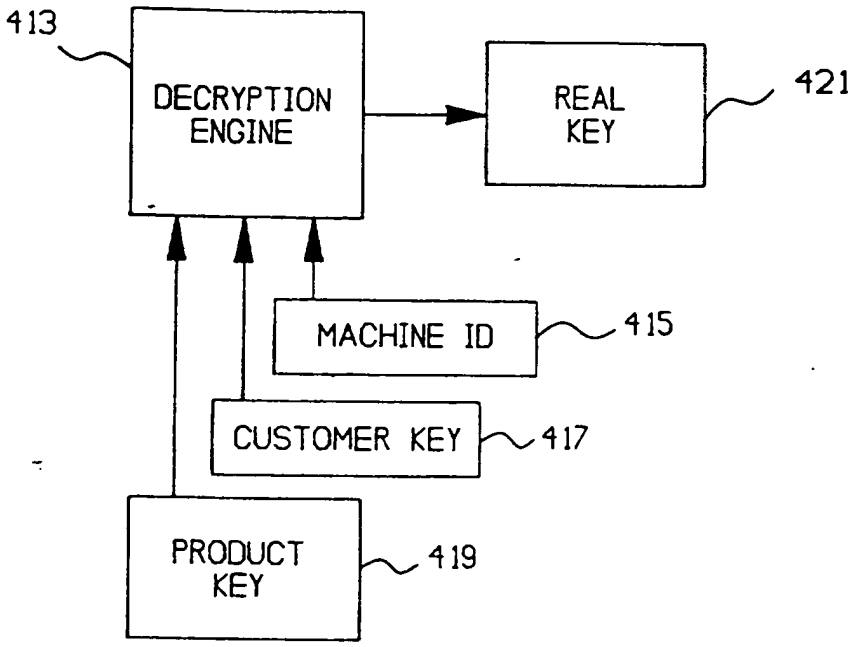


FIG. 20

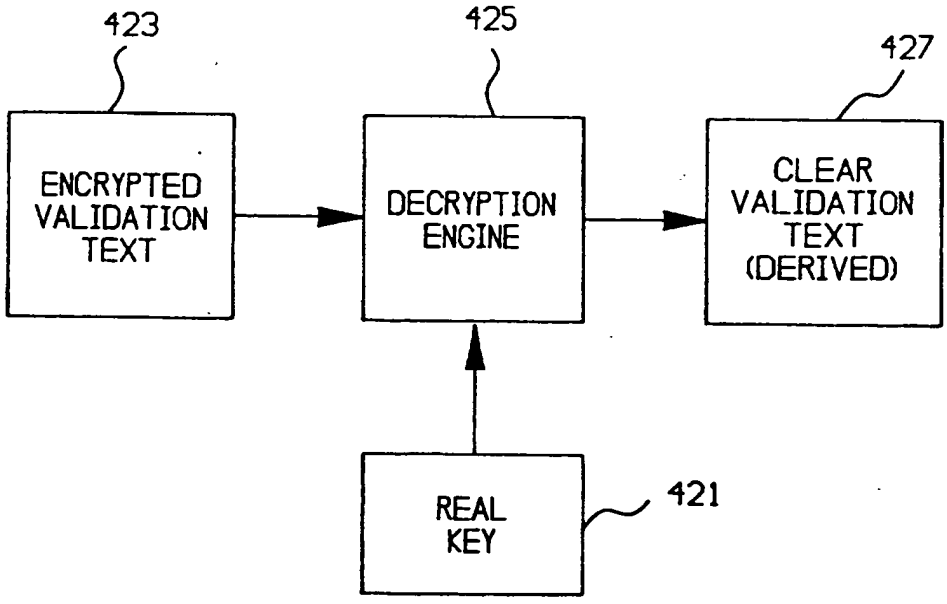


FIG. 21

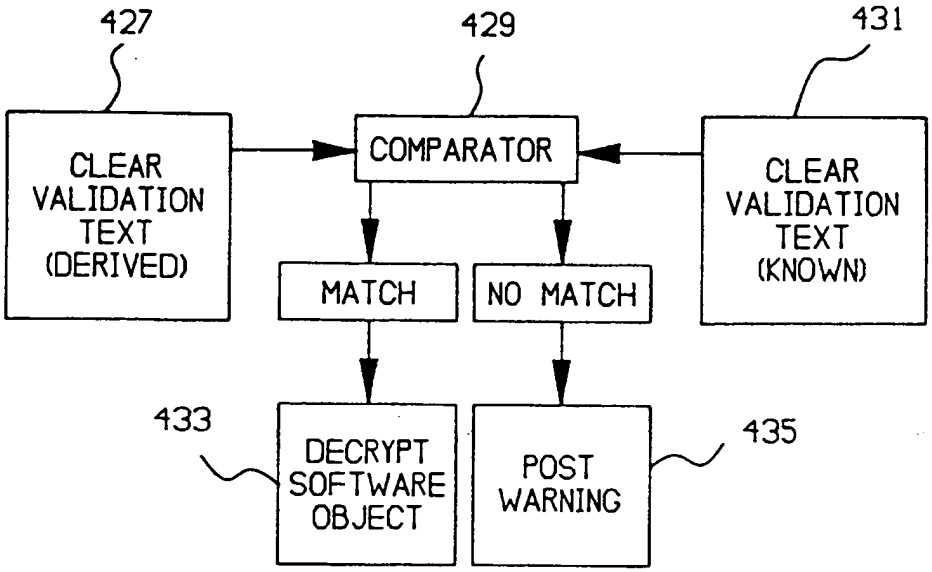


FIG. 22

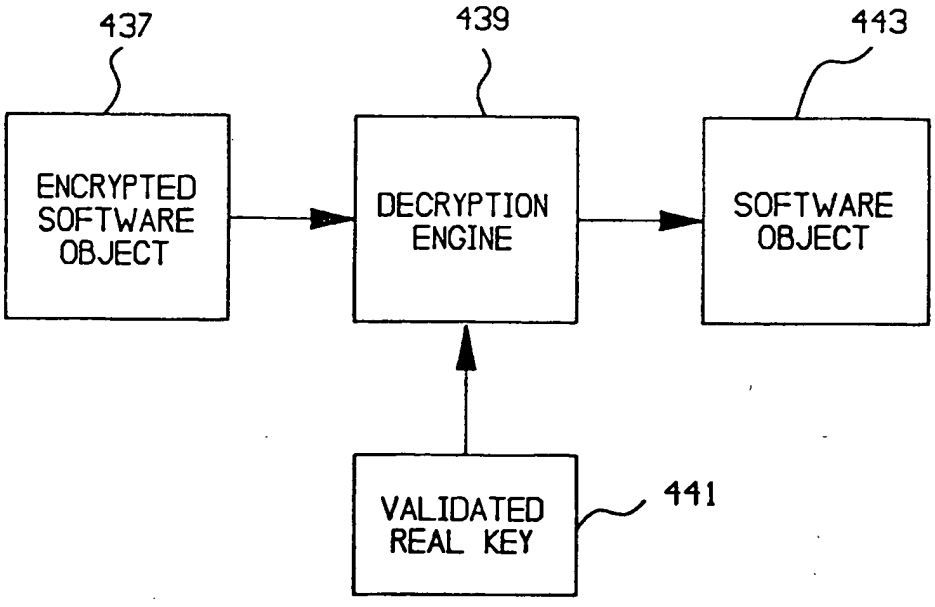
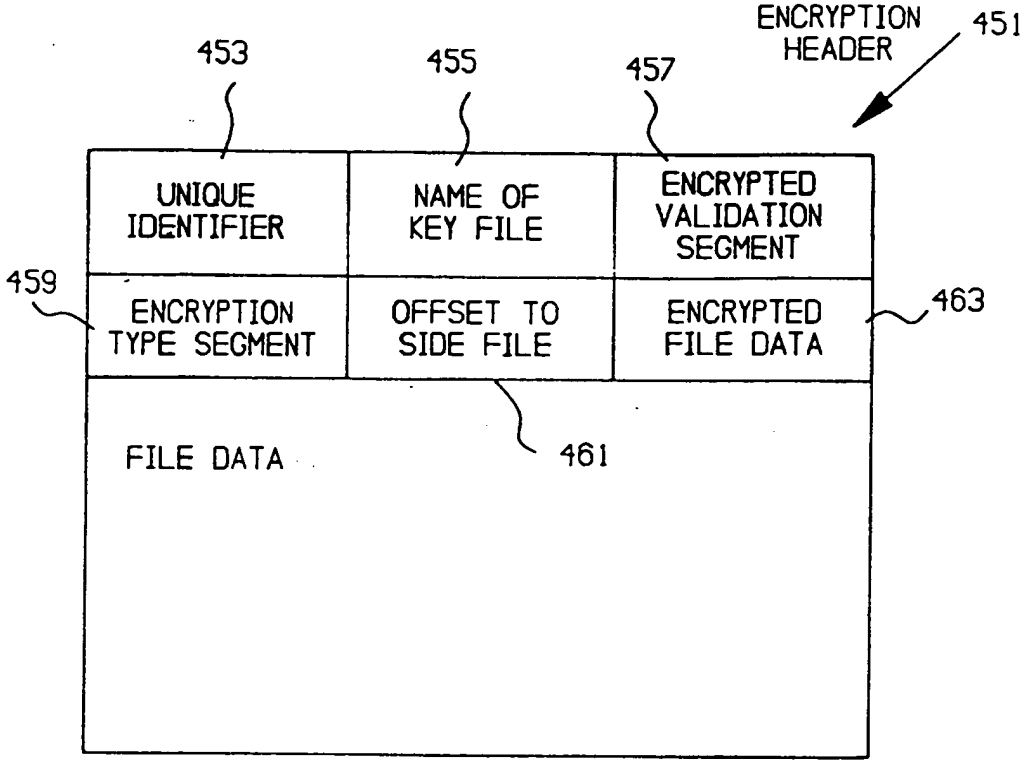


FIG. 23



ENCRYPTED FILE

FIG. 24

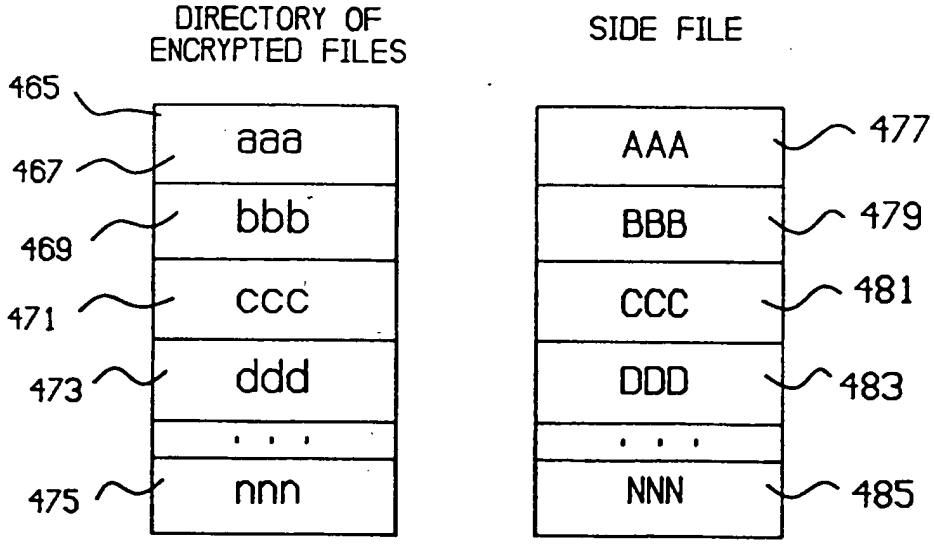


FIG. 25

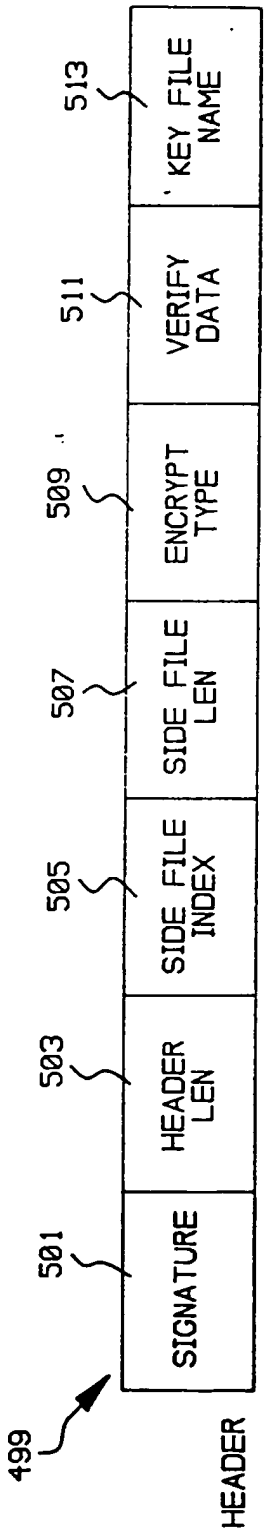


FIG. 26

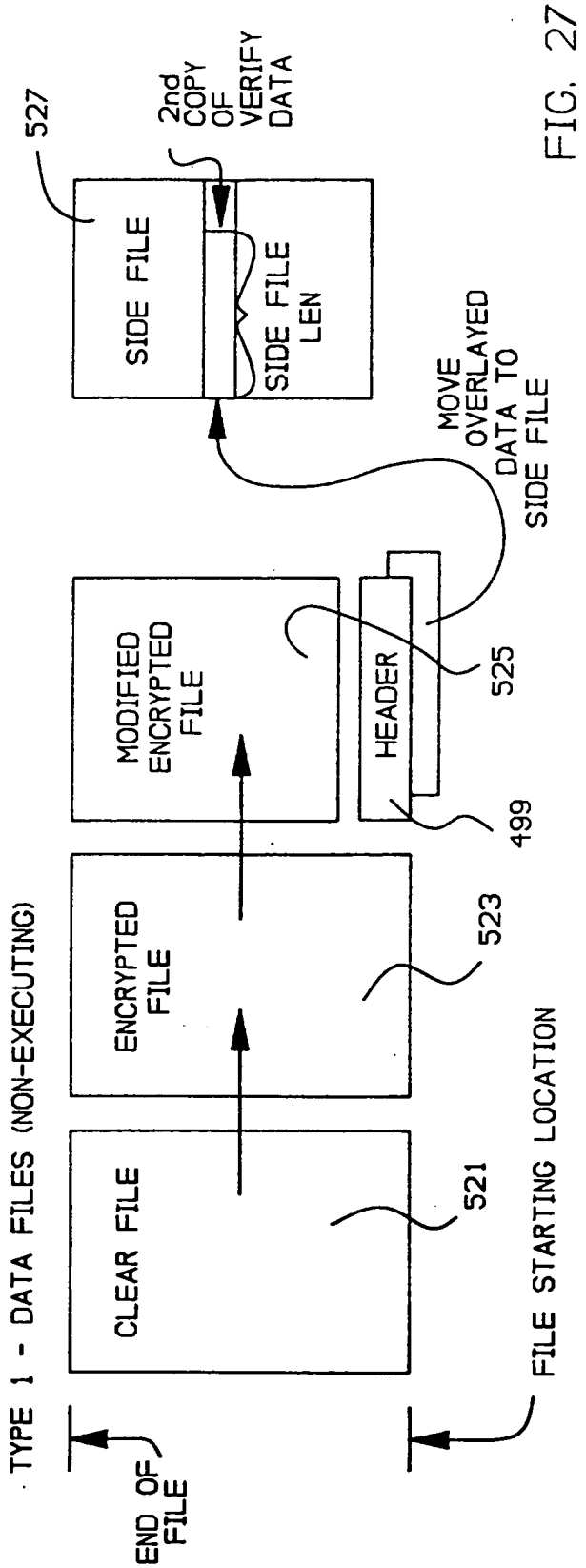


FIG. 27

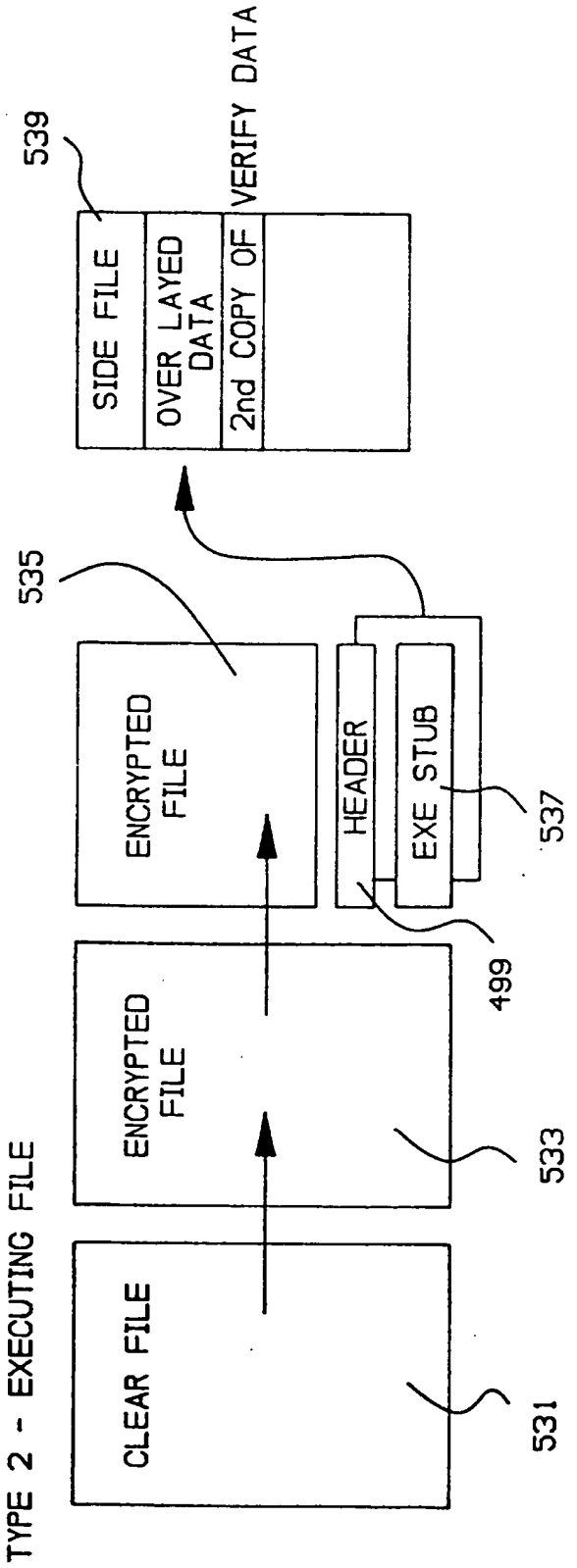
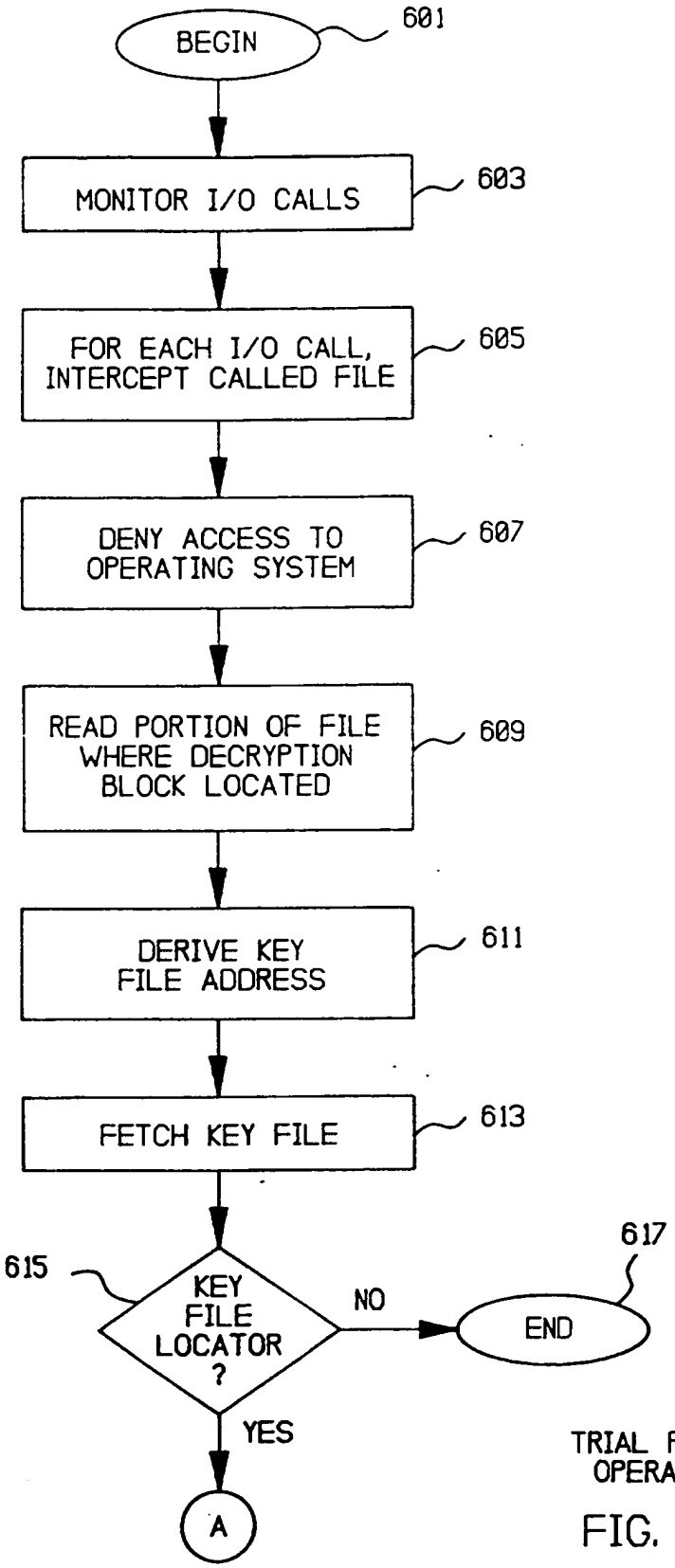
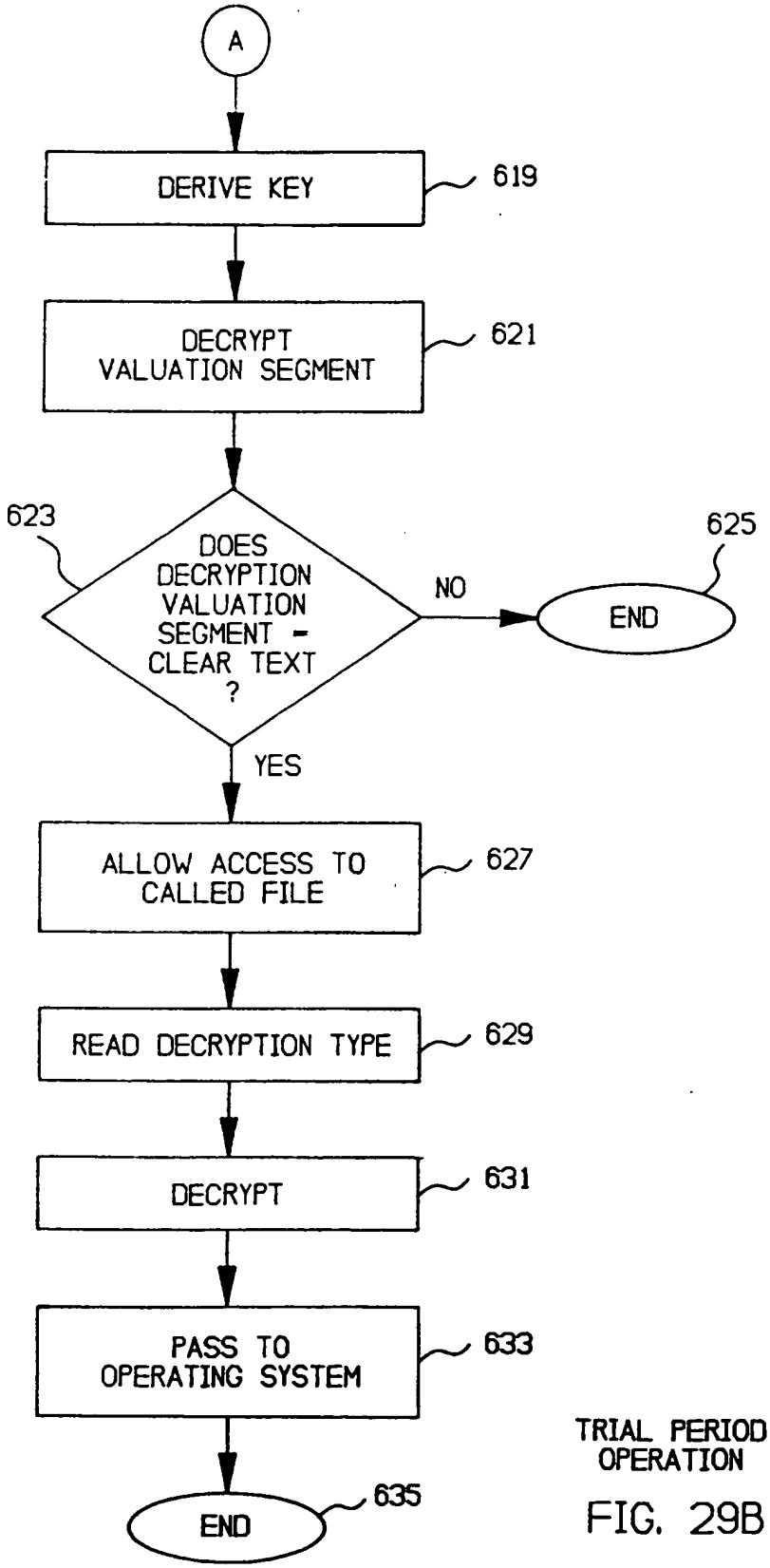


FIG. 28



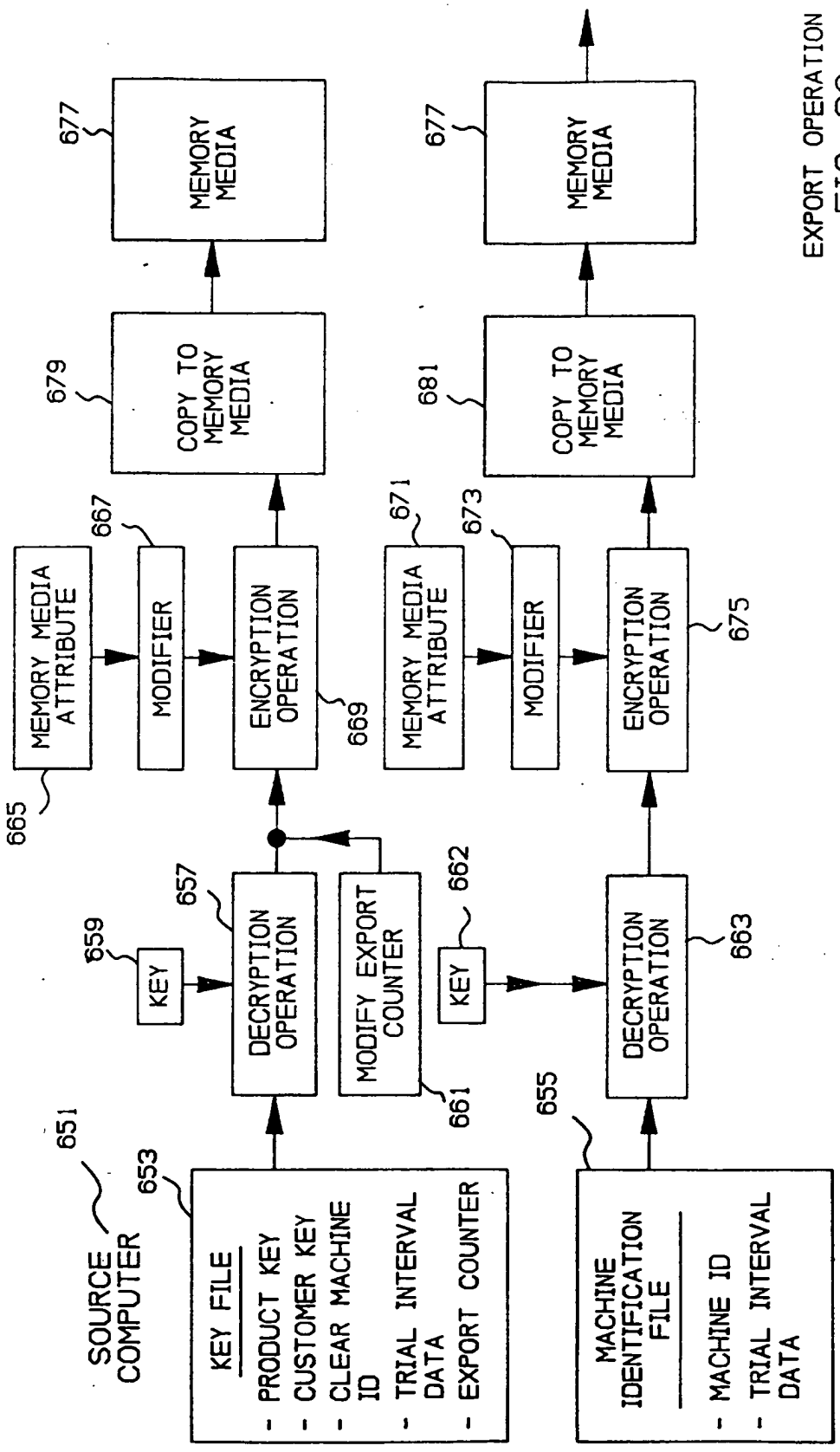
TRIAL PERIOD OPERATION

FIG. 29A



TRIAL PERIOD OPERATION

FIG. 29B



EXPORT OPERATION
FIG. 30

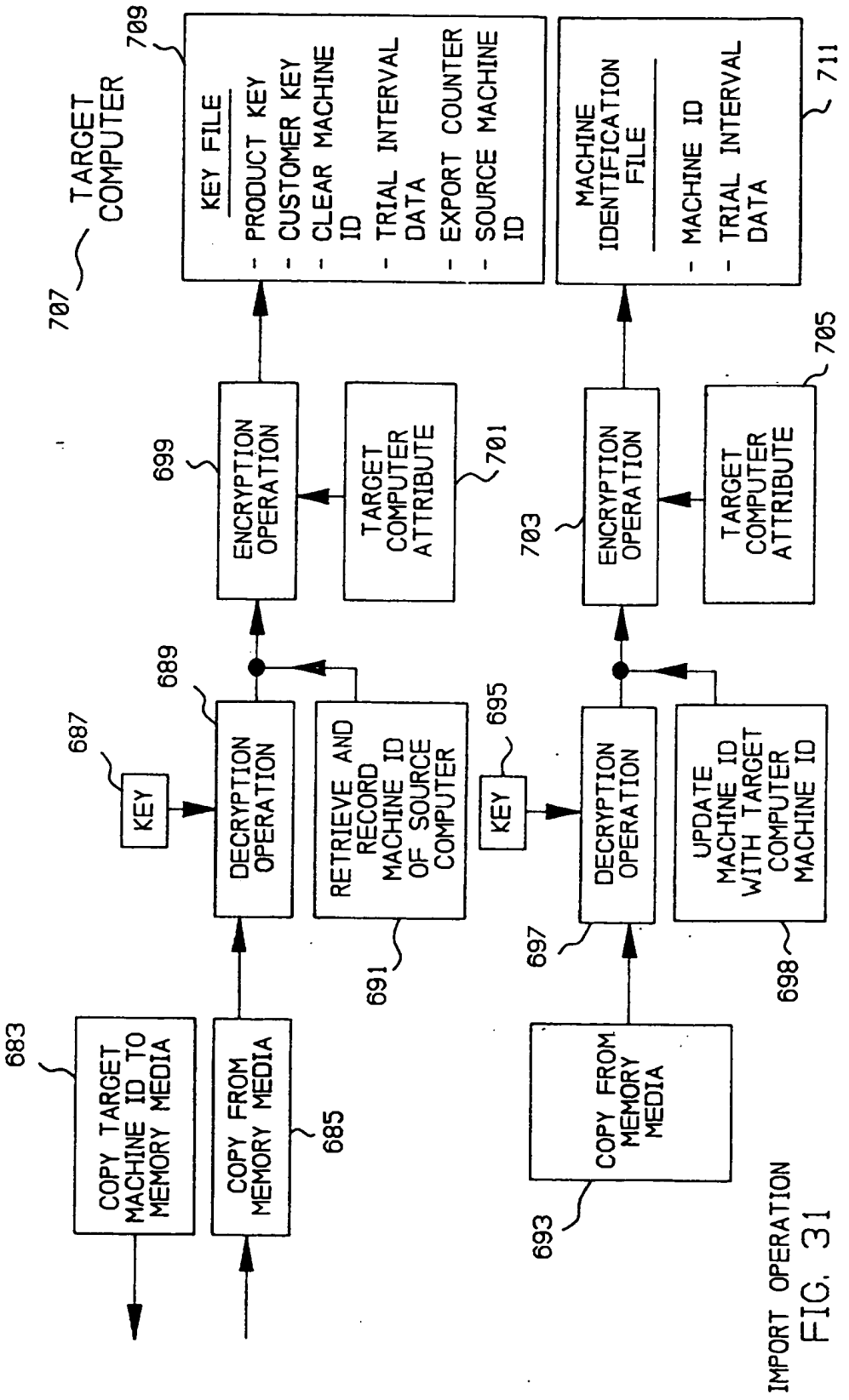


FIG. 31

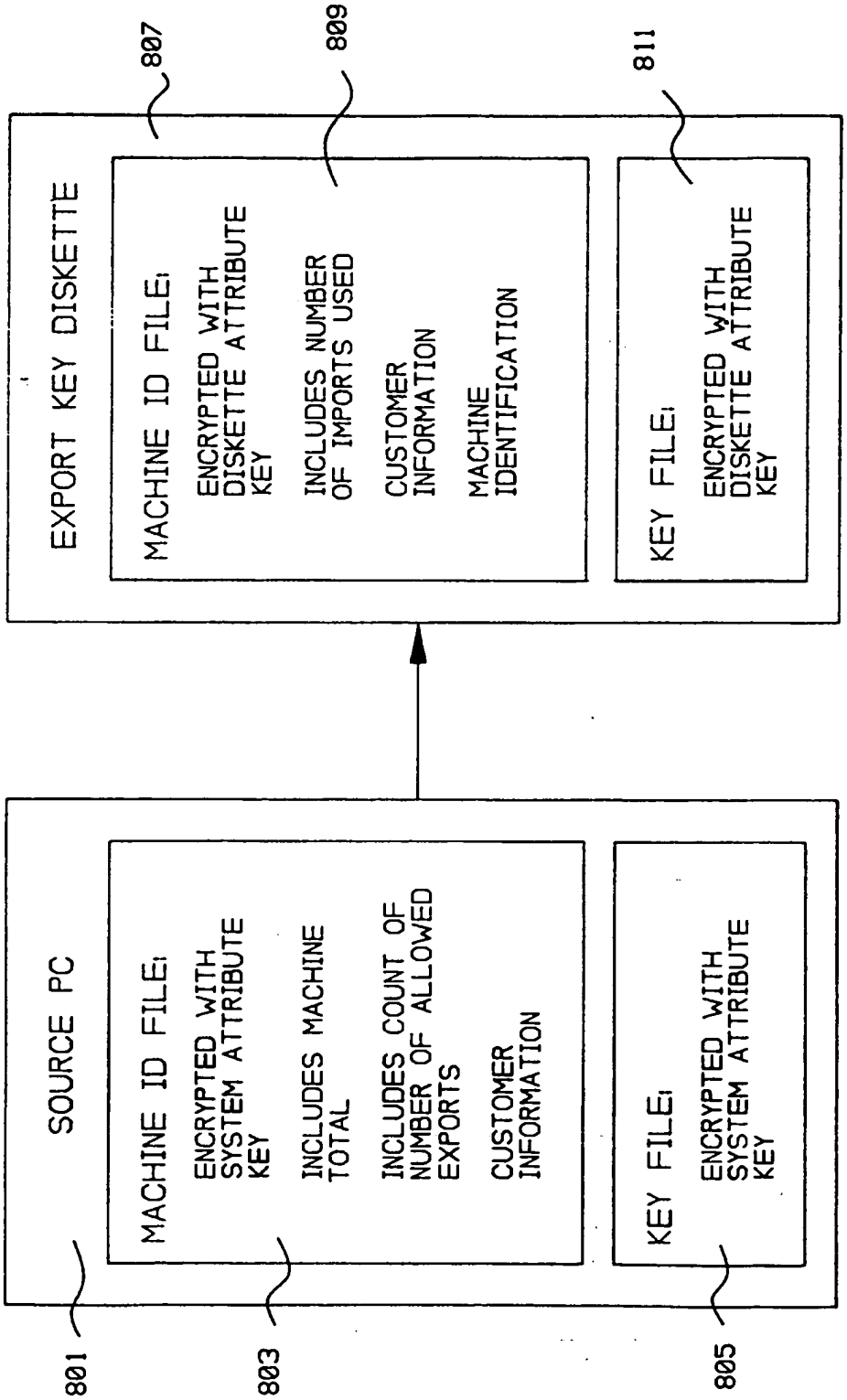


FIG. 32

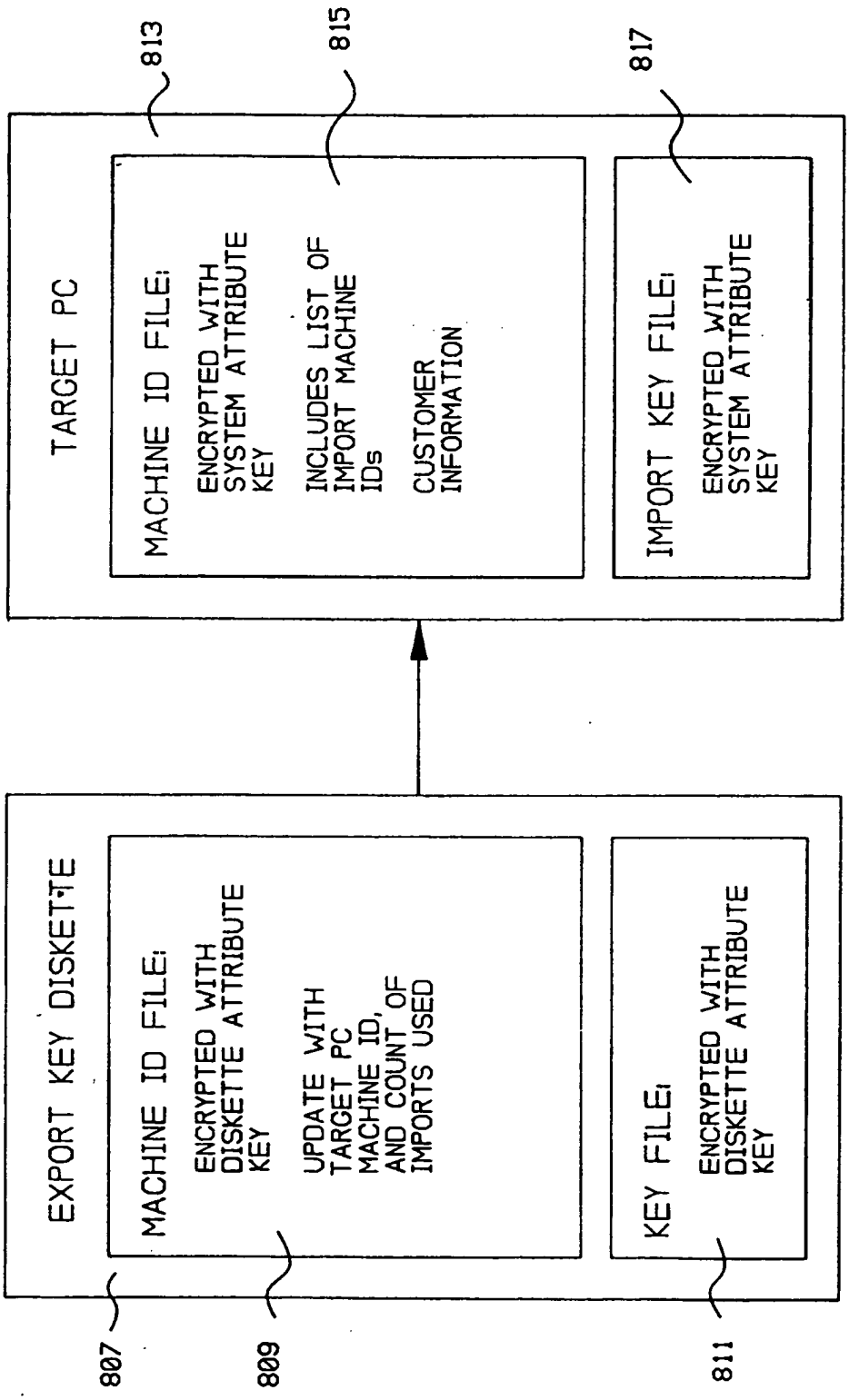
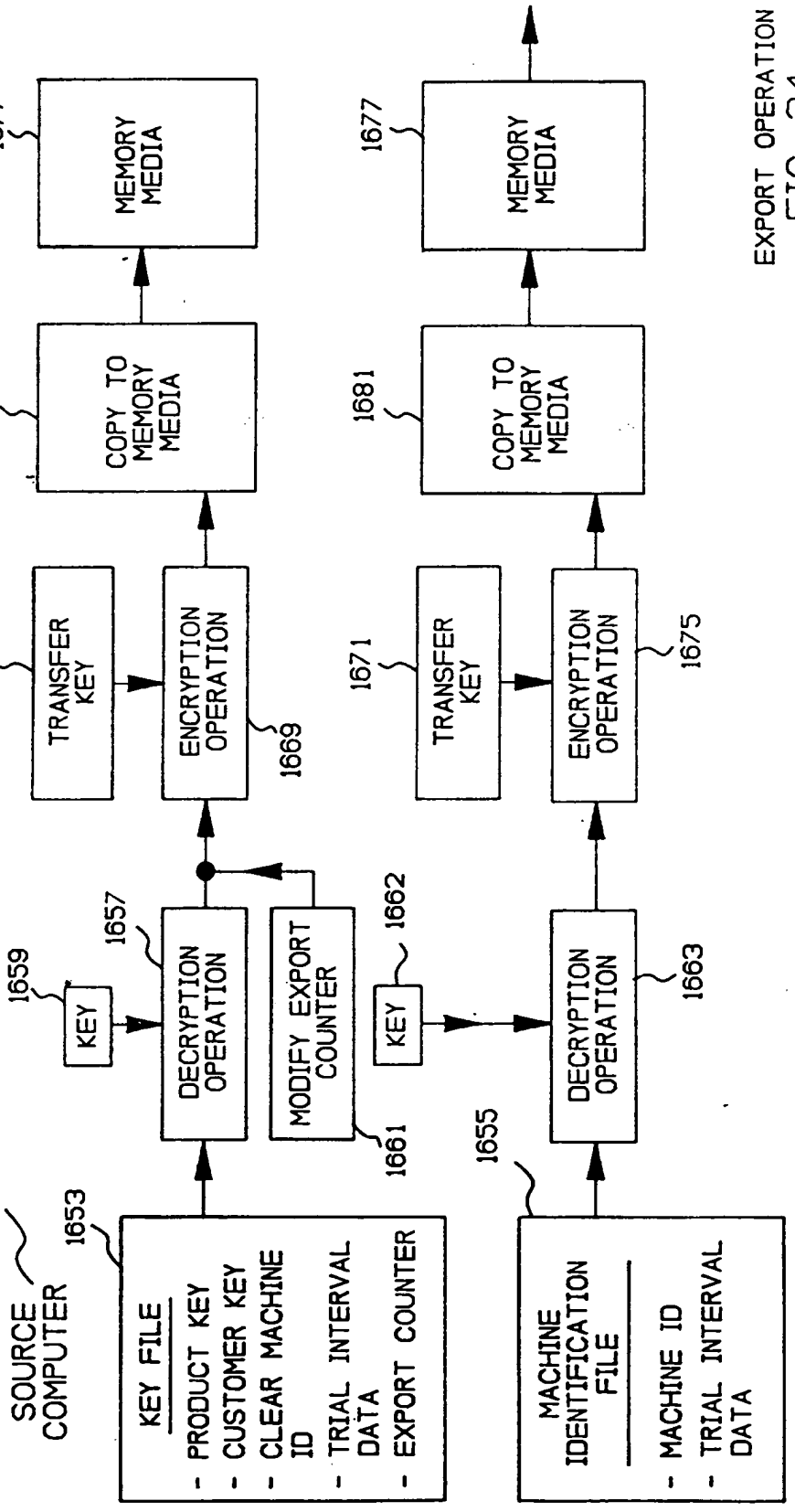


FIG. 33



EXPORT OPERATION
FIG. 34

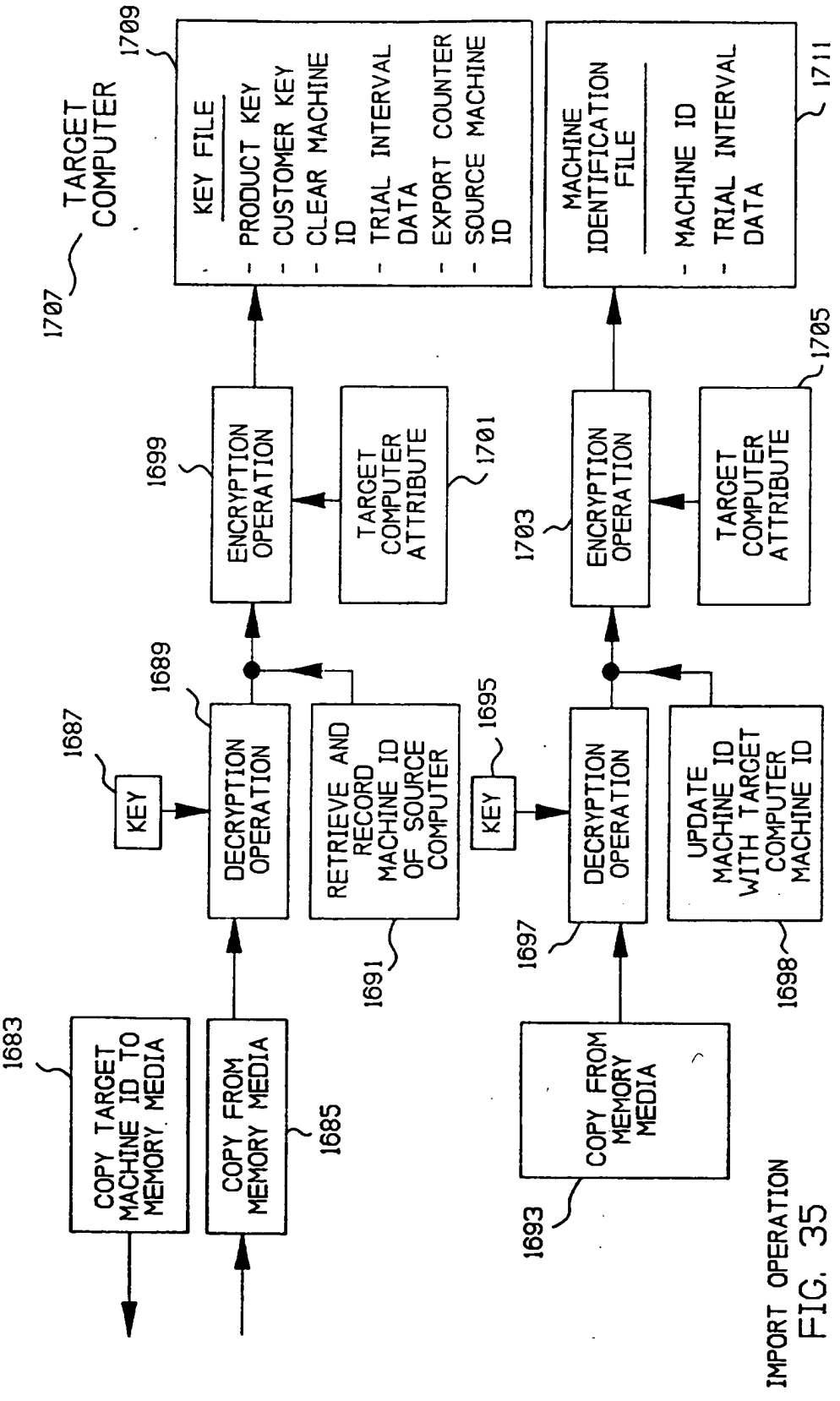


FIG. 35

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 95 10 5400

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
X	WO-A-94 07204 (UNILOC) * abstract; figures 41,2,8 * * page 6, line 11 - page 9, line 5 * * page 10, line 3 - line 10 * * page 12, line 7 - page 17, line 13 *	1,9	G06F1/00 G06F12/14
Y	---	2,4-8,10	
Y	GB-A-2 136 175 (ATALLA) * the whole document *	2	
Y	---	4-8,10	
	EP-A-0 268 139 (IBM) * column 1, line 1 - column 3, line 1 * * column 6, line 7 - column 7, line 50 * * column 9, line 20 - line 29 * * column 19, line 9 - line 50 * * column 21, line 6 - line 18 * * claims 2,9 *		
A	---	3	
A	EP-A-0 561 685 (FUJITSU) * the whole document *	9	

TECHNICAL FIELDS SEARCHED (Int.Cl.6)

G06F

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	25 July 1995	Powell, D

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 715 243 A1

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
05.06.1996 Bulletin 1996/23

(51) Int Cl.[6]: **G06F 1/00**, G06F 17/60

(21) Application number: **95308414.2**

(22) Date of filing: **23.11.1995**

(84) Designated Contracting States:
DE FR GB

(30) Priority: **23.11.1994 US 344773**

(71) Applicant: **XEROX CORPORATION**
Rochester New York 14644 (US)

(72) Inventors:
- **Stefik, Mark J.**
  Woodside, California 94062 (US)
- **Pirolli, Peter L.T.**
  El Cerrrito, California 94530 (US)
- **Merkle, Ralph C.**
  Sunnyvale, California 94087 (US)

(74) Representative: **Goode, Ian Roy et al**
Rank Xerox Ltd
Patent Department
Parkway
Marlow Buckinghamshire SL7 1YL (GB)

(54) System for controlling the distribution and use of digital works having a fee reporting mechanism

(57) A fee accounting mechanism for reporting fees associated with the distribution and use of digital works. Usage rights and fees are attached to digital works. The usage rights define how the digital work may be used or further distributed. Usage fees are specified as part of a usage right. The digital works and their usage rights and fees are stored in repositories (201). The repositories control access to the digital works. Upon determination that the exercise of a usage right requires a fee, the repository generates a fee reporting transaction (302). Fee reporting is done to a credit server (301). The credit server collects the fee information and periodically transmits it to a billing clearinghouse (303).

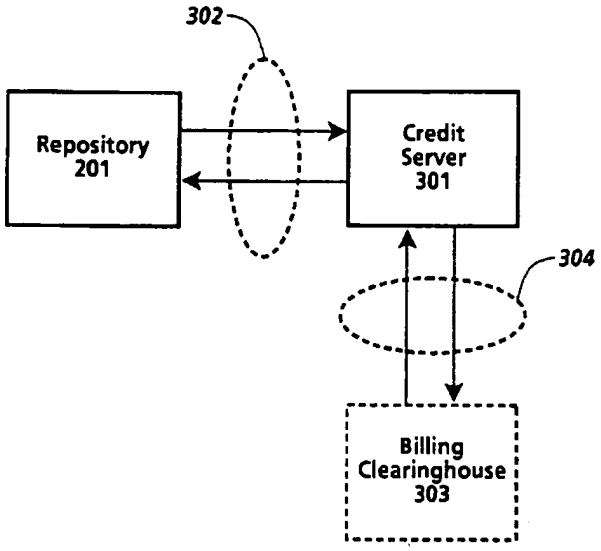


Fig. 3

EP 0 715 243 A1

Printed by Jouve, 75001 PARIS (FR)

Description

The present invention relates to the field of distribution and usage rights enforcement for digitally encoded works.

A fundamental issue facing the publishing and information industries as they consider electronic publishing is how to prevent the unauthorized and unaccounted distribution or usage of electronically published materials. Electronically published materials are typically distributed in a digital form and recreated on a computer based system having the capability to recreate the materials. Audio and video recordings, software, books and multimedia works are all being electronically published. Companies in these industries receive royalties for each accounted for delivery of the materials, e.g. the sale of an audio CD at a retail outlet. Any unaccounted distribution of a work results in an unpaid royalty (e.g. copying the audio recording CD to another digital medium.)

The ease in which electronically published works can be "perfectly" reproduced and distributed is a major concern. The transmission of digital works over networks is commonplace. One such widely used network is the Internet. The Internet is a widespread network facility by which computer users in many universities, corporations and government entities communicate and trade ideas and information. Computer bulletin boards found on the Internet and commercial networks such as CompuServ and Prodigy allow for the posting and retrieving of digital information. Information services such as Dialog and LEXIS/NEXIS provide databases of current information on a wide variety of topics. Another factor which will exacerbate the situation is the development and expansion of the National Information Infrastructure (the NII). It is anticipated that, as the NII grows, the transmission of digital works over networks will increase many times over. It would be desirable to utilize the NII for distribution of digital works without the fear of widespread unauthorized copying.

The most straightforward way to curb unaccounted distribution is to prevent unauthorized copying and transmission. For existing materials that are distributed in digital form, various safeguards are used. In the case of software, copy protection schemes which limit the number of copies that can be made or which corrupt the output when copying is detected have been employed. Another scheme causes software to become disabled after a predetermined period of time has lapsed. A technique used for workstation based software is to require that a special hardware device must be present on the workstation in order for the software to run, e.g., see US-A-4,932,054 entitled "Method and Apparatus for Protecting Computer Software Utilizing Coded Filter Network in Conjunction with an Active Coded Hardware Device." Such devices are provided with the software and are commonly referred to as dongles.

Yet another scheme is to distribute software, but which requires a "key" to enable its use. This is employed in distribution schemes where "demos" of the software are provided on a medium along with the entire product. The demos can be freely used, but in order to use the actual product, the key must be purchased. These schemes do not hinder copying of the software once the key is initially purchased.

It is an object of the present invention to provide an improved system and method for controlling the use and distribution of digital works.

The invention accordingly provides a system and method as claimed in the accompanying claims.

In a system for the control of distribution and use of digital works, a fee reporting mechanism for reporting fees associated with such distribution and use is disclosed. The system includes a means for attaching usage rights to a digital work. The usage rights define how the digital work may be used or further distributed by a possessor of the digital work. Usage fees are specified as part of a usage right. The ability to report usage fees may be a condition to the exercise of a usage right. Further, different fees may be assigned to different usage rights.

The present invention enables various usage fee scenarios to be used. Fees may be assessed on a per use basis, on a metered basis or based on a predetermined schedule. Fees may also be discounted on a predetermined schedule, or they can be marked-up a predetermined percentage (e.g. as a distributor fee). Fee reporting may also be deferred to a later time, to accommodate special deals, rebates or some other external information not yet available.

The present invention supports usage fees in an additive fashion. Usage fees may be reported for a composite digital work, i.e. a digital work comprised of a plurality of discrete digital works each having their own usage rights, and for distributors of digital works. Accordingly, fees to multiple revenue owners can be reported.

Usage fee reporting is done to a credit server. The credit server collects the fee information and periodically transmits it to a billing clearinghouse. Alternatively, the credit server may have a pre-allocated credit which is decremented as fees are incurred. In this alternative embodiment, the credit server would have to be periodically reallocated with credits to enable further use.

A system and method in accordance with the invention will now be described, by way of example, with reference to the accompanying drawings, in which:-

Figure 1 is a flowchart illustrating a simple instantiation of the operation of the currently preferred embodiment of the present invention.

Figure 2 is a block diagram illustrating the various repository types and the repository transaction flow between them in the currently preferred embodiment of the present invention.

Figure 3 is a block diagram of a repository coupled with a credit server in the currently preferred embodiment of

the present invention.

Figures 4a and 4b are examples of rendering systems as may be utilized in the currently preferred embodiment of the present invention.

Figure 5 illustrates a contents file layout for a digital work as may be utilized in the currently preferred embodiment of the present invention.

Figure 6 illustrates a contents file layout for an individual digital work of the digital work of Figure 5 as may be utilized in the currently preferred embodiment of the present invention.

Figure 7 illustrates the components of a description block of the currently preferred embodiment of the present invention.

Figure 8 illustrates a description tree for the contents file layout of the digital work illustrated in Figure 5.

Figure 9 illustrates a portion of a description tree corresponding to the individual digital work illustrated in Figure 6.

Figure 10 illustrates a layout for the rights portion of a description block as may be utilized in the currently preferred embodiment of the present invention.

Figure 11 is a description tree wherein certain d-blocks have PRINT usage rights and is used to illustrate "strict" and "lenient" rules for resolving usage rights conflicts.

Figure 12 is a block diagram of the hardware components of a repository as are utilized in the currently preferred embodiment of the present invention.

Figure 13 is a block diagram of the functional (logical) components of a repository as are utilized in the currently preferred embodiment of the present invention.

Figure 14 is diagram illustrating the basic components of a usage right in the currently preferred embodiment of the present invention.

Figure 15 lists the usage rights grammar of the currently preferred embodiment of the present invention.

Figure 16 is a flowchart illustrating the steps of certificate delivery, hotlist checking and performance testing as performed in a registration transaction as may be performed in the currently preferred embodiment of the present invention.

Figure 17 is a flowchart illustrating the steps of session information exchange and clock synchronization as may be performed in the currently preferred embodiment of the present invention, after each repository in the registration transaction has successfully completed the steps described in Figure 16.

Figure 18 is a flowchart illustrating the basic flow for a usage transaction, including the common opening and closing step, as may be performed in the currently preferred embodiment of the present invention.

Figure 19 is a state diagram of server and client repositories in accordance with a transport protocol followed when moving a digital work from the server to the client repositories, as may be performed in the currently preferred embodiment of the present invention.

OVERVIEW

A system for controlling use and distribution of digital works is disclosed. The present invention is directed to supporting commercial transactions involving digital works.

Herein the terms "digital work", "work" and "content" refer to any work that has been reduced to a digital representation. This would include any audio, video, text, or multimedia work and any accompanying interpreter (e.g. software) that may be required for recreating the work. The term composite work refers to a digital work comprised of a collection of other digital works. The term "usage rights" or "rights" is a term which refers to rights granted to a recipient of a digital work. Generally, these rights define how a digital work can be used and if it can be further distributed. Each usage right may have one or more specified conditions which must be satisfied before the right may be exercised.

Figure 1 is a high level flowchart omitting various details but which demonstrates the basic operation of the present invention. Referring to Figure 1, a creator creates a digital work, step 101. The creator will then determine appropriate usage rights and fees, attach them to the digital work, and store them in Repository 1, step 102. The determination of appropriate usage rights and fees will depend on various economic factors. The digital work remains securely in Repository 1 until a request for access is received. The request for access begins with a session initiation by another repository. Here a Repository 2 initiates a session with Repository 1, step 103. As will be described in greater detail below, this session initiation includes steps which helps to insure that the respective repositories are trustworthy. Assuming that a session can be established, Repository 2 may then request access to the Digital Work for a stated purpose, step 104. The purpose may be, for example, to print the digital work or to obtain a copy of the digital work. The purpose will correspond to a specific usage right. In any event, Repository 1 checks the usage rights associated with the digital work to determine if the access to the digital work may be granted, step 105. The check of the usage rights essentially involves a determination of whether a right associated with the access request has been attached to the digital work and if all conditions associated with the right are satisfied. If the access is denied, repository 1 terminates the session with an error message, step 106. If access is granted, repository 1 transmits the digital work to repository

2, step 107. Once the digital work has been transmitted to repository 2, repository 1 and 2 each generate billing information for the access which is transmitted to a credit server, step 108. Such double billing reporting is done to insure against attempts to circumvent the billing process.

Figure 2 illustrates the basic interactions between repository types in the present invention. As will become apparent from Figure 2, the various repository types will serve different functions. It is fundamental that repositories will share a core set of functionality which will enable secure and trusted communications. Referring to Figure 2, a repository 201 represents the general instance of a repository. The repository 201 has two modes of operation; a server mode and a requester mode. When in the server mode, the repository will be receiving and processing access requests to digital works. When in the requester mode, the repository will be initiating requests to access digital works. Repository 201 is general in the sense that its primary purpose is as an exchange medium for digital works. During the course of operation, the repository 201 may communicate with a plurality of other repositories, namely authorization repository 202, rendering repository 203 and master repository 204. Communication between repositories occurs utilizing a repository transaction protocol 205.

Communication with an authorization repository 202 may occur when a digital work being accessed has a condition requiring an authorization. Conceptually, an authorization is a digital certificate such that possession of the certificate is required to gain access to the digital work. An authorization is itself a digital work that can be moved between repositories and subjected to fees and usage rights conditions. An authorization may be required by both repositories involved in an access to a digital work.

Communication with a rendering repository 203 occurs in connection with the rendering of a digital work. As will be described in greater detail below, a rendering repository is coupled with a rendering device (e.g. a printer device) to comprise a rendering system.

Communication with a master repository 205 occurs in connection with obtaining an identification certificate. Identification certificates are the means by which a repository is identified as "trustworthy". The use of identification certificates is described below with respect to the registration transaction.

Figure 3 illustrates the repository 201 coupled to a credit server 301. The credit server 301 is a device which accumulates billing information for the repository 201. The credit server 301 communicates with repository 201 via billing transactions 302 to record billing transactions. Billing transactions are reported to a billing clearinghouse 303 by the credit server 301 on a periodic basis. The credit server 301 communicates to the billing clearinghouse 303 via clearinghouse transactions 304. The clearinghouse transactions 304 enable a secure and encrypted transmission of information to the billing clearinghouse 303.

RENDERING SYSTEMS

A rendering system is generally defined as a system comprising a repository and a rendering device which can render a digital work into its desired form. Examples of a rendering system may be a computer system, a digital audio system, or a printer. A rendering system has the same security features as a repository. The coupling of a rendering repository with the rendering device may occur in a manner suitable for the type of rendering device.

Figure 4a illustrates a printer as an example of a rendering system. Referring to Figure 4, printer system 401 has contained therein a printer repository 402 and a print device 403. It should be noted that the the dashed line defining printer system 401 defines a secure system boundary. Communications within the boundary are assumed to be secure. Depending on the security level, the boundary also represents a barrier intended to provide physical integrity. The printer repository 402 is an instantiation of the rendering repository 205 of Figure 2. The printer repository 402 will in some instances contain an ephemeral copy of a digital work which remains until it is printed out by the print engine 403. In other instances, the printer repository 402 may contain digital works such as fonts, which will remain and can be billed based on use. This design assures that all communication lines between printers and printing devices are encrypted, unless they are within a physically secure boundary. This design feature eliminates a potential "fault" point through which the digital work could be improperly obtained. The printer device 403 represents the printer components used to create the printed output.

Also illustrated in Figure 4a is the repository 404. The repository 404 is coupled to the printer repository 402. The repository 404 represents an external repository which contains digital works.

Figure 4b is an example of a computer system as a rendering system. A computer system may constitute a "multifunction" device since it may execute digital works (e.g. software programs) and display digital works (e.g. a digitized photograph). Logically, each rendering device can be viewed as having its own repository, although only one physical repository is needed. Referring to Figure 4b, a computer system 410 has contained therein a display/execution repository 411. The display/execution repository 411 is coupled to display device, 412 and execution device 413. The dashed box surrounding the computer system 410 represents a security boundary within which communications are assumed to be secure. The display/execution repository 411 is further coupled to a credit server 414 to report any fees to be billed for access to a digital work and a repository 415 for accessing digital works stored therein.

STRUCTURE OF DIGITAL WORKS

Usage rights are attached directly to digital works. Thus, it is important to understand the structure of a digital work. The structure of a digital work, in particular composite digital works, may be naturally organized into an acyclic structure such as a hierarchy. For example, a magazine has various articles and photographs which may have been created and are owned by different persons. Each of the articles and photographs may represent a node in a hierarchical structure. Consequently, controls, i.e. usage rights, may be placed on each node by the creator. By enabling control and fee billing to be associated with each node, a creator of a work can be assured that the rights and fees are not circumvented.

In the currently preferred embodiment, the file information for a digital work is divided into two files: a "contents" file and a "description tree" file. From the perspective of a repository, the "contents" file is a stream of addressable bytes whose format depends completely on the interpreter used to play, display or print the digital work. The description tree file makes it possible to examine the rights and fees for a work without reference to the content of the digital work. It should be noted that the term description tree as used herein refers to any type of acyclic structure used to represent the relationship between the various components of a digital work.

Figure 5 illustrates the layout of a contents file. Referring to Figure 5, a digital work is comprised of story A 510, advertisement 511, story B 512 and story C 513. It is assumed that the digital work is stored starting at a relative address of 0. Each of the parts of the digital work are stored linearly so that story A 510 is stored at approximately addresses 0-30,000, advertisement 511 at addresses 30,001-40,000, story B 512 at addresses 40,001-60,000 and story C 513 at addresses 60,001-85K. The detail of story A 510 is illustrated in Figure 6. Referring to Figure 6, the story A 510 is further broken down to show text 614 stored at address 0-1500, soldier photo 615 at addresses 1501-10,000, graphics 616 stored at addresses 10,001-25,000 and sidebar 617 stored address 25,001-30,000. Note that the data in the contents file may be compressed (for saving storage) or encrypted (for security).

From Figures 5 and 6 it is readily observed that a digital work can be represented by its component parts as a hierarchy. The description tree for a digital work is comprised of a set of related descriptor blocks (d-blocks). The contents of each d-block is described with respect to Figure 7. Referring to Figure 7, a d-block 700 includes an identifier 701 which is a unique identifier for the work in the repository, a starting address 702 providing the start address of the first byte of the work, a length 703 giving the number of bytes in the work, a rights portion 704 wherein the granted usage rights and their status data are maintained, a parent pointer 705 for pointing to a parent d-block and child pointers 706 for pointing to the child d-blocks. In the currently preferred embodiment, the identifier 701 has two parts. The first part is a unique number assigned to the repository upon manufacture. The second part is a unique number assigned to the work upon creation. The rights portion 704 will contain a data structure, such as a look-up table, wherein the various information associated with a right is maintained. The information required by the respective usage rights is described in more detail below. D-blocks form a strict hierarchy. The top d-block of a work has no parent; all other d-blocks have one parent. The relationship of usage rights between parent and child d-blocks and how conflicts are resolved is described below.

A special type of d-block is a "shell" d-block. A shell d-block adds no new content beyond the content of its parts. A shell d-block is used to add rights and fee information, typically by distributors of digital works.

Figure 8 illustrates a description tree for the digital work of Figure 5. Referring to Figure 8, a top d-block 820 for the digital work points to the various stories and advertisements contained therein. Here, the top d-block 820 points to d-block 821 (representing story A 510), d-block 822 (representing the advertisement 511), d-block 823 (representing story B 512) and and d-block 824 (representing story C 513).

The portion of the description tree for Story A 510 is illustrated in Figure 9. D-block 925 represents text 614, d-block 926 represents photo 615, d-block 927 represents graphics 616 by and d-block 928 represents sidebar 617.

The rights portion 704 of a descriptor block is further illustrated in Figure 10. Figure 10 illustrates a structure which is repeated in the rights portion 704 for each right. Referring to Figure 10, each right will have a right code field 1050 and status information field 1052. The right code field 1050 will contain a unique code assigned to a right. The status information field 1052 will contain information relating to the state of a right and the digital work. Such information is indicated below in Table 1. The rights as stored in the rights portion 704 may typically be in numerical order based on the right code.

TABLE 1

DIGITAL WORK STATE INFORMATION		
Property	**Value**	**Use**
Copies-in-Use	Number	A counter of the number of copies of a work that are in use. Incremented when another copy is used; decremented when use is completed.
Loan-Period	Time-Units	Indicator of the maximum number of time-units that a document can be loaned out
Loaner-Copy	Boolean	Indicator that the current work is a loaned out copy of an authorized digital work.
Remaining-Time	Time-Units	Indicator of the remaining time of use on a metered document right.
Document-Descr	String	A string containing various identifying information about a document. The exact format of this is not specified, but it can include information such as a publisher name, author name, ISBN number, and so on.
Revenue-Owner	RO-Descr	A handle identifying a revenue owner for a digital work. This is used reporting usage fees.
Publication-Date	Date-Descr	The date that the digital work was published.
History-list	History-Rec	A list of events recording the repostories and dates for operations that copy, transfer, backup, or restore a digital work.

The approach for representing digtal works by separating description data from content assumes that parts of a file are contiguous but takes no position on the actual representation of content. In particular, it is neutral to the question of whether content representation may take an object oriented approach. It would be natural to represent content as objects. In principle, it may be convenient to have content objects that include the billing structure and rights information that is represented in the d-blocks. Such variations in the design of the representation are possible and are viable alternatives but may introduce processing overhead, e.g. the interpretation of the objects.

Digital works are stored in a repository as part of a hierarchical file system. Folders (also termed directories and sub-directories) contain the digital works as well as other folders. Digital works and folders in a folder are ordered in alphabetical order. The digital works are typed to reflect how the files are used. Usage rights can be attached to folders so that the folder itself is treated as a digital work. Access to the folder would then be handled in the same fashion as any other digital work As will be described in more detail below, the contents of the folder are subject to their own rights. Moreover, file management rights may be attached to the folder which define how folder contents can be managed.

ATTACHING USAGE RIGHTS TO A DIGITAL WORK

It is fundamental to the present invention that the usage rights are treated as part of the digital work. As the digital work is distributed, the scope of the granted usage rights will remain the same or may be narrowed. For example, when a digital work is transferred from a document server to a repository, the usage rights may include the right to loan a copy for a predetermined period of time (called the original rights). When the repository loans out a copy of the digital work, the usage rights in the loaner copy (called the next set of rights) could be set to prohibit any further rights to loan out the copy. The basic idea is that one cannot grant more rights than they have.

The attachment of usage rights into a digital work may occur in a variety of ways. If the usage rights will be the same for an entire digital work, they could be attached when the digital work is processed for deposit in the digital work server. In the case of a digital work having different usage rights for the various components, this can be done as the digital work is being created. An authoring tool or digital work assembling tool could be utilized which provides for an automated process of attaching the usage rights.

As will be described below, when a digital work is copied, transferred or loaned, a "next set of rights" can be specified. The"next set of rights" will be attached to the digital work as it is transported.

Resolving Conflicting Rights

Because each part of a digital work may have its own usage rights, there will be instances where the rights of a "contained part" are different from its parent or container part. As a result, conflict rules must be established to dictate when and how a right may be exercised. The hierarchical structure of a digital work facilitates the enforcement of such rules. A "strict" rule would be as follows: a right for a part in a digital work is sanctioned if and only if it is sanctioned

for the part, for ancestor d-blocks containing the part and for all descendent d-blocks. By sanctioned, it is meant that (1) each of the respective parts must have the right, and (2) any conditions for exercising the right are satisfied.

It also possible to implement the present invention using a more lenient rule. In the more lenient rule, access to the part may be enabled to the descendent parts which have the right, but access is denied to the descendents which do not.

An example of applying both the strict rule and lenient is illustrated with reference to Figure 11. Referring to Figure 11, a root d-block 1101 has child d-blocks 1102-1105. In this case, root d-block represents a magazine, and each of the child d-blocks 1102-1105 represent articles in the magazine. Suppose that a request is made to PRINT the digital work represented by root d-block 1101 wherein the strict rule is followed. The rights for the root d-block 1101 and child d-blocks 1102-1105 are then examined. Root d-block 1101 and child d-blocks 1102 and 1105 have been granted PRINT rights. Child d-block 1103 has not been granted PRINT rights and child d-block 1104 has PRINT rights conditioned on payment of a usage fee.

Under the strict rule the PRINT right cannot be exercised because the child d-block does not have the PRINT right. Under the lenient rule, the result would be different. The digital works represented by child d-blocks 1102 and 1105 could be printed and the digital work represented by d-block 1104 could be printed so long as the usage fee is paid. Only the digital work represented by d-block 1103 could not be printed. This same result would be accomplished under the strict rule if the requests were directed to each of the individual digital works.

The present invention supports various combinations of allowing and disallowing access. Moreover, as will be described below, the usage rights grammar permits the owner of a digital work to specify if constraints may be imposed on the work by a container part. The manner in which digital works may be sanctioned because of usage rights conflicts would be implementation specific and would depend on the nature of the digital works.

REPOSITORIES

In the description of Figure 2, it was indicated that repositories come in various forms. All repositories provide a core set of services for the transmission of digital works. The manner in which digital works are exchanged is the basis for all transaction between repositories. The various repository types differ in the ultimate functions that they perform. Repositories may be devices themselves, or they may be incorporated into other systems. An example is the rendering repository 203 of Figure 2.

A repository will have associated with it a repository identifier. Typically, the repository identifier would be a unique number assigned to the repository at the time of manufacture. Each repository will also be classified as being in a particular security class. Certain communications and transactions may be conditioned on a repository being in a particular security class. The various security classes are described in greater detail below.

As a prerequisite to operation, a repository will require possession of an identification certificate. Identification certificates are encrypted to prevent forgery and are issued by a Master repository. A master repository plays the role of an authorization agent to enable repositories to receive digital works. Identification certificates must be updated on a periodic basis. Identification certificates are described in greater detail below with respect to the registration transaction.

A repository has both a hardware and functional embodiment. The functional embodiment is typically software executing on the hardware embodiment. Alternatively, the functional embodiment may be embedded in the hardware embodiment such as an Application Specific Integrated Circuit (ASIC) chip.

The hardware embodiment of a repository will be enclosed in a secure housing which if compromised, may cause the repository to be disabled. The basic components of the hardware embodiment of a repository are described with reference to Figure 12. Referring to Figure 12, a repository is comprised of a processing means 1200, storage system 1207, clock 1205 and external interface 1206. The processing means 1200 is comprised of a processor element 1201 and processor memory 1202. The processing means 1201 provides controller, repository transaction and usage rights transaction functions for the repository. Various functions in the operation of the repository such as decryption and/or decompression of digital works and transaction messages are also performed by the processing means 1200. The processor element 1201 may be a microprocessor or other suitable computing component. The processor memory 1202 would typically be further comprised of Read Only Memories (ROM) and Random Access Memories (RAM). Such memories would contain the software instructions utilized by the processor element 1201 in performing the functions of the repository.

The storage system 1207 is further comprised of descriptor storage 1203 and content storage 1204. The description tree storage 1203 will store the description tree for the digital work and the content storage will store the associated content. The description tree storage 1203 and content storage 1204 need not be of the same type of storage medium, nor are they necessarily on the same physical device. So for example, the descriptor storage 1203 may be stored on a solid state storage (for rapid retrieval of the description tree information), while the content storage 1204 may be on a high capacity storage such as an optical disk.

The clock 1205 is used to time-stamp various time based conditions for usage rights or for metering usage fees which may be associated with the digital works. The clock 1205 will have an uninterruptable power supply, e.g. a battery, in order to maintain the integrity of the time-stamps. The external interface means 1206 provides for the signal connection to other repositories and to a credit server. The external interface means 1206 provides for the exchange of signals via such standard interfaces such as RS-232 or Personal Computer Manufacturers Card Industry Association (PCMCIA) standards, or FDDI. The external interface means 1206 may also provide network connectivity.

The functional embodiment of a repository is described with reference to Figure 13. Referring to Figure 13, the functional embodiment is comprised of an operating system 1301, core repository services 1302, usage transaction handlers 1303, repository specific functions, 1304 and a user interface 1305. The operating system 1301 is specific to the repository and would typically depend on the type of processor being used. The operating system 1301 would also provide the basic services for controlling and interfacing between the basic components of the repository.

The core repository services 1302 comprise a set of functions required by each and every repository. The core repository services 1302 include the session initiation transactions which are defined in greater detail below. This set of services also includes a generic ticket agent which is used to "punch" a digital ticket and a generic authorization server for processing authorization specifications. Digital tickets and authorizations are specific mechanisms for controlling the distribution and use of digital works and are described in more detail below. Note that coupled to the core repository services are a plurality of identification certificates 1306. The identification certificates 1306 are required to enable the use of the repository.

The usage transactions handlers 1303 comprise functionality for processing access requests to digital works and for billing fees based on access. The usage transactions supported will be different for each repository type. For example, it may not be necessary for some repositories to handle access requests for digital works.

The repository specific functionality 1304 comprises functionality that is unique to a repository. For example, the master repository has special functionality for issuing digital certificates and maintaining encryption keys. The repository specific functionality 1304 would include the user interface implementation for the repository.

Repository Security Classes

For some digital works the losses caused by any individual instance of unauthorized copying is insignificant and the chief economic concern lies in assuring the convenience of access and low-overhead billing. In such cases, simple and inexpensive handheld repositories and network-based workstations may be suitable repositories, even though the measures and guarantees of security are modest.

At the other extreme, some digital works such as a digital copy of a first run movie or a bearer bond or stock certificate would be of very high value so that it is prudent to employ caution and fairly elaborate security measures to ensure that they are not copied or forged. A repository suitable for holding such a digital work could have elaborate measures for ensuring physical integrity and for verifying authorization before use.

By arranging a universal protocol, all kinds of repositories can communicate with each other in principle. However, creators of some works will want to specify that their works will only be transferred to repositories whose level of security is high enough. For this reason, document repositories have a ranking system for classes and levels of security. The security classes in the currently preferred embodiment are described in Table 2.

TABLE 2

REPOSITORY SECURITY LEVELS	
Level	Description of Security
0	Open system.Document transmission is unencrypted. No digital certificate is required for identification. The security of the system depends mostly on user honesty, since only modest knowledge may be needed to circumvent the security measures. The repositery has no provisions for preventing unauthorized programs from running and accessing or copying files. The system does not prevent the use of removable storage and does not encrypt stored files.
1	Minimal security. Like the previous class except that stored files are minimally encrypted, including ones on removable storage.
2	Basic security. Like the previous class except that special tools and knowledge are required to compromise the programming, the contents of the repository, or the state of the clock. All digital communications are encrypted. A digital certificate is provided as identification. Medium level encryption is used. Repository identification number is unforgeable.

Continuation of the Table on the next page

TABLE 2 (continued)

REPOSITORY SECURITY LEVELS	
Level	Description of Security
3	General security. Like the previous class plus the requirement of special tools are needed to compromise the physical integrity of the repository and that modest encryption is used on all transmissions. Password protection is required to use the local user interface. The digital clock system cannot be reset without authorization. No works would be stored on removable storage. When executing works as programs, it runs them in their own address space and does not give them direct access to any file storage or other memory containing system code or works. They can access works only through the transmission transaction protocol.
4	Like the previous class except that high level encryption is used on all communications. Sensors are used to record attempts at physical and electronic tampering. After such tampering, the repository will not perform other transactions until it has reported such tampering to a designated server.
5	Like the previous class except that if the physical or digital attempts at tampering exceed some preset thresholds that threaten the physical integrity of the repository or the integrity of digital and cryptographic barriers, then the repository will save only document description records of history but will erase or destroy any digital identifiers that could be misused if released to an unscrupulous. It also modifies any certificates of authenticity to indicate that the physical system has been compromised. It also erases the contents of designated documents.
6	Like the previous class except that the repository will attempt wireless communication to report tampering and will employ noisy alarms.
10	This would correspond to a very high level of security. This server would maintain constant communications to remote security systems reporting transactions, sensor readings, and attempts to circumvent security.

The characterization of security levels described in Table 2 is not intended to be fixed. More important is the idea of having different security levels for different repositories. It is anticipated that new security classes and requirements will evolve according to social situations and changes in technology.

Repository User Interface

A user interface is broadly defined as the mechanism by which a user interacts with a repository in order to invoke transactions to gain access to a digital work, or exercise usage rights. As described above, a repository may be embodied in various forms. The user interface for a repository will differ depending on the particular embodiment. The user interface may be a graphical user interface having icons representing the digital works and the various transactions that may be performed. The user interface may be a generated dialog in which a user is prompted for information.

The user interface itself need not be part of the repository. As a repository may be embedded in some other device, the user interface may merely be a part of the device in which the repository is embedded. For example, the repository could be embedded in a "card" that is inserted into an available slot in a computer system. The user interface may be a combination of a display, keyboard, cursor control device and software executing on the computer system.

At a minimum, the user interface must permit a user to input information such as access requests and alpha numeric data and provide feedback as to transaction status. The user interface will then cause the repository to initiate the suitable transactions to service the request. Other facets of a particular user interface will depend on the functionality that a repository will provide.

CREDIT SERVERS

In the present invention, fees may be associated with the exercise of a right. The requirement for payment of fees is described with each version of a usage right in the usage rights language. The recording and reporting of such fees is performed by the credit server. One of the capabilities enabled by associating fees with rights is the possibility of supporting a wide range of charging models. The simplest model, used by conventional software, is that there is a single fee at the time of purchase, after which the purchaser obtains unlimited rights to use the work as often and for as long as he or she wants. Alternative models, include metered use and variable fees. A single work can have different fees for different uses. For example, viewing a photograph on a display could have different fees than making a hardcopy

or including it in a newly created work. A key to these alternative charging models is to have a low overhead means of establishing fees and accounting for credit on these transactions.

A credit server is a computational system that reliably authorizes and records these transactions so that fees are billed and paid. The credit server reports fees to a billing clearinghouse. The billing clearinghouse manages the financial transactions as they occur. As a result, bills may be generated and accounts reconciled. Preferably, the credit server would store the fee transactions and periodically communicate via a network with the billing clearinghouse for reconciliation. In such an embodiment, communications with the billing clearinghouse would be encrypted for integrity and security reasons. In another embodiment, the credit server acts as a "debit card" where transactions occur in "real-time" against a user account.

A credit server is comprised of memory, a processing means, a clock, and interface means for coupling to a repository and a financial institution (e.g. a modem). The credit server will also need to have security and authentication functionality. These elements are essentially the same elements as those of a repository. Thus, a single device can be both a repository and a credit server, provided that it has the appropriate processing elements for carrying out the corresponding functions and protocols. Typically, however, a credit server would be a cardsized system in the possession of the owner of the credit. The credit server is coupled to a repository and would interact via financial transactions as described below. Interactions with a financial institution may occur via protocols established by the financial institutions themselves.

In the currently preferred embodiment credit servers associated with both the server and the repository report the financial transaction to the billing clearinghouse. For example, when a digital work is copied by one repository to another for a fee, credit servers coupled to each of the repositories will report the transaction to the billing clearinghouse. This is desirable in that it insures that a transaction will be accounted for in the event of some break in the communication between a credit server and the billing clearinghouse. However, some implementations may embody only a single credit server reporting the transaction to minimize transaction processing at the risk of losing some transactions.

USAGE RIGHTS LANGUAGE

The present invention uses statements in a high level "usage rights language" to define rights associated with digital works and their parts. Usage rights statements are interpreted by repositories and are used to determine what transactions can be successfully carried out for a digital work and also to determine parameters for those transactions. For example, sentences in the language determine whether a given digital work can be copied, when and how it can be used, and what fees (if any) are to be charged for that use. Once the usage rights statements are generated, they are encoded in a suitable form for accessing during the processing of transactions.

Defining usage rights in terms of a language in combination with the hierarchical representation of a digital work enables the support of a wide variety of distribution and fee schemes. An example is the ability to attach multiple versions of a right to a work. So a creator may attach a PRINT right to make 5 copies for $10.00 and a PRINT right to make unlimited copies for $100.00. A purchaser may then choose which option best fits his needs. Another example is that rights and fees are additive. So in the case of a composite work, the rights and fees of each of the components works is used in determining the rights and fees for the work as a whole.

The basic contents of a right are illustrated in Figure 14. Referring to Figure 14, a right 1450 has a transactional component 1451 and a specifications component 1452. A right 1450 has a label (e.g. COPY or PRINT) which indicates the use or distribution privileges that are embodied by the right. The transactional component 1451 corresponds to a particular way in which a digital work may be used or distributed. The transactional component 1451 is typically embodied in software instructions in a repository which implement the use or distribution privileges for the right. The specifications components 1452 are used to specify conditions which must be satisfied prior to the right being exercised or to designate various transaction related parameters. In the currently preferred embodiment, these specifications include copy count 1453, Fees and Incentives 1454, Time 1455, Access and Security 1456 and Control 1457. Each of these specifications will be described in greater detail below with respect to the language grammar elements.

The usage rights language is based on the grammar described below. A grammar is a convenient means for defining valid sequence of symbols for a language. In describing the grammar the notation "[a|b|c]" is used to indicate distinct choices among alternatives. In this example, a sentence can have either an "a", "b" or "c". It must include exactly one of them. The braces {} are used to indicate optional items. Note that brackets, bars and braces are used to describe the language of usage rights sentences but do not appear in actual sentences in the language.

In contrast, parentheses are part of the usage rights language. Parentheses are used to group items together in lists. The notation (x^+) is used to indicate a variable length list, that is, a list containing one or more items of type x. The notation $(x)^*$ is used to indicate a variable number of lists containing x.

Keywords in the grammar are words followed by colons. Keywords are a common and very special case in the language. They are often used to indicate a single value, typically an identifier. In many cases, the keyword and the parameter are entirely optional. When a keyword is given, it often takes a single identifier as its value. In some cases,

the keyword takes a list of identifiers.

In the usage rights language, time is specified in an hours:minutes:seconds (or hh:mm:ss) representation. Time zone indicators, e.g. PDT for Pacific Daylight Time, may also be specified. Dates are represented as year/ month/day (or YYYY/MMM/DD). Note that these time and date representations may specify moments in time or units of time Money units are specified in terms of dollars.

Finally, in the usage rights language, various "things" will need to interact with each other. For example, an instance of a usage right may specify a bank account, a digital ticket, etc.. Such things need to be identified and are specified herein using the suffix "-ID."

The Usage Rights Grammar is listed in its entirety in Figure 15 and is described below.

Grammar element 1501 **"Digital Work Rights: = (Rights")"** define the digital work rights as a set of rights. The set of rights attached to a digital work define how that digital work may be transferred, used, performed or played. A set of rights will attach to the entire digital work and in the case of compound digital works, each of the components of the digital work. The usage rights of components of a digital may be different.

Grammar element 1502 **"Right : = (Right-Code {Copy-Count} {Control-Spec} {Time-Spec} {Access-Spec} {Fee-Spec})"** enumerates the content of a right. Each usage right must specify a right code. Each right may also optionally specify conditions which must be satisfied before the right can be exercised. These conditions are copy count, control, time, access and fee conditions. In the currently preferred embodiment, for the optional elements, the following defaults apply: copy count equals 1, no time limit on the use of the right, no access tests or a security level required to use the right and no fee is required. These conditions will each be described in greater detail below.

It is important to note that a digital work may have multiple versions of a right, each having the same right code. The multiple version would provide alternative conditions and fees for accessing the digital work.

Grammar element 1503 **"Right-Code : = Render-Code I Transport-Code I File-Management-CodeI Derivative-Works- Code Configuration-Code"** distinguishes each of the specific rights into a particular right type (although each right is identified by distinct right codes). In this way, the grammar provides a catalog of possible rights that can be associated with parts of digital works. In the following, rights are divided into categories for convenience in describing them.

Grammar element 1504 **"Render-Code : = [Play : {Player: Player-ID} I Print: {Printer: Printer-ID}]"** lists a category of rights all involving the making of ephemeral, transitory, or non-digital copies of the digital work. After use the copies are erased.

- Play — A process of rendering or performing a digital work on some processor. This includes such things as playing digital movies, playing digital music, playing a video game, running a computer program, or displaying a document on a display.
- Print — To render the work in a medium that is not further protected by usage rights, such as printing on paper.

Grammar element 1505 **"Transport-Code : = [Copy I Transfer I Loan (Remaining-Rights: Next-Set-of-Rights)] {(Next-Copy-Rights: Next-Set of Rights)}"** lists a category of rights involving the making of persistent, usable copies of the digital work on other repositories. The optional Next-Copy-Rights determine the rights on the work after it is transported. If this is not specified, then the rights on the transported copy are the same as on the original. The optional Remaining-Rights specify the rights that remain with a digital work when it is loaned out. If this is not specified, then the default is that no rights can be exercised when it is loaned out.

- Copy — Make a new copy of a work
- Transfer — Moving a work from one repository to another.
- Loan — Temporarily loaning a copy to another repository for a specified period of time.

Grammar element 1506 **"File-Management-Code : = Backup {Back-Up-Copy-Rights: Next-Set -of Rights}I Restore I Delete I Folder I Directory {Name:Hide-Local I Hide - Remote}{Parts:Hide-Local I Hide-Remote}"** lists a category of rights involving operations for file management, such as the making of backup copies to protect the copy owner against catastrophic equipment failure.

Many software licenses and also copyright law give a copy owner the right to make backup copies to protect against catastrophic failure of equipment. However, the making of uncontrolled backup copies is inherently at odds with the ability to control usage, since an uncontrolled backup copy can be kept and then restored even after the authorized copy was sold.

The File management rights enable the making and restoring of backup copies in a way that respects usage rights, honoring the requirements of both the copy owner and the rights grantor and revenue owner. Backup copies of work descriptions (including usage rights and fee data) can be sent under appropriate protocol and usage rights control to other document repositories of sufficiently high security. Further rights permit organization of digital works into folders

which themselves are treated as digital works and whose contents may be "hidden" from a party seeking to determine the contents of a repository.

- Backup To make a backup copy of a digital work as protection against media failure.
- Restore To restore a backup copy of a digital work.
- Delete To delete or erase a copy of a digital work.
- Folder To create and name folders, and to move files and folders between folders.
- Directory To hide a folder or its contents.

Grammar element 1507 **"Derivative-Works-Code : [Extract | Embed | Edit {Process: Process-ID}] {Next-Copy-Rights : Next-Set-of Rights}"** lists a category of rights involving the use of a digital work to create new works.

- Extract To remove a portion of a work, for the purposes of creating a new work.
- Embed To include a work in an existing work.
- Edit To alter a digital work by copying, selecting and modifying portions of an existing digital work.

Grammar element 1508 **"Configuration-Code: = Install | Uninstall"** lists a category of rights for installing and uninstalling software on a repository (typically a rendering repository.) This would typically occur for the installation of a new type of player within the rendering repository.

- Install: To install new software on a repository.
- Uninstall: To remove existing software from a repository.

Grammar element 1509 **"Next-Set-of-Rights : = {(Add: Set-Of-Rights)} {(Delete: Set-Of-Rights)} {(Replace: Set-Of-Rights)} {(Keep: Set-Of-Rights)}"** defines how rights are carried forward for a copy of a digital work. If the Next-Copy-Rights is not specified, the rights for the next copy are the same as those of the current copy. Otherwise, the set of rights for the next copy can be specified. Versions of rights after Add: are added to the current set of rights. Rights after Delete: are deleted from the current set of rights. If only right codes are listed after Delete:, then all versions of rights with those codes are deleted. Versions of rights after Replace: subsume all versions of rights of the same type in the current set of rights.

If Remaining-Rights is not specified, then there are no rights for the original after all Loan copies are loaned out. If Remaining-Rights is specified, then the Keep: token can be used to simplify the expression of what rights to keep behind. A list of right codes following keep means that all of the versions of those listed rights are kept in the remaining copy. This specification can be overridden by subsequent Delete: or Replace: specifications.

Copy Count Specification

For various transactions, it may be desirable to provide some limit as to the number of "copies" of the work which may be exercised simultaneously for the right. For example, it may be desirable to limit the number of copies of a digital work that may be loaned out at a time or viewed at a time.

Grammar element 1510 **"Copy-Count : = (Copies: positive-integer | 0 | unlimited)"** provides a condition which defines the number of "copies" of a work subject to the right . A copy count can be 0, a fixed number, or unlimited. The copy-count is associated with each right, as opposed to there being just a single copy-count for the digital work. The Copy-Count for a right is decremented each time that a right is exercised. When the Copy-Count equals zero, the right can no longer be exercised. If the Copy-Count is not specified, the default is one.

Control Specification

Rights and fees depend in general on rights granted by the creator as well as further restrictions imposed by later distributors. Control specifications deal with interactions between the creators and their distributors governing the imposition of further restrictions and fees. For example, a distributor of a digital work may not want an end consumer of a digital work to add fees or otherwise profit by commercially exploiting the purchased digital work.

Grammar element 1511 **"Control-Spec : = (Control: {Restrictable | Unrestrictable} {Unchargeable | Chargeable})"** provides a condition to specify the effect of usage rights and fees of parents on the exercise of the right. A digital work is restrictable if higher level d-blocks can impose further restrictions (time specifications and access specifications) on the right. It is unrestrictable if no further restrictions can be imposed. The default setting is restrictable. A right is unchargeable if no more fees can be imposed on the use of the right. It is chargeable if more fees can be imposed. The default is chargeable.

Time Specification

It is often desirable to assign a start date or specify some duration as to when a right may be exercised. Grammar element 1512 **"Time-Spec : = ({Fixed-Interval | Sliding-Interval | Meter-Time} Until: Expiration-Date)"** provides for specification of time conditions on the exercise of a right. Rights may be granted for a specified time. Different kinds of time specifications are appropriate for different kinds of rights. Some rights may be exercised during a fixed and predetermined duration. Some rights may be exercised for an interval that starts the first time that the right is invoked by some transaction. Some rights may be exercised or are charged according to some kind of metered time, which may be split into separate intervals. For example, a right to view a picture for an hour might be split into six ten minute viewings or four fifteen minute viewings or twenty three minute viewings.

The terms "time" and "date" are used synonymously to refer to a moment in time. There are several kinds of time specifications. Each specification represents some limitation on the times over which the usage right applies. The Expiration-Date specifies the moment at which the usage right ends. For example, if the Expiration-Date is "Jan 1, 1995," then the right ends at the first moment of 1995. If the Expiration-Date is specified as "forever", then the rights are interpreted as continuing without end. If only an expiration date is given, then the right can be exercised as often as desired until the expiration date.

Grammar element 1513 **"Fixed-Interval : = From: Start-Time"** is used to define a predetermined interval that runs from the start time to the expiration date.

Grammar element 1514 **"Sliding-Interval : = Interval: Use-Duration"** is used to define an indeterminate (or "open") start time. It sets limits on a continuous period of time over which the contents are accessible. The period starts on the first access and ends after the duration has passed or the expiration date is reached, whichever comes first. For example, if the right gives 10 hours of continuous access, the use-duration would begin when the first access was made and end 10 hours later.

Grammar element 1515 **"Meter-Time: = Time-Remaining: Remaining-Use"** is used to define a "meter time," that is, a measure of the time that the right is actually exercised. It differs from the Sliding-Interval specification in that the time that the digital work is in use need not be continuous. For example, if the rights guarantee three days of access, those days could be spread out over a month. With this specification, the rights can be exercised until the meter time is exhausted or the expiration date is reached, whichever comes first.

Remaining-Use: = Time-Unit
Start-Time: = Time-Unit
Use-Duration: = Time-Unit
All of the time specifications include time-unit specifications in their ultimate instantiation.

Security Class and Authorization Specification

The present invention provides for various security mechanisms to be introduced into a distribution or use scheme. Grammar element 1516 **"Access-Spec : = ({SC: Security-Class} {Authorization: Authorization-ID*} {Other-Authorization: Authorization-ID*} {Ticket: Ticket-ID})"** provides a means for restricting access and transmission. Access specifications can specify a required security class for a repository to exercise a right or a required authorization test that must be satisfied.

The keyword **"SC:"** is used to specify a minimum security level for the repositories involved in the access. If "SC:" is not specified, the lowest security level is acceptable.

The optional **"Authorization:"** keyword is used to specify required authorizations on the same repository as the work. The optional **"Other-Authorization:"** keyword is used to specify required authorizations on the other repository in the transaction.

The optional **"Ticket:"** keyword specifies the identity of a ticket required for the transaction. A transaction involving digital tickets must locate an appropriate digital ticket agent who can "punch" or otherwise validate the ticket before the transaction can proceed. Tickets are described in greater detail below.

In a transaction involving a repository and a document server, some usage rights may require that the repository have a particular authorization, that the server have some authorization, or that both repositories have (possibly different) authorizations. Authorizations themselves are digital works (hereinafter referred to as an authorization object) that can be moved between repositories in the same manner as other digital works. Their copying and transferring is subject to the same rights and fees as other digital works. A repository is said to have an authorization if that authorization object is contained within the repository.

In some cases, an authorization may be required from a source other than the document server and repository. An authorization object referenced by an Authorization-ID can contain digital address information to be used to set up a communications link between a repository and the authorization source. These are analogous to phone numbers. For such access tests, the communication would need to be established and authorization obtained before the right

could be exercised.

For one-time usage rights, a variant on this scheme is to have a digital ticket. A ticket is presented to a digital ticket agent, whose type is specified on the ticket. In the simplest case, a certified generic ticket agent, available on all repositories, is available to "punch" the ticket. In other cases, the ticket may contain addressing information for locating a "special" ticket agent. Once a ticket has been punched, it cannot be used again for the same kind of transaction (unless it is unpunched or refreshed in the manner described below.) Punching includes marking the ticket with a timestamp of the date and time it was used. Tickets are digital works and can be copied or transferred between repositories according to their usage rights.

In the currently preferred embodiment, a "punched" ticket becomes "unpunched" or "refreshed" when it is copied or extracted. The Copy and Extract operations save the date and time as a property of the digital ticket. When a ticket agent is given a ticket, it can simply check whether the digital copy was made after the last time that it was punched. Of course, the digital ticket must have the copy or extract usage rights attached thereto.

The capability to unpunch a ticket is inportant in the following cases:

- A digital work is circulated at low cost with a limitation that it can be used only once.
- A digital work is circulated with a ticket that can be used once to give discounts on purchases of other works.
- A digital work is circulated with a ticket (included in the purchase price and possibly embedded in the work) that can be used for a future upgrade.

In each of these cases, if a paid copy is made of the digital work (including the ticket) the new owner would expect to get a fresh (unpunched) ticket, whether the copy seller has used the work or not. In contrast, loaning a work or simply transferring it to another repository should not revitalize the ticket.

Usage Fees and Incentives Specification

The billing for use of a digital work is fundamental to a commercial distribution system. Grammar Element 1517 **"Fee-Spec: = {Scheduled-Discount} Regular-Fee-Spec I Scheduled-Fee-Spec I Markup-Spec"** provides a range of options for billing for the use of digital works.

A key feature of this approach is the development of low-overhead billing for transactions in potentially small amounts. Thus, it becomes feasible to collect fees of only a few cents each for thousands of transactions.

The grammar differentiates between uses where the charge is per use from those where it is metered by the time unit. Transactions can support fees that the user pays for using a digital work as well as incentives paid by the right grantor to users to induce them to use or distribute the digital work.

The optional scheduled discount refers to the rest of the fee specification--discounting it by a percentage over time. If it is not specified, then there is no scheduled discount. Regular fee specifications are constant over time. Scheduled fee specifications give a schedule of dates over which the fee specifications change. Markup specifications are used in d-blocks for adding a percentage to the fees already being charged.

Grammar Element 1518 **"Scheduled-Discount: = (Scheduled-Discount: (Time-Spec Percentage)*)"** A Scheduled-Discount is a essentially a scheduled modifier of any other fee specification for this version of the right of the digital work. (It does not refer to children or parent digital works or to other versions of rights.). It is a list of pairs of times and percentages. The most recent time in the list that has not yet passed at the time of the transaction is the one in effect. The percentage gives the discount percentage. For example, the number 10 refers to a 10% discount.

Grammar Element 1519 **"Regular-Fee-Spec : = ({Fee: I Incentive: } [Per-Use-Spec I Metered-Rate-Spec I Best-Price-Spec I Call-For-Price-Spec] {Min: Money-Unit Per: Time-Spec}{Max: Money-Unit Per: Time-Spec} To: Account-ID)"** provides for several kinds of fee specifications.

Fees are paid by the copy-owner/user to the revenue-owner if **Fee**: is specified. Incentives are paid by the revenue-owner to the user if Incentive: is specified. If the **Min**: specification is given, then there is a minimum fee to be charged per time-spec unit for its use. If the **Max**: specification is given, then there is a maximum fee to be charged per time-spec for its use. When **Fee**: is specified, **Account-ID** identifies the account to which the fee is to be paid. When Incentive: is specified, **Account-ID** identifies the account from which the fee is to be paid.

Grammar element 1520 **"Per-Use-Spec: = Per-Use: Money-unit"** defines a simple fee to be paid every time the right is exercised, regardless of how much time the transaction takes.

Grammar element 1521 **"Metered-Rate-Spec := Metered: Money-Unit Per: Time-Spec"** defines a metered-rate fee paid according to how long the right is exercised. Thus, the time it takes to complete the transaction determines the fee.

Grammar element 1522 **"Best-Price-Spec := Best-Price: Money-unit Max: Money-unit"** is used to specify a best-price that is determined when the account is settled. This specification is to accommodate special deals, rebates, and pricing that depends on information that is not available to the repository. All fee specifications can be combined

with tickets or authorizations that could indicate that the consumer is a wholesaler or that he is a preferred customer, or that the seller be authorized in some way. The amount of money in the **Max**: field is the maximum amount that the use will cost. This is the amount that is tentatively debited from the credit server. However, when the transaction is ultimately reconciled, any excess amount will be returned to the consumer in a separate transaction.

Grammar element 1523 **"Call-For-Price-Spec : = Call-For-Price "** is similar to a **"Best-Price-Spec"** in that it is intended to accommodate cases where prices are dynamic. **A Call-For-Price Spec** requires a communication with a dealer to determine the price. This option cannot be exercised if the repository cannot communicate with a dealer at the time that the right is exercised. It is based on a secure transaction whereby the dealer names a price to exercise the right and passes along a deal certificate which is referenced or included in the billing process.

Grammar element 1524 **"Scheduled-Fee-Spec: = (Schedule: (Time-Spec Regular-Fee-Spec)*)"** is used to provide a schedule of dates over which the fee specifications change. The fee specification with the most recent date not in the future is the one that is in effect. This is similar to but more general than the scheduled discount. It is more general, because it provides a means to vary the fee agreement for each time period.

Grammar element 1525 **"Markup-Spec: = Markup: percentage To: Account-ID"** is provided for adding a percentage to the fees already being charged. For example, a 5% markup means that a fee of 5% of cumulative fee so far will be allocated to the distributor. A markup specification can be applied to all of the other kinds of fee specifications. It is typically used in a shell provided by a distributor. It refers to fees associated with d-blocks that are parts of the current d-block. This might be a convenient specification for use in taxes, or in distributor overhead.

REPOSITORY TRANSACTIONS

When a user requests access to a digital work, the repository will initiate various transactions. The combination of transactions invoked will depend on the specifications assigned for a usage right. There are three basic types of transactions, Session Initiation Transactions, Financial Transactions and Usage Transactions. Generally, session initiation transactions are initiated first to establish a valid session. When a valid session is established, transactions corresponding to the various usage rights are invoked. Finally, request specific transactions are performed.

Transactions occur between two repositories (one acting as a server), between a repository and a document playback platform (e.g. for executing or viewing), between a repository and a credit server or between a repository and an authorization server. When transactions occur between more than one repository, it is assumed that there is a reliable communication channel between the repositories. For example, this could be a TCP/IP channel or any other commercially available channel that has built-in capabilities for detecting and correcting transmission errors. However, it is not assumed that the communication channel is secure. Provisions for security and privacy are part of the requirements for specifying and implementing repositories and thus form the need for various transactions.

Message Transmission

Transactions require that there be some communication between repositories. Communication between repositories occurs in units termed as messages. Because the communication line is assumed to be unsecure, all communications with repositories that are above the lowest security class are encrypted utilizing a public key encryption technique. Public key encryption is a well known technique in the encryption arts. The term key refers to a numeric code that is used with encryption and decryption algorithms. Keys come in pairs, where "writing keys" are used to encrypt data and "checking keys" are used to decrypt data. Both writing and checking keys may be public or private. Public keys are those that are distributed to others Private keys are maintained in confidence.

Key management and security is instrumental in the success of a public key encryption system. In the currently preferred embodiment, one or more master repositories maintain the keys and create the identification certificates used by the repositories.

When a sending repository transmits a message to a receiving repository, the sending repository encrypts all of its data using the public writing key of the receiving repository. The sending repository includes its name, the name of the receiving repository, a session identifier such as a nonce (described below), and a message counter in each message.

In this way, the communication can only be read (to a high probability) by the receiving repository, which holds the private checking key for decryption. The auxiliary data is used to guard against various replay attacks to security. If messages ever arrive with the wrong counter or an old nonce, the repositories can assume that someone is interfering with communication and the transaction terminated.

The respective public keys for the repositories to be used for encryption are obtained in the registration transaction described below.

Session Initiation Transactions

A usage transaction is carried out in a session between repositories. For usage transactions involving more than one repository, or for financial transactions between a repository and a credit server, a registration transaction is performed. A second transaction termed a login transaction, may also be needed to initiate the session. The goal of the registration transaction is to establish a secure channel between two repositories who know each others identities. As it is assumed that the communication channel between the repositories is reliable but not secure, there is a risk that a non-repository may mimic the protocol in order to gain illegitimate access to a repository.

The registration transaction between two repositories is described with respect to Figures 16 and 17. The steps described are from the perspective of a "repository-1" registering its identity with a "repository-2". The registration must be symmetrical so the same set of steps will be repeated for repository-2 registering its identity with repository-1. Referring to Figure 16, repository-1 first generates an encrypted registration identifier, step 1601 and then generates a registration message, step 1602. A registration message is comprised of an identifier of a master repository, the identification certificate for the repository-1 and an encrypted random registration identifier. The identification certificate is encrypted by the master repository in its private key and attests to the fact that the repository (here repository-1) is a bona fide repository. The identification certificate also contains a public key for the repository, the repository security level and a timestamp (indicating a time after which the certificate is no longer valid.) The registration identifier is a number generated by the repository for this registration. The registration identifier is unique to the session and is encrypted in repository-1's private key. The registration identifier is used to improve security of authentication by detecting certain kinds of communications based attacks. Repository-1 then transmits the registration message to repository-2, step 1603.

Upon receiving the registration message, repository-2 determines if it has the needed public key for the master repository, step 1604. If repository-2 does not have the needed public key to decrypt the identification certificate, the registration transaction terminates in an error, step 1618.

Assuming that repository-2 has the proper public key the identification certificate is decrypted, step 1605. Repository-2 saves the encrypted registration identifier, step 1606, and extracts the repository identifier, step 1607. The extracted repository identifier is checked against a "hotlist" of compromised document repositories, step 1608. In the currently preferred embodiment, each repository will contain "hotlists" of compromised repositories. If the repository is on the "hotlist', the registration transaction terminates in an error per step 1618. Repositories can be removed from the hotlist when their certificates expire, so that the list does not need to grow without bound. Also, by keeping a short list of hotlist certificates that it has previously received, a repository can avoid the work of actually going through the list. These lists would be encrypted by a master repository. A minor variation on the approach to improve efficiency would have the repositories first exchange lists of names of hotlist certificates, ultimately exchanging only those lists that they had not previously received. The "hotlists" are maintained and distributed by Master repositories.

Note that rather than terminating in error, the transaction could request that another registration message be sent based on an identification certificate created by another master repository. This may be repeated until a satisfactory identification certificate is found, or it is determined that trust cannot be established.

Assuming that the repository is not on the hotlist, the repository identification needs to be verified. In other words, repository-2 needs to validate that the repository on the other end is really repository-1. This is termed performance testing and is performed in order to avoid invalid access to the repository via a counterfeit repository replaying a recording of a prior session initiation between repository-1 and repository-2. Performance testing is initiated by repository-2 generating a performance message, step 1609. The performance message consists of a nonce, the names of the respective repositories, the time and the registration identifier received from repository-1. A nonce is a generated message based on some random and variable information (e.g. the time or the temperature.) The nonce is used to check whether repository-1 can actually exhibit correct encrypting of a message using the private keys it claims to have, on a message that it has never seen before. The performance message is encrypted using the public key specified in the registration message of repository-1. The performance message is transmitted to repository-1, step 1610, where it is decrypted by repository-1 using its private key, step 1611. Repository-1 then checks to make sure that the names of the two repositories are correct, step 1612, that the time is accurate, step 1613 and that the registration identifier corresponds to the one it sent, step 1614. If any of these tests fails, the transaction is terminated per step 1616. Assuming that the tests are passed, repository-1 transmits the nonce to repository-2 in the clear, step 1615. Repository-2 then compares the received nonce to the original nonce, step 1617. If they are not identical, the registration transaction terminates in an error per step 1618. If they are the same, the registration transaction has successfully completed.

At this point, assuming that the transaction has not terminated, the repositories exchange messages containing session keys to be used in all communications during the session and synchronize their clocks. Figure 17 illustrates the session information exchange and clock synchronization steps (again from the perspective of repository-1.) Referring to Figure 17, repository-1 creates a session key pair, step 1701. A first key is kept private and is used by repository-1 to encrypt messages. The second key is a public key used by repository-2 to decrypt messages. The

second key is encrypted using the public key of repository-2, step 1702 and is sent to repository-2, step 1703. Upon receipt, repository-2 decrypts the second key, step 1704. The second key is used to decrypt messages in subsequent communications. When each repository has completed this step, they are both convinced that the other repository is bona fide and that they are communicating with the original. Each repository has given the other a key to be used in decrypting further communications during the session. Since that key is itself transmitted in the public key of the receiving repository only it will be able to decrypt the key which is used to decrypt subsequent messages.

After the session information is exchanged, the repositories must synchronize their clocks Clock synchronization is used by the repositories to establish an agreed upon time base for the financial records of their mutual transactions. Referring back to Figure 17, repository-2 initiates clock synchronization by generating a time stamp exchange message, step 1705, and transmits it to repository- 1, step 1706. Upon receipt, repository-1 generates its own time stamp message, step 1707 and transmits it back to repository-2, step 1708. Repository-2 notes the current time, step 1709 and stores the time received from repository-1, step 1710. The current time is compared to the time received from repository-1, step 1711. The difference is then checked to see if it exceeds a predetermined tolerance (e.g. one minute), step 1712. If it does, repository-2 terminates the transaction as this may indicate tampering with the repository, step 1713. If not repository-2 computes an adjusted time delta, step 1714. The adjusted time delta is the difference between the clock time of repository-2 and the average of the times from repository-1 and repository-2.

To achieve greater accuracy, repository-2 can request the time again up to a fixed number of times (e.g. five times), repeat the clock synchronization steps, and average the results.

A second session initiation transaction is a Login transaction. The Login transaction is used to check the authenticity of a user requesting a transaction. A Login transaction is particularly prudent for the authorization of financial transactions that will be charged to a credit server. The Login transaction involves an interaction between the user at a user interface and the credit server associated with a repository. The information exchanged here is a login string supplied by the repository/credit server to identify itself to the user, and a Personal Identification Number (PIN) provided by the user to identify himself to the credit server. In the event that the user is accessing a credit server on a repository different from the one on which the user interface resides, exchange of the information would be encrypted using the public and private keys of the respective repositories.

Billing Transactions

Billing Transactions are concerned with monetary transactions with a credit server. Billing Transactions are carried out when all other conditions are satisfied and a usage fee is required for granting the request. For the most part, billing transactions are well understood in the state of the art. These transactions are between a repository and a credit server, or between a credit server and a billing clearinghouse. Briefly, the required transactions include the following:

- Registration and LOG IN transactions by which the repository and user establish their bona fides to a credit server. These transactions would be entirely internal in cases where the repository and credit server are implemented as a single system.
- Registration and LOG IN transactions, by which a credit server establishes its bona fides to a billing clearinghouse.
- An Assign-fee transaction to assign a charge. The information in this transaction would include a transaction identifier, the identities of the repositories in the transaction, and a list of charges from the parts of the digital work. If there has been any unusual event in the transaction such as an interruption of communications, that information is included as well.
- A Begin-charges transaction to assign a charge. This transaction is much the same as an assign-fee transaction except that it is used for metered use. It includes the same information as the assign-fee transaction as well as the usage fee information. The credit-server is then responsible for running a clock.
- An End-charges transaction to end a charge for metered use. (In a variation on this approach, the repositories would exchange periodic charge information for each block of time.)
- A report-charges transaction between a personal credit server and a billing clearinghouse. This transaction is invoked at least once per billing period. It is used to pass along information about charges. On debit and credit cards, this transaction would also be used to update balance information and credit limits as needed.

All billing transactions are given a transaction ID and are reported to the credit severs by both the server and the client. This reduces possible loss of billing information if one of the parties to a transaction loses a banking card and provides a check against tampering with the system.

Usage Transactions

After the session initiation transactions have been completed, the usage request may then be processed. To sim-

plify the description of the steps carried out in processing a usage request, the term requester is used to refer to a repository in the requester mode which is initiating a request, and the term server is used to refer to a repository in the server mode and which contains the desired digital work. In many cases such as requests to print or view a work, the requester and server may be the same device and the transactions described in the following would be entirely internal. In such instances, certain transaction steps, such as the registration transaction, need not be performed.

There are some common steps that are part of the semantics of all of the usage rights transactions. These steps are referred to as the common transaction steps. There are two sets --the "opening" steps and the "closing" steps. For simplicity, these are listed here rather than repeating them in the descriptions of all of the usage rights transactions.

Transactions can refer to a part of a digital work, a complete digital work, or a Digital work containing other digital works. Although not described in detail herein, a transaction may even refer to a folder comprised of a plurality of digital works. The term "work" is used to refer to what ever portion or set of digital works is being accessed.

Many of the steps here involve determining if certain conditions are satisfied. Recall that each usage right may have one or more conditions which must be satisfied before the right can be exercised. Digital works have parts and parts have parts. Different parts can have different rights and fees. Thus, it is necessary to verify that the requirements are met for ALL of the parts that are involved in a transaction For brevity, when reference is made to checking whether the rights exist and conditions for exercising are satisfied, it is meant that all such checking takes place for each of the relevant parts of the work.

Figure 18 illustrates the initial common opening and closing steps for a transaction. At this point it is assumed that registration has occurred and that a "trusted" session is in place. General tests are tests on usage rights associated with the folder containing the work or some containing folder higher in the file system hierarchy. These tests correspond to requirements imposed on the work as a consequence of its being on the particular repository, as opposed to being attached to the work itself. Referring to Figure 18, prior to initiating a usage transaction, the requester performs any general tests that are required before the right associated with the transaction can be exercised, step, 1801. For example, install, uninstall and delete rights may be implemented to require that a requester have an authorization certificate before the right can be exercised. Another example is the requirement that a digital ticket be present and punched before a digital work may be copied to a requester. If any of the general tests fail, the transaction is not initiated, step, 1802. Assuming that such required tests are passed, upon receiving the usage request, the server generates a transaction identifier that is used in records or reports of the transaction, step 1803. The server then checks whether the digital work has been granted the right corresponding to the requested transaction, step 1804. If the digital work has not been granted the right corresponding to the request, the transaction terminates, step 1805. If the digital work has been granted the requested right, the server then determines if the various conditions for exercising the right are satisfied. Time based conditions are examined, step 1806. These conditions are checked by examining the time specification for the the version of the right. If any of the conditions are not satisfied, the transaction terminates per step 1805.

Assuming that the time based conditions are satisfied, the server checks security and access conditions, step 1807. Such security and access conditions are satisfied if: 1) the requester is at the specified security class, or a higher security class, 2) the server satisfies any specified authorization test and 3) the requester satisfies any specified authorization tests and has any required digital tickets. If any of the conditions are not satisfied, the transaction terminates per step 1805.

Assuming that the security and access conditions are all satisfied, the server checks the copy count condition, step 1808. If the copy count equals zero, then the transaction cannot be completed and the transaction terminates per step 1805.

Assuming that the copy count does not equal zero, the server checks if the copies in use for the requested right is greater than or equal to any copy count for the requested right (or relevant parts), step 1809. If the copies in use is greater than or equal to the copy count, this indicates that usage rights for the version of the transaction have been exhausted. Accordingly, the server terminates the transaction, step 1805. If the copy count is less than the copies in use for the transaction the transaction can continue, and the copies in use would be incremented by the number of digital works requested in the transaction, step 1810.

The server then checks if the digital work has a "Loan" access right, step 1811. The "Loan" access right is a special case since remaining rights may be present even though all copies are loaned out. If the digital work has the "Loan" access right, a check is made to see if all copies have been loaned out, step 1812. The number of copies that could be loaned is the sum of the Copy-Counts for all of the versions of the loan right of the digital work. For a composite work, the relevant figure is the minimal such sum of each of the components of the composite work. If all copies have been loaned out, the remaining rights are determined, step 1813. The remaining-rights is determined from the remaining rights specifications from the versions of the Loan right. If there is only one version of the Loan right, then the determination is simple. The remaining rights are the ones specified in that version of the Loan right, or none if Remaining-Rights: is not specified. If there are multiple versions of the Loan right and all copies of all of the versions are loaned out, then the remaining rights is taken as the minimum set (intersection) of remaining rights across all of the versions of the loan right. The server then determines if the requested right is in the set of remaining rights, step 1814. If the

requested right is not in the set of remaining rights, the server terminates the transaction, step 1805.

If Loan is not a usage right for the digital work or if all copies have not been loaned out or the requested right is in the set of remaining rights, fee conditions for the right are then checked, step 1815. This will initiate various financial transactions between the repository and associated credit server. Further, any metering of usage of a digital work will commence. If any financial transaction fails, the transaction terminates per step 1805.

It should be noted that the order in which the conditions are checked need not follow the order of steps 1806-1815.

At this point, right specific steps are now performed and are represented here as step 1816. The right speci fi c steps are described in greater detail below.

The common closing transaction steps are now performed. Each of the closing transaction steps are performed by the server after a successful completion of a transaction. Referring back to Figure 18, the copies in use value for the requested right is decremented by the number of copies involved in the transaction, step 1817. Next, if the right had a metered usage fee specification, the server subtracts the elapsed time from the Remaining-Use-Time associated with the right for every part involved in the transaction, step 1818. Finally, if there are fee specifications associated with the right, the server initiates End-Charge financial transaction to confirm billing, step 1819.

Transmission Protocol

An important area to consider is the transmission of the digital work from the server to the requester. The transmission protocol described herein refers to events occurring after a valid session has been created. The transmission protocol must handle the case of disruption in the communications between the repositories. It is assumed that interference such as injecting noise on the communication channel can be detected by the integrity checks (e.g., parity, checksum, etc.) that are built into the transport protocol and are not discussed in detail herein.

The underlying goal in the transmission protocol is to preclude certain failure modes, such as malicious or accidental interference on the communications channel. Suppose, for example, that a user pulls a card with the credit server at a specific time near the end of a transaction. There should not be a vulnerable time at which "pulling the card" causes the repositories to fail to correctly account for the number of copies of the work that have been created. Restated, there should be no time at which a party can break a connection as a means to avoid payment after using a digital work.

If a transaction is interrupted (and fails), both repositories restore the digital works and accounts to their state prior to the failure, modulo records of the failure itself.

Figure 19 is a state diagram showing steps in the process of transmitting information during a transaction. Each box represents a state of a repository in either the server mode (above the central dotted line 1901) or in the requester mode (below the dotted line 1901). Solid arrows stand for transitions between states. Dashed arrows stand for message communications between the repositories. A dashed message arrow pointing to a solid transition arrow is interpreted as meaning that the transition takes place when the message is received. Unlabeled transition arrows take place unconditionally. Other labels on state transition arrows describe conditions that trigger the transition.

Referring now to Figure 19, the server is initially in a state 1902 where a new transaction is initiated via start message 1903. This message includes transaction information including a transaction identifier and a count of the blocks of data to be transferred. The requester, initially in a wait state 1904 then enters a data wait state 1905.

The server enters a data transmit state 1906 and transmits a block of data 1907 and then enters a wait for acknowledgement state 1908. As the data is received, the requester enters a data receive state 1909 and when the data blocks are completely received it enters an acknowledgement state 1910 and transmits an Acknowledgement message 1911 to the server.

If there are more blocks to send, the server waits until receiving an Acknowledgement message from the requester. When an Acknowledgement message is received it sends the next block to the requester and again waits for acknowledgement. The requester also repeats the same cycle of states.

If the server detects a communications failure before sending the last block, it enters a cancellation state 1912 wherein the transaction is cancelled. Similarly, if the requester detects a communications failure before receiving the last block it enters a cancellation state 1913.

If there are no more blocks to send, the server commits to the transaction and waits for the final Acknowledgement in state 1914. If there is a communications failure before the server receives the final Acknowledgement message, it still commits to the transaction but includes a report about the event to its credit server in state 1915. This report serves two purposes. It will help legitimize any claims by a user of having been billed for receiving digital works that were not completely received. Also it helps to identify repositories and communications lines that have suspicious patterns of use and interruption. The server then enters its completion state 1916.

On the requester side, when there are no more blocks to receive, the requester commits to the transaction in state 1917. If the requester detects a communications failure at this state, it reports the failure to its credit server in state 1918, but still commits to the transaction. When it has committed, it sends an acknowledgement message to the server. The server then enters its completion state 1919.

The key property is that both the server and the requester cancel a transaction if it is interrupted before all of the data blocks are delivered, and commits to it if all of the data blocks have been delivered.

There is a possibility that the server will have sent all of the data blocks (and committed) but the requester will not have received all of them and will cancel the transaction. In this case, both repositories will presumably detect a communications failure and report it to their credit server. This case will probably be rare since it depends on very precise timing of the communications failure. The only consequence will be that the user at the requester repository may want to request a refund from the credit services -- and the case for that refund will be documented by reports by both repositories.

To prevent loss of data, the server should not delete any transferred digital work until receiving the final acknowledgement from the requester. But it also should not use the file. A well known way to deal with this situation is called "two-phase commit" or 2PC.

Two-phase commit works as follows. The first phase works the same as the method described above. The server sends all of the data to the requester. Both repositories mark the transaction (and appropriate files) as uncommitted. The server sends a ready-to-commit message to the requester. The requester sends back an acknowledgement. The server then commits and sends the requester a commit message. When the requester receives the commit message, it commits the file.

If there is a communication failure or other crash, the requester must check back with the server to determine the status of the transaction. The server has the last word on this. The requester may have received all of the data, but if it did not get the final message, it has not committed. The server can go ahead and delete files (except for transaction records) once it commits, since the files are known to have been fully transmitted before starting the 2PC cycle.

There are variations known in the art which can be used to achieve the same effect. For example, the server could use an additional level of encryption when transmitting a work to a client. Only after the client sends a message acknowledging receipt does it send the key. The client then agrees to pay for the digital work. The point of this variation is that it provides a clear audit trail that the client received the work. For trusted systems, however, this variation adds a level of encryption for no real gain in accountability.

The transaction for specific usage rights are now discussed.

The Copy Transaction

A Copy transaction is a request to make one or more independent copies of the work with the same or lesser usage rights. Copy differs from the extraction right discussed later in that it refers to entire digital works or entire folders containing digital works. A copy operation cannot be used to remove a portion of a digital work.

- The requester sends the server a message to initiate the Copy Transaction. This message indicates the work to be copied, the version of the copy right to be used for the transaction, the destination address information (location in a folder) for placing the work, the file data for the work (including its size), and the number of copies requested.
- The repositories perform the common opening transaction steps.
- The server transmits the requested contents and data to the client according to the transmission protocol. If a Next-Set-Of-Rights has been provided in the version of the right, those rights are transmitted as the rights for the work. Otherwise, the rights of the original are transmitted. In any event, the Copy-Count field for the copy of the digital work being sent right is set to the number-of-copies requested.
- The requester records the work contents, data, and usage rights and stores the work. It records the date and time that the copy was made in the properties of the digital work.
- The repositories perform the common closing transaction steps.

The Transfer Transaction

A Transfer transaction is a request to move copies of the work with the same or lesser usage rights to another repository. In contrast with a copy transaction, this results in removing the work copies from the server.

- The requester sends the server a message to initiate the Transfer Transaction. This message indicates the work to be transferred, the version of the transfer right to be used in the transaction, the destination address information for placing the work, the file data for the work, and the number of copies involved.
- The repositories perform the common opening transaction steps.
- The server transmits the requested contents and data to the requester according to the transmission protocol. If a Next-Set-Of-Rights has been provided, those rights are transmitted as the rights for the work. Otherwise, the rights of the original are transmitted. In either case, the Copy-Count field for the transmitted rights are set to the number-of-copies requested.

- The requester records the work contents, data, and usage rights and stores the work.
- The server decrements its copy count by the number of copies involved in the transaction.
- The repositories perform the common closing transaction steps.
- If the number of copies remaining in the server is now zero, it erases the digital work from its memory.

The Loan Transaction

A loan transaction is a mechanism for loaning copies of a digital work. The maximum duration of the loan is determined by an internal parameter of the digital work. Works are automatically returned after a predetermined time period.

- The requester sends the server a message to initiate the Transfer Transaction. This message indicates the work to be loaned, the version of the loan right to be used in the transaction, the destination address information for placing the work, the number of copies involved, the file data for the work, and the period of the loan.
- The server checks the validity of the requested loan period, and ends with an error if the period is not valid. Loans for a loaned copy cannot extend beyond the period of the original loan to the server.
- The repositories perform the common opening transaction steps.
- The server transmits the requested contents and data to the requester. If a Next-Set-Of-Rights has been provided, those rights are transmitted as the rights for the work. Otherwise, the rights of the original are transmitted, as modified to reflect the loan period.
- The requester records the digital work contents, data, usage rights, and loan period and stores the work.
- The server updates the usage rights information in the digital work to reflect the number of copies loaned out.
- The repositories perform the common closing transaction steps.
- The server updates the usage rights data for the digital work. This may preclude use of the work until it is returned from the loan. The user on the requester platform can now use the transferred copies of the digital work. A user accessing the original repository cannot use the digital work , unless there are copies remaining. What happens next depends on the order of events in time.

Case 1. If the time of the loan period is not yet exhausted and the requester sends the repository a Return message.

- The return message includes the requester identification, and the transaction ID.
- The server decrements the copies-in-use field by the number of copies that were returned. (If the number of digital works returned is greater than the number actually borrowed, this is treated as an error.) This step may now make the work available at the server for other users.
- The requester deactivates its copies and removes the contents from its memory.

Case 2. If the time of the loan period is exhausted and the requester has not yet sent a Return message.

- The server decrements the copies-in-use field by the number digital works that were borrowed.
- The requester automatically deactivates its copies of the digital work. It terminates all current uses and erases the digital work copies from memory. One question is why a requester would ever return a work earlier than the period of the loan, since it would be returned automatically anyway. One reason for early return is that there may be a metered fee which determines the cost of the loan. Returning early may reduce that fee.

The Play Transaction

A play transaction is a request to use the contents of a work. Typically, to "play" a work is to send the digital work through some kind of transducer, such as a speaker or a display device. The request implies the intention that the contents will not be communicated digitally to any other system. For example, they will not be sent to a printer, recorded on any digital medium, retained after the transaction or sent to another repository.

This term "play" is natural for examples like playing music, playing a movie, or playing a video game. The general form of play means that a "player" is used to use the digital work. However, the term play covers all media and kinds of recordings. Thus one would "play" a digital work, meaning, to render it for reading, or play a computer program, meaning to execute it. For a digital ticket the player would be a digital ticket agent.

- The requester sends the server a message to initiate the play transaction. This message indicates the work to be played, the version of the play right to be used in the transaction, the identity of the player being used, and the file data for the work.

- The server checks the validity of the player identification and the compatibility of the player identification with the player specification in the right. It ends with an error if these are not satisfactory.
- The repositories perform the common opening transaction steps.
- The server and requester read and write the blocks of data as requested by the player according to the transmission protocol. The requester plays the work contents, using the player.
- When the player is finished, the player and the requester remove the contents from their memory.
- The repositories perform the common closing transaction steps.

The Print Transaction

A Print transaction is a request to obtain the contents of a work for the purpose of rendering them on a "printer." We use the term "printer" to include the common case of writing with ink on paper. However, the key aspect of "printing" in our use of the term is that it makes a copy of the digital work in a place outside of the protection of usage rights. As with all rights, this may require particular authorization certificates.

Once a digital work is printed, the publisher and user are bound by whatever copyright laws are in effect. However, printing moves the contents outside the control of repositories. For example, absent any other enforcement mechanisms, once a digital work is printed on paper, it can be copied on ordinary photocopying machines without intervention by a repository to collect usage fees. If the printer to a digital disk is permitted, then that digital copy is outside of the control of usage rights. Both the creator and the user know this, although the creator does not necessarily give tacit consent to such copying, which may violate copyright laws.

- The requester sends the server a message to initiate a Print transaction. This message indicates the work to be played, the identity of the printer being used, the file data for the work, and the number of copies in the request.
- The server checks the validity of the printer identification and the compatibility of the printer identification with the printer specification in the right. It ends with an error if these are not satisfactory.
- The repositories perform the common opening transaction steps.
- The server transmits blocks of data according to the transmission protocol.
- The requester prints the work contents, using the printer.
- When the printer is finished, the printer and the requester remove the contents from their memory.
- The repositories perform the common closing transaction steps.

The Backup Transaction

A Backup transaction is a request to make a backup copy of a digital work, as a protection against media failure. In the context of repositories, secure backup copies differ from other copies in three ways: (1) they are made under the control of a Backup transaction rather than a Copy transaction, (2) they do not count as regular copies, and (3) they are not usable as regular copies. Generally, backup copies are encrypted.

Although backup copies may be transferred or copied, depending on their assigned rights, the only way to make them useful for playing, printing or embedding is to restore them.

The output of a Backup operation is both an encrypted data file that contains the contents and description of a work, and a restoration file with an encryption key for restoring the encrypted contents. In many cases, the encrypted data file would have rights for "printing" it to a disk outside of the protection system, relying just on its encryption for security. Such files could be stored anywhere that was physically safe and convenient. The restoration file would be held in the repository. This file is necessary for the restoration of a backup copy. It may have rights for transfer between repositories.

- The requester sends the server a message to initiate a backup transaction. This message indicates the work to be backed up, the version of the backup right to be used in the transaction, the destination address information for placing the backup copy, the file data for the work.
- The repositories perform the common opening transaction steps.
- The server transmits the requested contents and data to the requester. If a Next-Set-Of-Rights has been provided, those rights are transmitted as the rights for the work. Otherwise, a set of default rights for backup files of the original are transmitted by the server.
- The requester records the work contents, data, and usage rights. It then creates a one-time key and encrypts the contents file. It saves the key information in a restoration file.
- The repositories perform the common closing transaction steps.

In some cases, it is convenient to be able to archive the large, encrypted contents file to secure offline storage,

such as a magneto-optical storage system or magnetic tape. This creation of a non-repository archive file is as secure as the encryption process. Such non-repository archive storage is considered a form of "printing" and is controlled by a print right with a specified "archive-printer." An archive-printer device is programmed to save the encrypted contents file (but not the description file) offline in such a way that it can be retrieved.

The Restore Transaction

A Restore transaction is a request to convert an encrypted backup copy of a digital work into a usable copy. A restore operation is intended to be used to compensate for catastrophic media failure. Like all usage rights, restoration rights can include fees and access tests including authorization checks.

- The requester sends the server a message to initiate a Restore transaction. This message indicates the work to be restored, the version of the restore right for the transaction, the destination address information for placing the work, and the file data for the work.
- The server verifies that the contents file is available (i.e. a digital work corresponding to the request has been backed-up.) If it is not, it ends the transaction with an error.
- The repositories perform the common opening transaction steps.
- The server retrieves the key from the restoration file. It decrypts the work contents, data, and usage rights.
- The server transmits the requested contents and data to the requester according to the transmission protocol. If a Next-Set-Of-Rights has been provided, those rights are transmitted as the rights for the work. Otherwise, a set of default rights for backup files of the original are transmitted by the server.
- The requester stores the digital work.
- The repositories perform the common closing transaction steps.

The Delete Transaction

A Delete transaction deletes a digital work or a number of copies of a digital work from a repository. Practically all digital works would have delete rights.

- The requester sends the server a message to initiate a delete transaction. This message indicates the work to be deleted, the version of the delete right for the transaction.
- The repositories perform the common opening transaction steps.
- The server deletes the file, erasing it from the file system.
- The repositories perform the common closing transaction steps.

The Directory Transaction

A Directory transaction is a request for information about folders, digital works, and their parts. This amounts to roughly the same idea as protection codes in a conventional file system like TENEX, except that it is generalized to the full power of the access specifications of the usage rights language.

The Directory transaction has the important role of passing along descriptions of the rights and fees associated with a digital work. When a user wants to exercise a right, the user interface of his repository implicitly makes a directory request to determine the versions of the right that are available. Typically these are presented to the user -- such as with different choices of billing for exercising a right. Thus, many directory transactions are invisible to the user and are exercised as part of the normal process of exercising all rights.

- The requester sends the server a message to initiate a Directory transaction. This message indicates the file or folder that is the root of the directory request and the version of the directory right used for the transaction.
- The server verifies that the information is accessible to the requester. In particular, it does not return the names of any files that have a HIDE-NAME status in their directory specifications, and it does not return the parts of any folders or files that have HIDE-PARTS in their specification. If the information is not accessible, the server ends the transaction with an error.
- The repositories perform the common opening transaction steps.
- The server sends the requested data to the requester according to the transmission protocol.
- The requester records the data.
- The repositories perform the common closing transaction steps.

The Folder Transaction

A Folder transaction is a request to create or rename a folder, or to move a work between folders. Together with Directory rights, Folder rights control the degree to which organization of a repository can be accessed or modified from another repository.

- The requester sends the server a message to initiate a Folder transaction. This message indicates the folder that is the root of the folder request, the version of the folder right for the transaction, an operation, and data. The operation can be one of create, rename, and move file. The data are the specifications required for the operation, such as a specification of a folder or digital work and a name.
- The repositories perform the common opening transaction steps.
- The server performs the requested operation -- creating a folder, renaming a folder, or moving a work between folders.
- The repositories perform the common closing transaction steps.

The Extract Transaction

A extract transaction is a request to copy a part of a digital work and to create a new work containing it. The extraction operation differs from copying in that it can be used to separate a part of a digital work from d-blocks or shells that place additional restrictions or fees on it. The extraction operation differs from the edit operation in that it does not change the contents of a work, only its embedding in d-blocks. Extraction creates a new digital work.

- The requester sends the server a message to initiate an Extract transaction. This message indicates the part of the work to be extracted, the version of the extract right to be used in the transaction, the destination address information for placing the part as a new work, the file data for the work, and the number of copies involved.
- The repositories perform the common opening transaction steps.
- The server transmits the requested contents and data to the requester according to the transmission protocol. If a Next-Set-Of-Rights has been provided, those rights are transmitted as the rights for the new work. Otherwise, the rights of the original are transmitted. The Copy-Count field for this right is set to the number-of-copies requested.
- The requester records the contents, data, and usage rights and stores the work. It records the date and time that new work was made in the properties of the work.
- The repositories perform the common closing transaction steps.

The Embed Transaction

An embed transaction is a request to make a digital work become a part of another digital work or to add a shell d-block to enable the adding of fees by a distributor of the work.

- The requester sends the server a message to initiate an Embed transaction. This message indicates the work to be embedded, the version of the embed right to be used in the transaction, the destination address information for placing the part as a a work, the file data for the work, and the number of copies involved.
- The server checks the control specifications for all of the rights in the part and the destination. If they are incompatible, the server ends the transaction with an error.
- The repositories perform the common opening transaction steps.
- The server transmits the requested contents and data to the requester according to the transmission protocol. If a Next-Set-Of-Rights has been provided, those rights are transmitted as the rights for the new work. Otherwise, the rights of the original are transmitted. The Copy-Count field for this right is set to the number-of-copies requested.
- The requester records the contents, data, and usage rights and embeds the work in the destination file.
- The repositories perform the common closing transaction steps.

The Edit Transaction

An Edit transaction is a request to make a new digital work by copying, selecting and modifying portions of an existing digital work. This operation can actually change the contents of a digital work. The kinds of changes that are permitted depend on the process being used. Like the extraction operation, edit operates on portions of a digital work. In contrast with the extract operation, edit does not affect the rights or location of the work. It only changes the contents. The kinds of changes permitted are determined by the type specification of the processor specified in the rights. In the currently preferred embodiment, an edit transaction changes the work itself and does not make a new work. However,

it would be a reasonable variation to cause a new copy of the work to be made.

- The requester sends the server a message to initiate an Edit transaction. This message indicates the work to be edited, the version of the edit right to be used in the transaction, the file data for the work (including its size), the process-ID for the process, and the number of copies involved.
- The server checks the compatibility of the process-ID to be used by the requester against any process-ID specification in the right. If they are incompatible, it ends the transaction with an error.
- The repositories perform the common opening transaction steps.
- The requester uses the process to change the contents of the digital work as desired. (For example, it can select and duplicate parts of it; combine it with other information; or compute functions based on the information. This can amount to editing text, music, or pictures or taking whatever other steps are useful in creating a derivative work.)
- The repositories perform the common closing transaction steps.

The edit transaction is used to cover a wide range of kinds of works. The category describes a process that takes as its input any portion of a digital work and then modifies the input in some way. For example, for text, a process for editing the text would require edit rights. A process for "summarizing" or counting words in the text would also be considered editing. For a music file, processing could involve changing the pitch or tempo, or adding reverberations, or any other audio effect. For digital video works, anything which alters the image would require edit rights. Examples would be colorizing, scaling, extracting still photos, selecting and combining frames into story boards, sharpening with signal processing, and so on.

Some creators may want to protect the authenticity of their works by limiting the kinds of processes that can be performed on them. If there are no edit rights, then no processing is allowed at all. A processor identifier can be included to specify what kind of process is allowed. If no process identifier is specified, then arbitrary processors can be used. For an example of a specific process, a photographer may want to allow use of his photograph but may not want it to be colorized. A musician may want to allow extraction of portions of his work but not changing of the tonality.

Authorization Transactions

There are many ways that authorization transactions can be defined. In the following, our preferred way is to simply define them in terms of other transactions that we already need for repositories. Thus, it is convenient sometimes to speak of "authorization transactions," but they are actually made up of other transactions that repositories already have.

A usage right can specify an authorization-ID, which identifies an authorization object (a digital work in a file of a standard format) that the repository must have and which it must process. The authorization is given to the generic authorization (or ticket) server of the repository which begins to interpret the authorization.

As described earlier, the authorization contains a server identifier, which may just be the generic authorization server or it may be another server. When a remote authorization server is required, it must contain a digital address. It may also contain a digital certificate.

If a remote authorization server is required, then the authorization process first performs the following steps:

- The generic authorization server attempts to set up the communications channel. (If the channel cannot be set up, then authorization fails with an error.)
- When the channel is set up, it performs a registration process with the remote repository. (If registration fails, then the authorization fails with an error.)
- When registration is complete, the generic authorization server invokes a "Play" transaction with the remote repository, supplying the authorization document as the digital work to be played, and the remote authorization server (a program) as the "player." (If the player cannot be found or has some other error, then the authorization fails with an error.)
- The authorization server then "plays" the authorization. This involves decrypting it using either the public key of the master repository that issued the certificate or the session key from the repository that transmitted it. The authorization server then performs various tests. These tests vary according to the authorization server. They include such steps as checking issue and validity dates of the authorization and checking any hot-lists of known invalid authorizations. The authorization server may require carrying out any other transactions on the repository as well, such as checking directories, getting some person to supply a password, or playing some other digital work. It may also invoke some special process for checking information about locations or recent events. The "script" for such steps is contained within the authorization server.
- If all of the required steps are completed satisfactorily, the authorization server completes the transaction normally, signaling that authorization is granted.

The Install Transaction

An Install transaction is a request to install a digital work as runnable software on a repository. In a typical case, the requester repository is a rendering repository and the software would be a new kind or new version of a player. Also in a typical case, the software would be copied to file system of the requester repository before it is installed.

- The requester sends the server an Install message. This message indicates the work to be installed, the version of the Install right being invoked, and the file data for the work (including its size).
- The repositories perform the common opening transaction steps.
- The requester extracts a copy of the digital certificate for the software. If the certificate cannot be found or the master repository for the certificate is not known to the requester, the transaction ends with an error.
- The requester decrypts the digital certificate using the public key of the master repository, recording the identity of the supplier and creator, a key for decrypting the software, the compatibility information, and a tamper-checking code. (This step certifies the software.)
- The requester decrypts the software using the key from the certificate and computes a check code on it using a 1-way hash function. If the check-code does not match the tamper-checking code from the certificate, the installation transaction ends with an error. (This step assures that the contents of the software, including the various scripts, have not been tampered with.)
- The requester retrieves the instructions in the compatibility-checking script and follows them. If the software is not compatible with the repository, the installation transaction ends with an error. (This step checks platform compatibility.)
- The requester retrieves the instructions in the installation script and follows them. If there is an error in this process (such as insufficient resources), then the transaction ends with an error. Note that the installation process puts the runnable software in a place in the repository where it is no longer accessible as a work for exercising any usage rights other than the execution of the software as part of repository operations in carrying out other transactions.
- The repositories perform the common closing transaction steps.

The Uninstall Transaction

An Uninstall transaction is a request to remove software from a repository. Since uncontrolled or incorrect removal of software from a repository could compromise its behavioral integrity, this step is controlled.

- The requester sends the server an Uninstall message. This message indicates the work to be uninstalled, the version of the Uninstall right being invoked, and the file data for the work (including its size).
- The repositories perform the common opening transaction steps.
- The requester extracts a copy of the digital certificate for the software. If the certificate cannot be found or the master repository for the certificate is not known to the requester, the transaction ends with an error.
- The requester checks whether the software is installed. If the software is not installed, the transaction ends with an error.
- The requester decrypts the digital certificate using the public key of the master repository, recording the identity of the supplier and creator, a key for decrypting the software, the compatibility information, and a tamper-checking code. (This step authenticates the certification of the software, including the script for uninstalling it.)
- The requester decrypts the software using the key from the certificate and computes a check code on it using a 1-way hash function. If the check-code does not match the tamper-checking code from the certificate, the installation transaction ends with an error. (This step assures that the contents of the software, including the various scripts, have not been tampered with.)
- The requester retrieves the instructions in the uninstallation script and follows them. If there is an error in this process (such as insufficient resources), then the transaction ends with an error.
- The repositories perform the common closing transaction steps.

Claims

1. A system for controlling the distribution and use of digital works having a mechanism for reporting fees based on the distribution and use of digital works, said system comprising:

   means for attaching usage rights to a digital work, each of said usage rights specifying how a digital work may be used or distributed, each of said usage rights specifying usage fee information, said usage fee information

comprising a fee type and fee parameters which define a fee to be paid in connection with the exercise of said usage right;
a communication medium for coupling repositories to enable communication between repositories; and
a plurality of repositories, each of said repositories comprising:
an external interface for removably coupling to said communications medium;
storage means for storing digital works having attached usage rights and fees;
requesting means for generating a request to access a digital work stored in another of said plurality of repositories, said request indicating a particular usage right; and
processing means for processing requests to access digital works stored in said storage means and for generating fee transactions when a request indicates a usage right that is attached to a digital work and said usage right specifies usage fee information;
each of said plurality of repositories being removably coupled to a credit server, said credit server being arranged for recording fee transactions from said repository and subsequently reporting said fee transactions to a billing clearinghouse.

2. The fee reporting system as recited in Claim 1 wherein said fee type of said fee information is a metered use fee, a per use fee, a best price fee, a scheduled fee,or a mark-up fee.

3. A method for reporting fees associated with the distribution and use of digital works in a system for controlling the distribution and use of digital works, said method comprising the steps of:

a) attaching one or more usage rights to a digital work, each of said one or more usage rights comprising an indicator of how said digital work may be distributed or used and a usage fee to be paid upon exercise of said right;
b) storing said digital work and attached one or more usage rights in a server repository, said server repository controlling access to said digital work;
c) said server repository receiving a request to access said digital work from a requesting repository;
d) said server repository identifying a usage right associated with said access request;
e) said server repository determining if said identified usage right is the same as one of said one or more usage rights attached to said digital work;
f) if said identified usage right is not the same as any one of said one or more usage rights attached to said digital work, said server repository denying access to said digital work;
g) if said usage right is included with said digital work, said server repository determining if a usage fee is associated with the exercise of said usage right;
h) if a usage fee is associated with usage right, said server repository calculating said usage fee;
i) said server repository transmitting a first assign fee transaction identifying said requesting repository as a payer for said usage fee to a first credit server;
j) said requesting repository transmitting a second assign fee transaction identifying said requesting repository as a payer for said usage fee to a second credit server;
k) said server repository transmitting said digital work to said requesting repository;
l) said server repository transmitting a first confirm fee transaction to said first credit server; and
m) said requesting repository transmitting a second confirm fee transaction to said second credit server.

4. The method as recited in Claim 3 wherein said digital work is comprised of a plurality of independent digital works and said step of said server calculating said usage fee is further comprised of the step of reporting the usage fees for each of the plurality of independent digital works.

5. A method for reporting fees associated with the distribution and use of digital works in a system for controlling the distribution and use of digital works, said method comprising the steps of:

a) attaching one or more usage rights to a digital work, each of said one or more usage rights comprising an indicator of how said digital work may be distributed or used and a usage fee to be paid for exercise of said right;
b) storing said digital work and said attached one or more usage rights in a server repository, said server repository controlling access to said digital work;
c) said server repository receiving a request to access said digital work from a requesting repository;
d) said server repository identifying a usage right associated with said access request;
e) said server repository determining if said digital work has attached thereto said identified usage right;
f) if said identified usage right is not attached to said digital work, said server repository denying access to

said digital work;
g) if said usage right is attached to said digital work, said server repository determining if a usage fee is associated with the exercise of said usage right;
h) if a usage fee is associated with said usage right, said server repository determining a fee type;
i) said server repository transmitting a first fee transaction identifying said requesting repository as a payee for said usage fee to a credit server, said first fee transaction being dependent on said determined fee type; and
k) said server repository transmitting said digital work to said requesting repository.

6. A system for controlling the distribution and utilization of digital works having a mechanism for reporting usage fees, said system comprising:

digital works comprising a first part for storing the digitally encoded data corresponding to a digital work and a second part for storing usage rights and fees for said digital work, said usage rights specifying how a digital work may be used or distributed and said usage fees specifying a fee to be paid in connection with the exercise of a corresponding usage right;
a plurality of repositories, each of said repositories comprising:
communication means for communicating with another of said plurality of repositories;
storage means for storing digital works;
requesting means for generating a request to access a digital work stored in another of said plurality of repositories, said request indicating a particular usage right;
processing means for processing requests to access digital works stored in said storage means and granting access when said particular usage right corresponds to a stored usage right stored in said digital work, said processing means generating fee transactions when said access is granted and said stored usage right specifies a fee;
each of said plurality of repositories being removably coupled to a credit server, said credit server being arranged for recording fee transactions from said repository and subsequently reporting said fee transactions to a billing clearinghouse.

7. The system as recited in Claim 6 wherein said storage means is further comprised of a first storage device for storing said first part of said digital work and a second storage device for storing said second part of said digital work.

8. A method for reporting fees associated with use of rendering digital works by a rendering device in a system for controlling the rendering of digital works by a rendering system, said rendering system comprised of a rendering repository and a rendering device, said rendering device utilizing a rendering digital work for rendering a digital work, said method comprising the steps of:

a) storing a first digital work in a server repository, said digital work specifying a first usage fee to be reported for a use of said first digital work;
b) storing a rendering digital work in said rendering repository, said first rendering digital work specifying a second usage fee to be reported for a use of said rendering digital work;
c) said server repository receiving a request to use said first digital work from said rendering repository;
d) said server repository determining if said request may be granted;
e) if said server repository determines that said request may not be granted, said server repository denying access to said first digital work;
f) if said server repository determines that said request may be granted, said server repository transmitting said digital work to said rendering repository;
g) said server repository transmitting a first fee transaction identifying said rendering repository as a payee for said first usage fee for use of said first digital work to a first credit server;
h) said rendering device rendering said first digital work using said rendering digital work; and
i) said rendering repository transmitting a second fee transaction identifying said rendering repository as a payee for said second usage fee for use of said rendering digital work to a second credit server.

9. The method as recited in Claim 8 further comprising the step of said rendering repository transmitting a third fee transaction identifying said rendering repository as a payee for said first usage fee for use of said first digital work to said second credit server.

10. The method as recited in Claim 9 wherein said rendering digital work is a set of coded rendering instructions for controlling said rendering device.

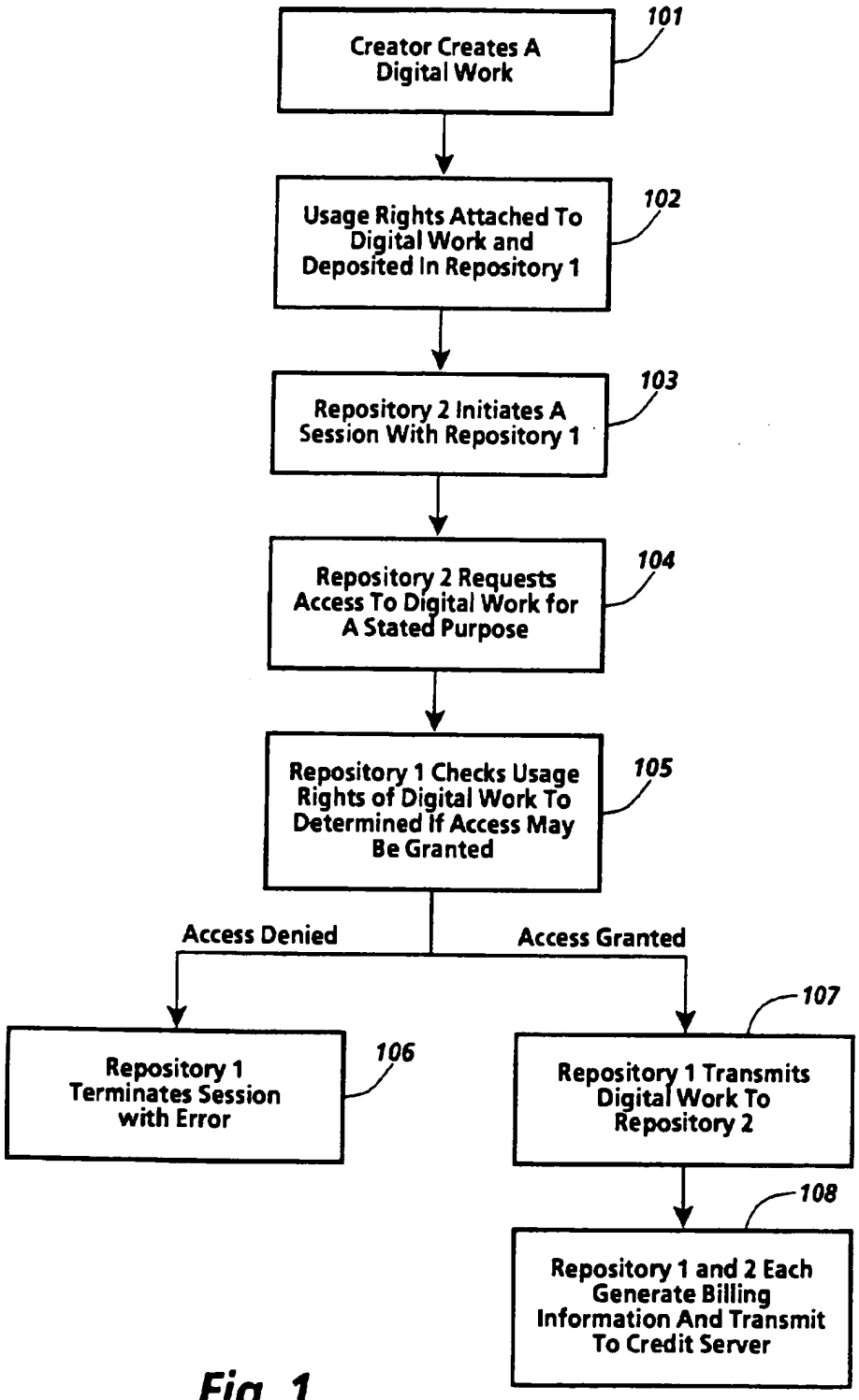


Fig. 1

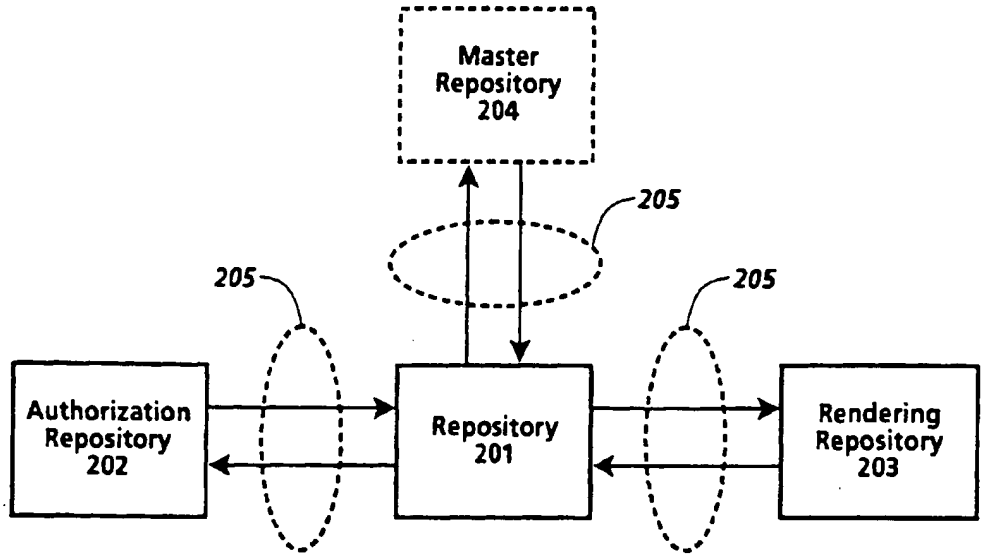


Fig. 2

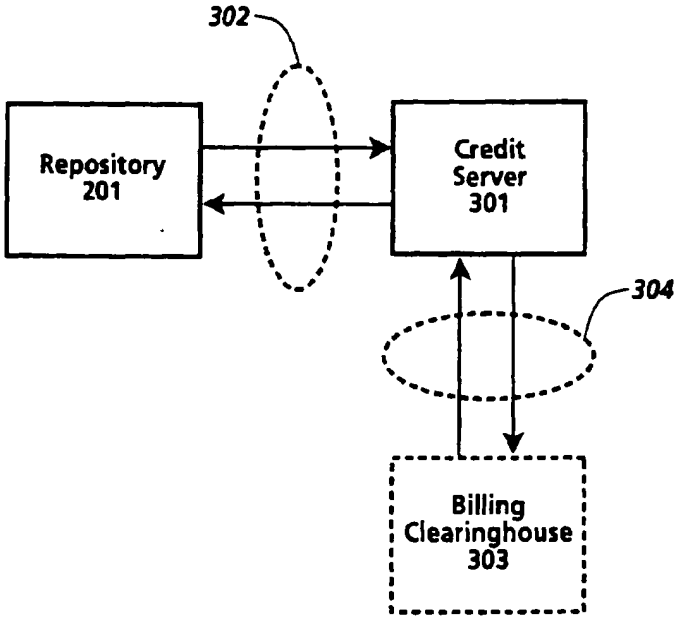


Fig. 3

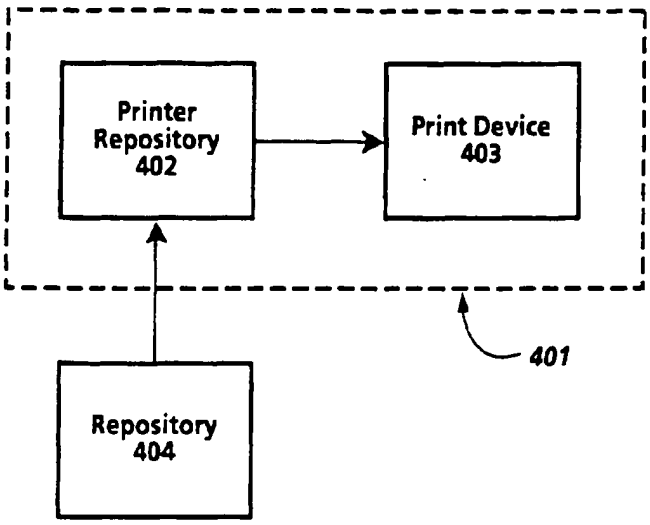


Fig. 4a

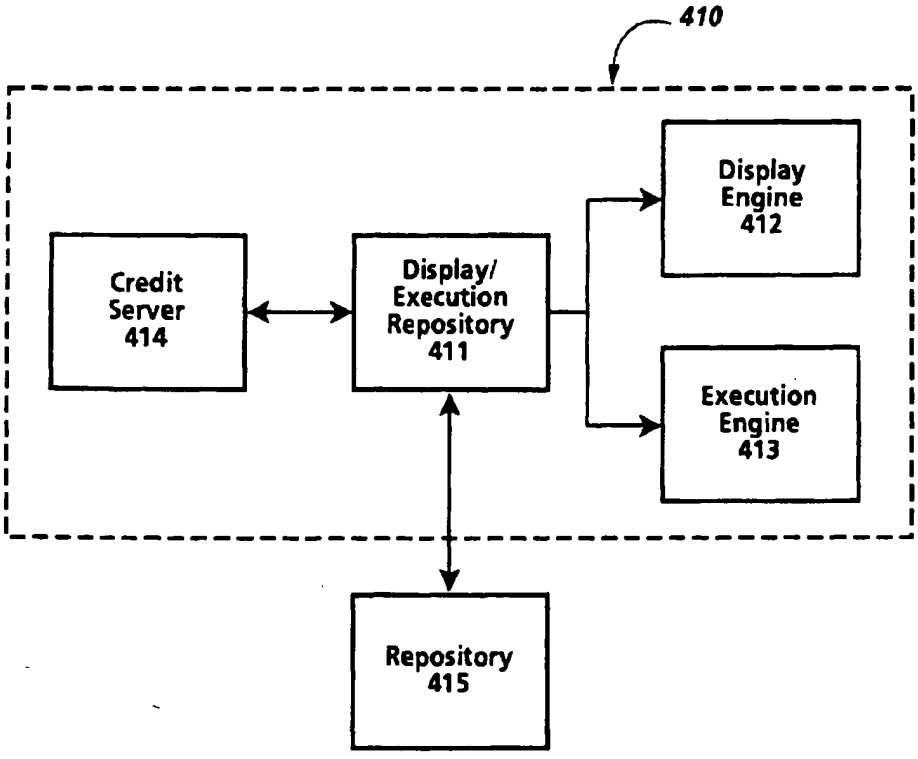


Fig. 4b

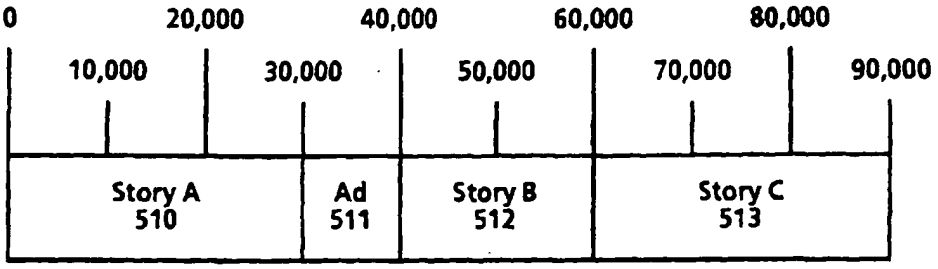


Fig. 5

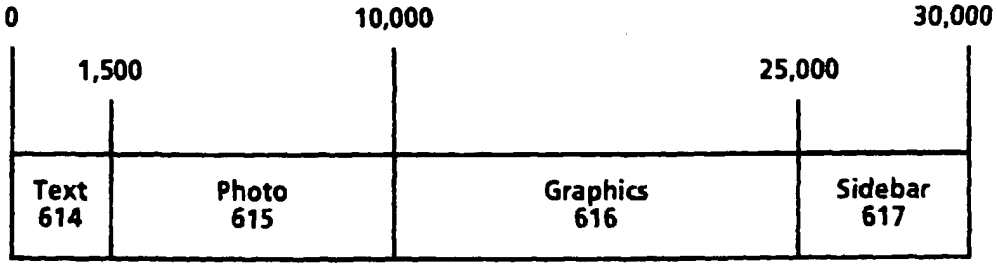


Fig. 6

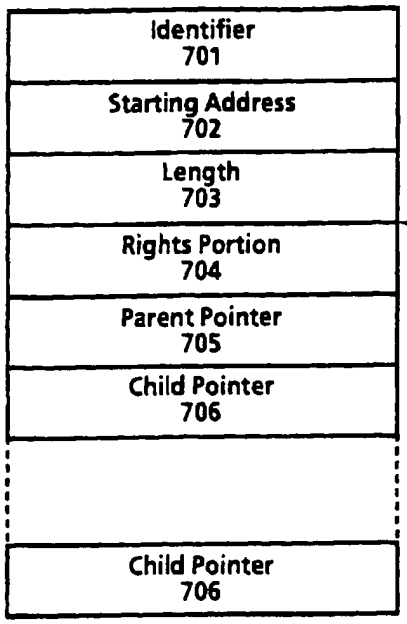


Fig. 7

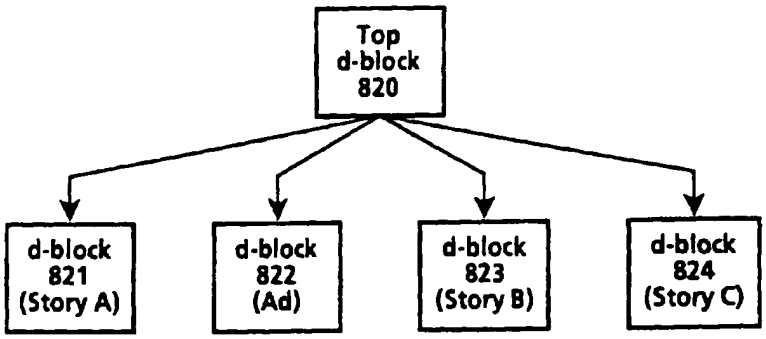


Fig. 8

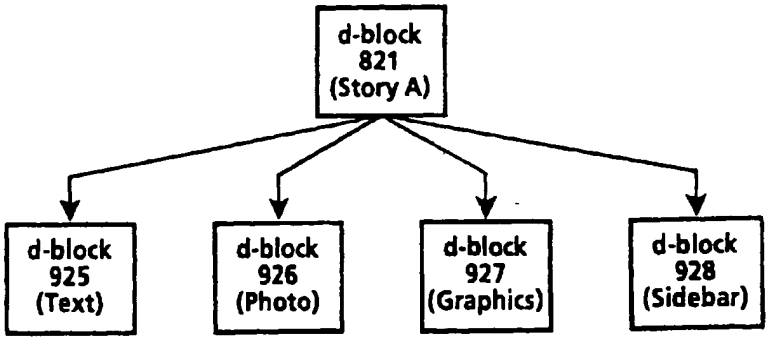


Fig. 9

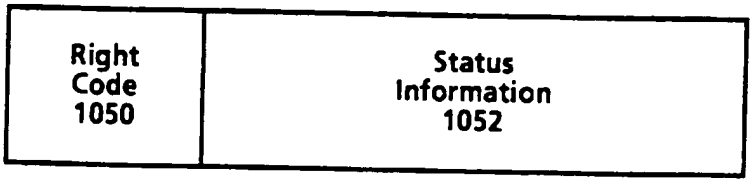


Fig.10

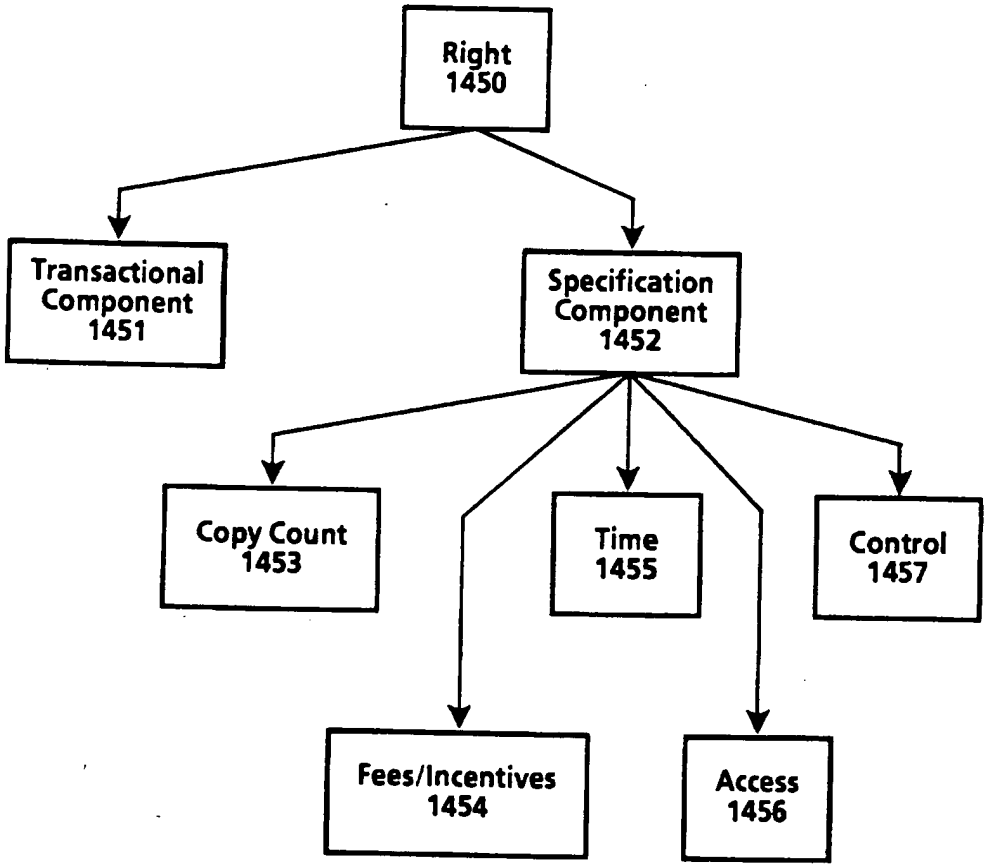


Fig.14

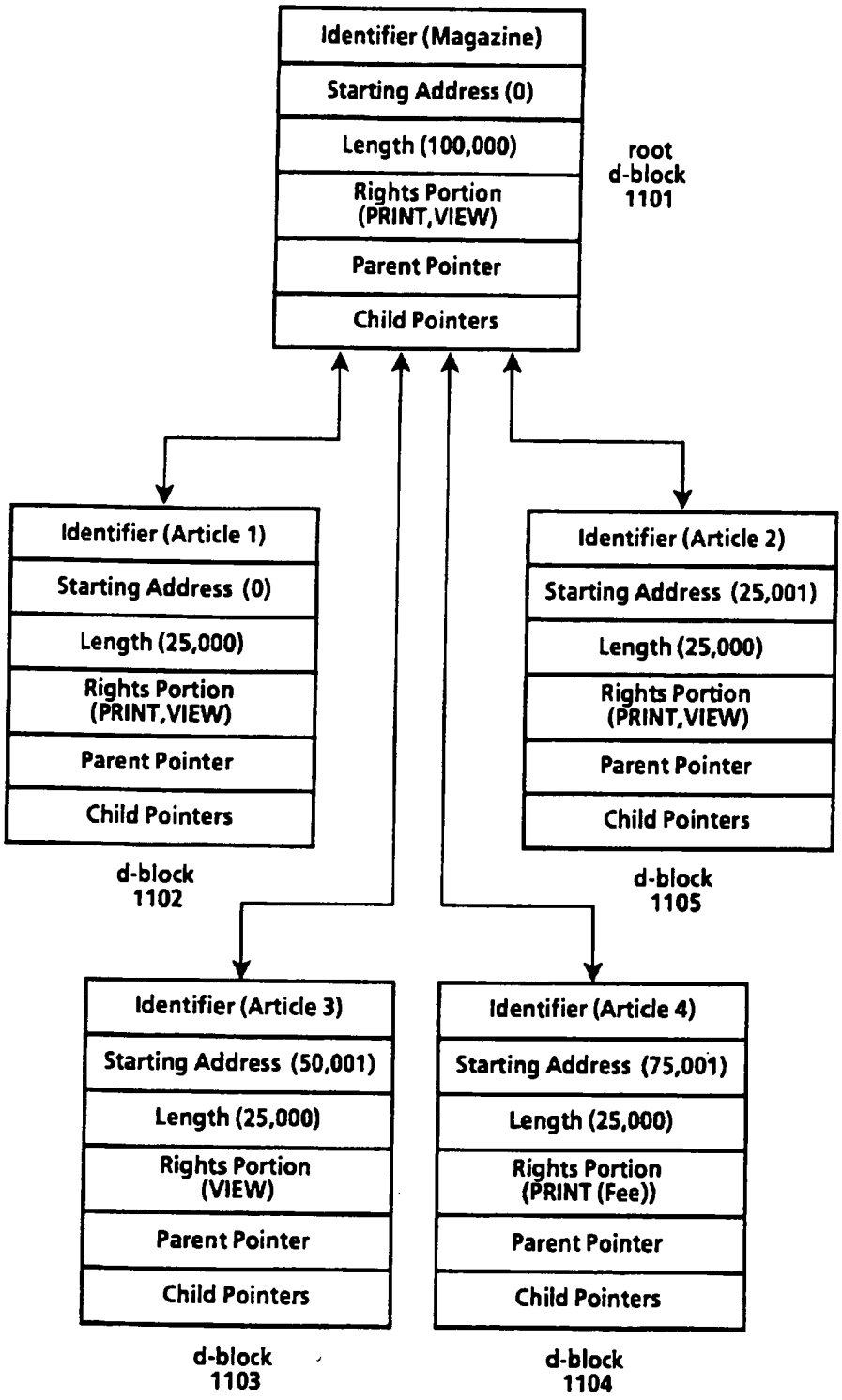


Fig.11

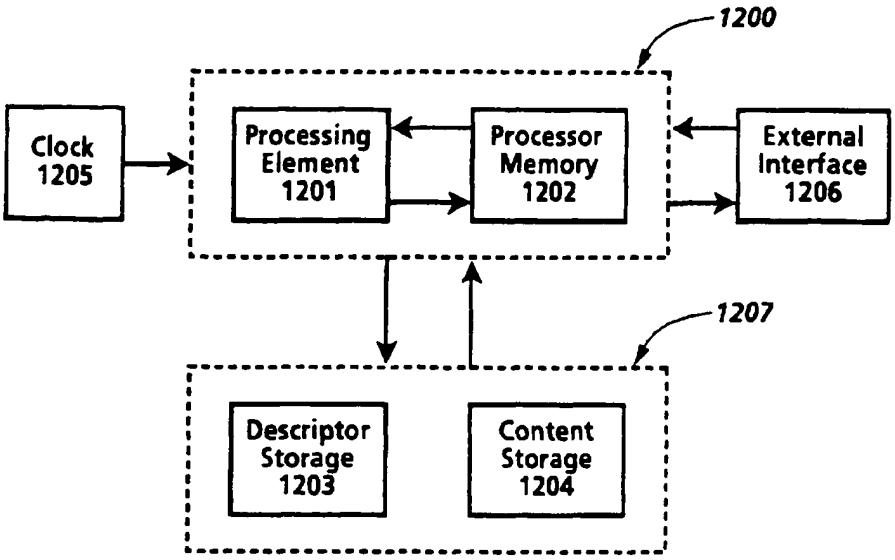


Fig.12

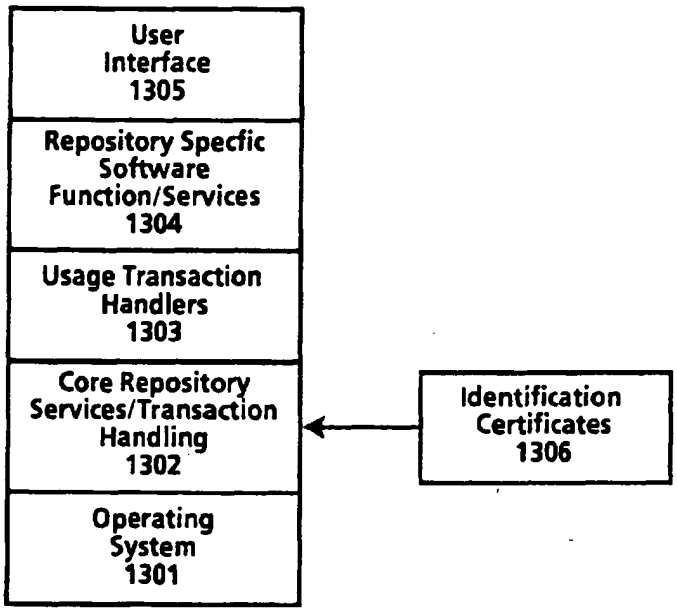


Fig.13

```
1501 ~Digital Work Rights: = (Rights*)
1502 ~Right : = (Right-Code {Copy-Count} {Control-Spec} {Time-Spec }
        {Access-Spec} {Fee-Spec})
   1503 ~Right-Code : = Render-Code | Transport-Code | File-Management-
          Code| Derivative-Works- Code | Configuration-Code
      1504 ~Render-Code : = [ Play : {Player: Player-ID} | Print: {Printer: Printer-ID}]
      1505 ~Transport-Code : = [Copy | Transfer | Loan {Remaining-Rights:
          Next-Set-of-Rights}]{(Next-Copy-Rights: Next-Set-of-Rights)}
      1506 ~File-Management-Code  : = Backup {Back-Up-Copy-Rights:
                                  Next-Set-of-Rights} | Restore | Delete | Folder
                                  | Directory {Name: Hide-Local | Hide-Remote}
                                  {Parts: Hide-Local | Hide-Remote}
      1507 ~Derivative-Works-Code : =  [Extract | Embed | Edit{Process:
                                  Process-ID}] {Next-Copy-Rights :
                                  Next-Set-of Rights}
      1508 ~Configuration-Code : = Install | Uninstall
      1509 ~Next-Set-of-Rights : = [(Add: Set-Of-Rights)} {(Delete:
          Set-Of-Rights)} {(Replace: Set-Of-Rights )}{(Keep: Set-Of-Rights )}
1510 ~Copy-Count : = (Copies:positive-integer | 0 | Unlimited)
1511 ~Control-Spec : = (Control: {Restrictable | Unrestrictable}
                    {Unchargeable | Chargeable})
1512 ~Time-Spec : = ({Fixed-Interval | Sliding-Interval | Meter-Time}
                  Until: Expiration-Date)
   1513 ~Fixed-Interval : = From: Start-Time
   1514 ~Sliding-Interval : = Interval: Use-Duration
   1515 ~Meter-Time: = Time-Remaining: Remaining-Use
1516 ~Access-Spec : = ({SC: Security-Class} {Authorization: Authorization-ID*}
        {Other-Authorization: Authorization-ID*} {Ticket: Ticket-ID})
1517 ~Fee-Spec: = {Scheduled-Discount} Regular-Fee-Spec | Scheduled-Fee-Spec |
                Markup-Spec
   1518 ~Scheduled-Discount: = Scheduled-Discount: (Scheduled-Discount:
                              (Time-Spec Percentage)*)
   1519 ~Regular-Fee-Spec : = ({Fee: | Incentive: } [Per-Use-Spec | Metered-Rate-
                          Spec | Best-Price-Spec | Call-For-Price-Spec]
                          {Min: Money-Unit Per: Time-Spec}{Max:
                          Money-Unit Per: Time-Spec} To: Account-ID)
      1520 ~Per-Use-Spec: = Per-Use: Money-unit
      1521 ~Metered-Rate-Spec : = Metered: Money-Unit Per: Time-Spec
      1522 ~Best-Price-Spec : = Best-Price: Money-unit Max: Money-unit
      1523 ~Call-For-Price-Spec : = Call-For -Price
   1524 ~Scheduled-Fee-Spec: = (Schedule: (Time-Spec Regular-Fee-Spec)* )
   1525 ~Markup-Spec: = Markup: percentage To: Account-ID
```

Fig.15

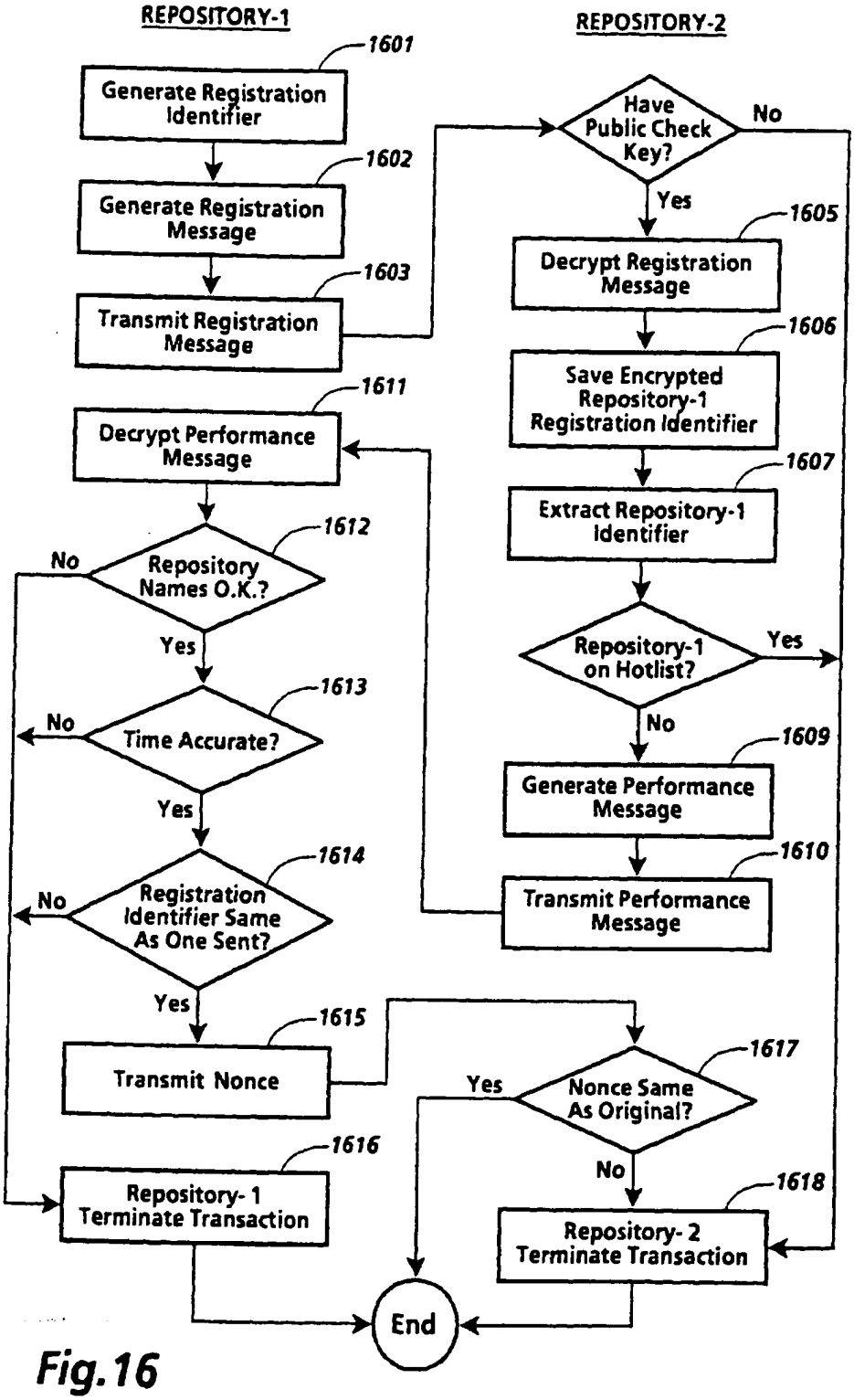


Fig.16

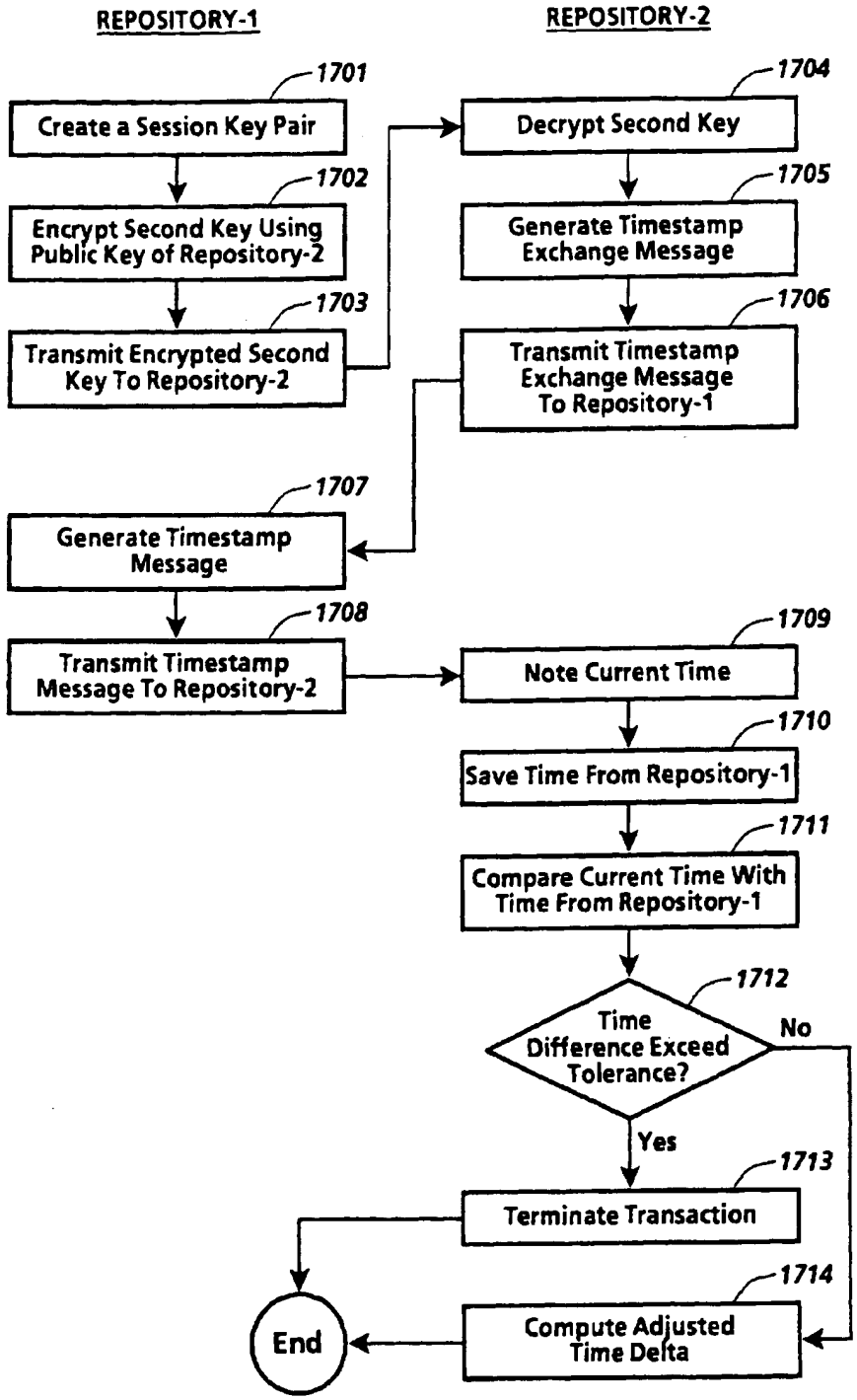


Fig.17

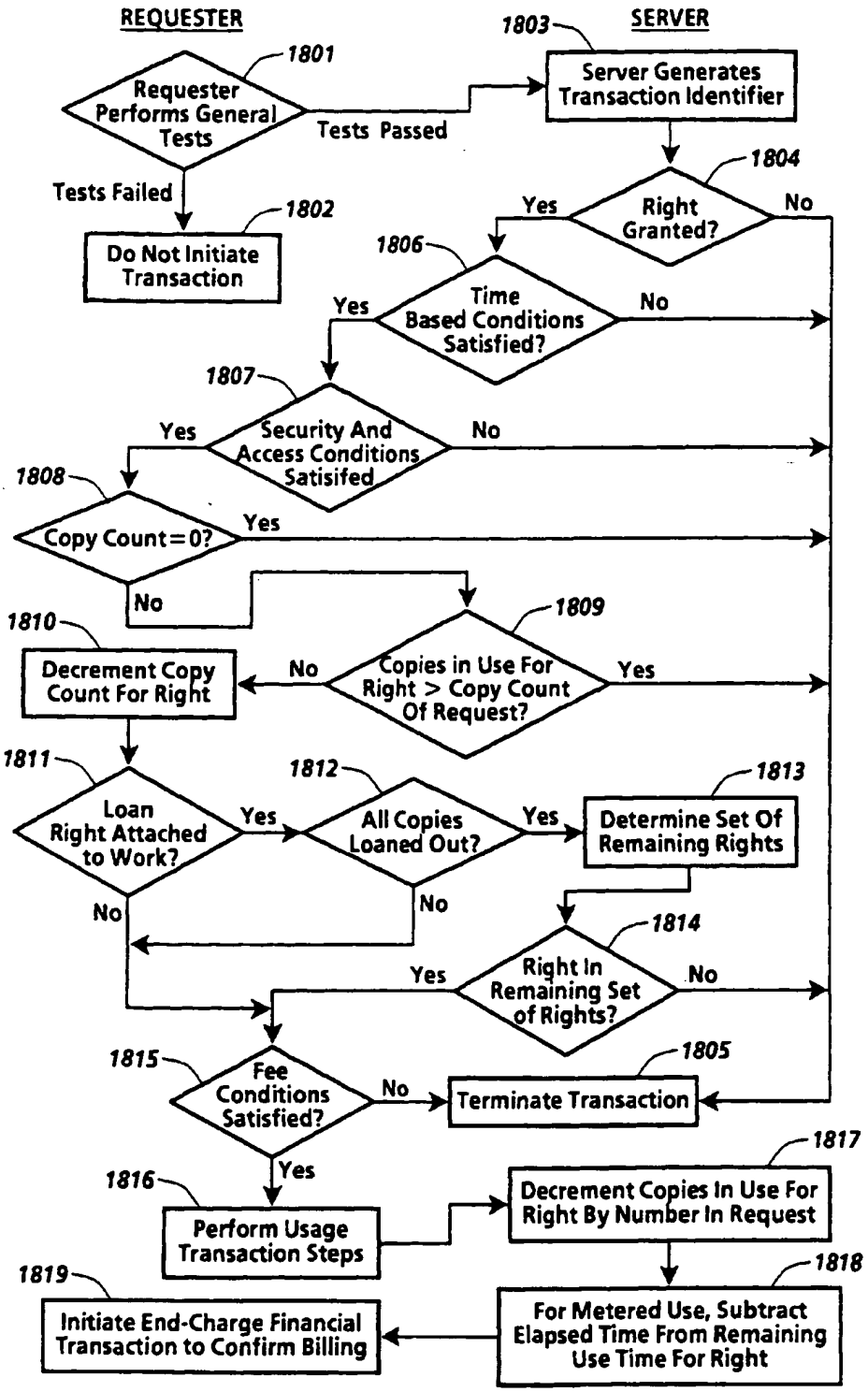


Fig. 18

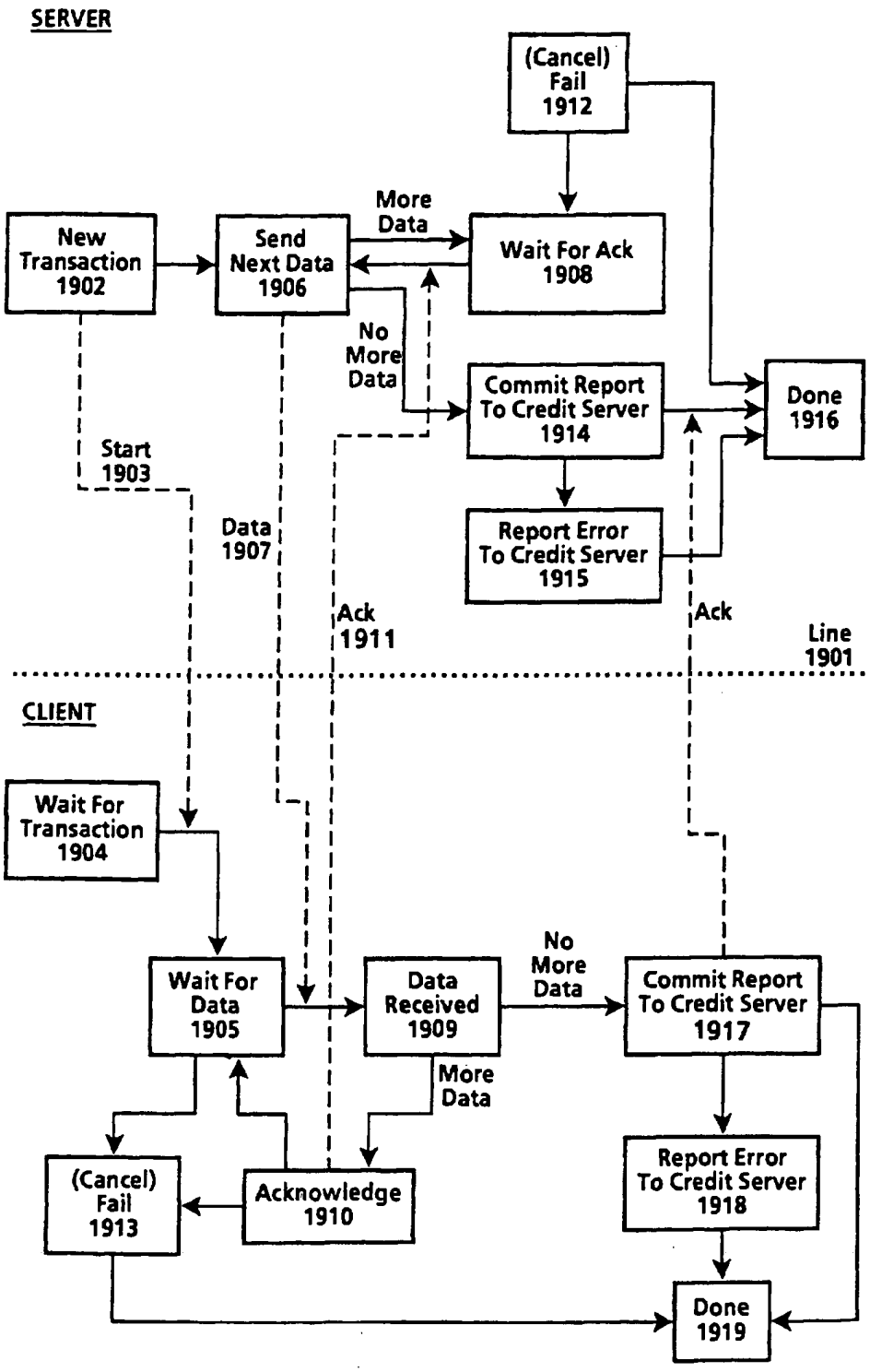


Fig.19

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 95 30 8414

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
A	WO-A-92 20022 (DIGITAL EQUIPMENT CORP.) * page 45, line 10 - page 64, line 17 * ---	1,3,5,6,8	G06F1/00 G06F17/60
A	TRANSACTIONS OF THE INSTITUTE OF ELECTRONICS, INFORMATION AND COMMUNICATION ENGINEERS OF JAPAN, vol. E73, no. 7, July 1990 TOKYO JP, pages 1133-1146, XP 000159229 MORI ET AL. 'SUPERDISTRIBUTION: THE CONCEPT AND THE ARCHITECTURE' * page 1135, left column, line 17 - page 1136, left column, line 40 * ---	1,3,5,6,8	
A	US-A-5 291 596 (MITA) * the whole document * ---	1,3,5,6,8	
A	GB-A-2 236 604 (SUN MICROSYSTEMS INC) * page 9, line 11 - page 20, line 15 * -----	1,3,5,6,8	
			TECHNICAL FIELDS SEARCHED (Int.Cl.6) G06F

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	1 April 1996	Moens, R

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 715 244 A1

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
05.06.1996 Bulletin 1996/23

(51) Int Cl.[6]: **G06F 1/00**

(21) Application number: **95308417.5**

(22) Date of filing: **23.11.1995**

(84) Designated Contracting States:
DE FR GB

(30) Priority: **23.11.1994 US 334041**

(71) Applicant: **XEROX CORPORATION**
Rochester New York 14644 (US)

(72) Inventor: **Stefik, Mark J.**
Woodside, California 94062 (US)

(74) Representative: **Goode, Ian Roy**
Rank Xerox Ltd
Patent Department
Parkway
Marlow Buckinghamshire SL7 1YL (GB)

(54) System for controlling the distribution and use of digital works utilizing a usage rights grammar

(57) A system for controlling use and distribution of digital works. The present invention allows the owner of a digital work to attach usage rights (1450) to their work. The usage rights define how the individual digital work may be used and distributed (1451). Instances of usage rights are defined using a flexible and extensible usage rights grammar. Conceptually, a right in the usage rights grammar is a label associated with a predetermined behavior and conditions to exercising the right. The behavior of a usage right is embodied in a predetermined set (1452) of usage transactions steps. The usage transaction steps further check all conditions (1453-1457) which must be satisfied before the right may be exercised. These usage transaction steps define a protocol for requesting the exercise of a right and the carrying out of a right.

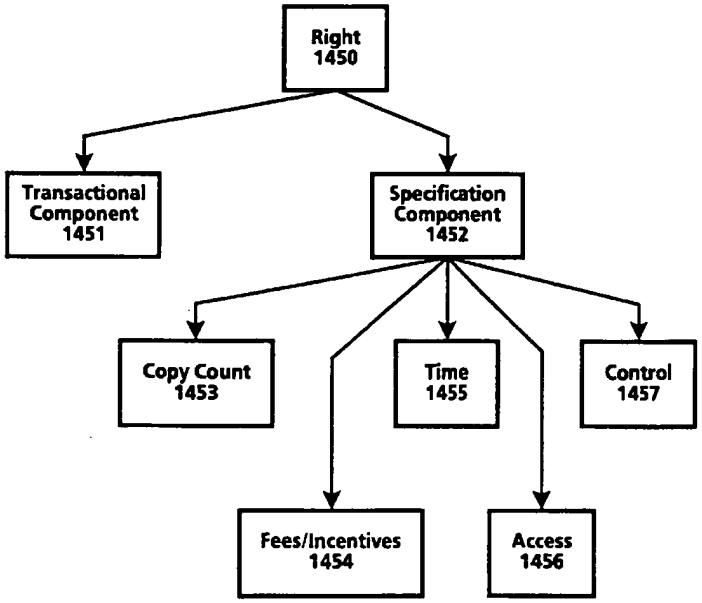


Fig.14

EP 0 715 244 A1

Printed by Jouve, 75001 PARIS (FR)

Description

The present invention relates to the field of distribution and usage rights enforcement for digitally encoded works.

A fundamental issue facing the publishing and information industries as they consider electronic publishing is how to prevent the unauthorized and unaccounted distribution or usage of electronically published materials. Electronically published materials are typically distributed in a digital form and recreated on a computer based system having the capability to recreate the materials. Audio and video recordings, software, books and multimedia works are all being electronically published. Companies in these industries receive royalties for each accounted for delivery of the materials, e.g. the sale of an audio CD at a retail outlet. Any unaccounted distribution of a work results in an unpaid royalty (e.g. copying the audio recording CD to another digital medium.)

The ease in which electronically published works can be "perfectly" reproduced and distributed is a major concern. The transmission of digital works over networks is commonplace. One such widely used network is the Internet. The Internet is a widespread network facility by which computer users in many universities, corporations and government entities communicate and trade ideas and information. Computer bulletin boards found on the Internet and commercial networks such as CompuServ and Prodigy allow for the posting and retrieving of digital information. Information services such as Dialog and LEXIS/NEXIS provide databases of current information on a wide variety of topics. Another factor which will exacerbate the situation is the development and expansion of the National Information Infrastructure (the NII). It is anticipated that, as the NII grows, the transmission of digital works over networks will increase many times over. It would be desirable to utilize the NII for distribution of digital works without the fear of widespread unauthorized copying.

The most straightforward way to curb unaccounted distribution is to prevent unauthorized copying and transmission. For existing materials that are distributed in digital form, various safeguards are used. In the case of software, copy protection schemes which limit the number of copies that can be made or which corrupt the output when copying is detected have been employed. Another scheme causes software to become disabled after a predetermined period of time has lapsed. A technique used for workstation based software is to require that a special hardware device must be present on the workstation in order for the software to run, e.g., see US-A-4,932,054 entitled "Method and Apparatus for Protecting Computer Software Utilizing Coded Filter Network in Conjunction with an Active Coded Hardware Device." Such devices are provided with the software and are commonly referred to as dongles.

Yet another scheme is to distribute software, but which requires a "key" to enable its use. This is employed in distribution schemes where "demos" of the software are provided on a medium along with the entire product. The demos can be freely used, but in order to use the actual product, the key must be purchased. These schemes do not hinder copying of the software once the key is initially purchased.

It is an object of the present invention to provide an improved system and method for controlling the use and distribution of digital works.

The invention accordingly provides a system and method as claimed in the accompanying claims.

A system for controlling use and distribution of digital works is disclosed. A digital work is any written, aural, graphical or video based work that has been translated to or created in a digital form, and which can be recreated using suitable rendering means such as software programs. The present invention allows the owner of a digital work to attach usage rights to their work. The usage rights define how the digital work may be used and distributed. These usage rights become part of the digital work and are always honored.

Instances of usage rights are defined using a flexible and extensible usage rights grammar. Conceptually, a right in the usage rights grammar is a label associated with a predetermined behavior and conditions to exercising the right. For example, a COPY right denotes that a copy of the digital work may be made. A condition to exercising the right is that the requester must pass certain security criteria. Conditions may also be attached to limit the right itself. For example, a LOAN right may be defined so as to limit the duration of which a work may be LOANed.

In the present invention a usage right is comprised of a right code along with the various conditions for exercising the right. Such conditions include a copy-count condition for limiting the number of times a right can be concurrently exercised (e.g. limit the number of copies on loan to some predetermined number), a security class condition for insuring that a repository has an appropriate level of security, access conditions for specifying access tests that must be passed, a time specification for indicating time based constraints for exercising a right and a fee specification for indicating usage fees for the exercise of a right. A digital work may have different versions of a right attached thereto. A version of a right will have the same right code as other versions, but the conditions (and typically the fees) would be different.

Digital works and their attached usage rights are stored in repositories. Digital works are transmitted between repositories. Repositories interact to exchange digital works according to a predetermined set of usage transactions steps. The behavior of a usage right is embodied in a predetermined set of usage transactions steps. The usage transaction steps further check all conditions which must be satisfied before the right may be exercised. These usage transaction steps define a protocol used by the repositories for requesting the exercise of a right and the carrying out of a right.

A system and method in accordance with the invention will now be described, by way of example, with reference to the accompanying drawings, in which:-

Figure 1 is a flowchart illustrating a simple instantiation of the operation of the currently preferred embodiment of the present invention.

Figure 2 is a block diagram illustrating the various repository types and the repository transaction flow between them in the currently preferred embodiment of the present invention.

Figure 3 is a block diagram of a repository coupled with a credit server in the currently preferred embodiment of the present invention.

Figures 4a and 4b are examples of rendering systems as may be utilized in the currently preferred embodiment of the present invention.

Figure 5 illustrates a contents file layout for a digital work as may be utilized in the currently preferred embodiment of the present invention.

Figure 6 illustrates a contents file layout for an individual digital work of the digital work of Figure 5 as may be utilized in the currently preferred embodiment of the present invention.

Figure 7 illustrates the components of a description block of the currently preferred embodiment of the present invention.

Figure 8 illustrates a description tree for the contents file layout of the digital work illustrated in Figure 5.

Figure 9 illustrates a portion of a description tree corresponding to the individual digital work illustrated in Figure 6.

Figure 10 illustrates a layout for the rights portion of a description block as may be utilized in the currently preferred embodiment of the present invention.

Figure 11 is a description tree wherein certain d-blocks have PRINT usage rights and is used to illustrate "strict" and "lenient" rules for resolving usage rights conflicts.

Figure 12 is a block diagram of the hardware components of a repository as are utilized in the currently preferred embodiment of the present invention.

Figure 13 is a block diagram of the functional (logical) components of a repository as are utilized in the currently preferred embodiment of the present invention.

Figure 14 is diagram illustrating the basic components of a usage right in the currently preferred embodiment of the present invention.

Figure 15 lists the usage rights grammar of the currently preferred embodiment of the present invention.

Figure 16 is a flowchart illustrating the steps of certificate delivery, hotlist checking and performance testing as performed in a registration transaction as may be performed in the currently preferred embodiment of the present invention.

Figure 17 is a flowchart illustrating the steps of session information exchange and clock synchronization as may be performed in the currently preferred embodiment of the present invention, after each repository in the registration transaction has successfully completed the steps described in Figure 16.

Figure 18 is a flowchart illustrating the basic flow for a usage transaction, including the common opening and closing step, as may be performed in the currently preferred embodiment of the present invention.

Figure 19 is a state diagram of server and client repositories in accordance with a transport protocol followed when moving a digital work from the server to the client repositories, as may be performed in the currently preferred embodiment of the present invention.

OVERVIEW

A system for controlling use and distribution of digital works is disclosed. The present invention is directed to supporting commercial transactions involving digital works.

Herein the terms "digital work", "work" and "content" refer to any work that has been reduced to a digital representation. This would include any audio, video, text, or multimedia work and any accompanying interpreter (e.g. software) that may be required for recreating the work. The term composite work refers to a digital work comprised of a collection of other digital works. The term "usage rights" or "rights" is a term which refers to rights granted to a recipient of a digital work. Generally, these rights define how a digital work can be used and if it can be further distributed. Each usage right may have one or more specified conditions which must be satisfied before the right may be exercised.

Figure 1 is a high level flowchart omitting various details but which demonstrates the basic operation of the present invention. Referring to Figure 1, a creator creates a digital work, step 101. The creator will then determine appropriate usage rights and fees, attach them to the digital work, and store them in Repository 1, step 102. The determination of appropriate usage rights and fees will depend on various economic factors. The digital work remains securely in Repository 1 until a request for access is received. The request for access begins with a session initiation by another repository. Here a Repository 2 initiates a session with Repository 1, step 103. As will be described in greater detail below, this session initiation includes steps which helps to insure that the respective repositories are trustworthy. As-

suming that a session can be established, Repository 2 may then request access to the Digital Work for a stated purpose, step 104. The purpose may be, for example, to print the digital work or to obtain a copy of the digital work. The purpose will correspond to a specific usage right. In any event, Repository 1 checks the usage rights associated with the digital work to determine if the access to the digital work may be granted, step 105. The check of the usage rights essentially involves a determination of whether a right associated with the access request has been attached to the digital work and if all conditions associated with the right are satisfied. If the access is denied, repository 1 terminates the session with an error message, step 106. If access is granted, repository 1 transmits the digital work to repository 2, step 107. Once the digital work has been transmitted to repository 2, repository 1 and 2 each generate billing information for the access which is transmitted to a credit server, step 108. Such double billing reporting is done to insure against attempts to circumvent the billing process.

Figure 2 illustrates the basic interactions between repository types in the present invention. As will become apparent from Figure 2, the various repository types will serve different functions. It is fundamental that repositories will share a core set of functionality which will enable secure and trusted communications. Referring to Figure 2, a repository 201 represents the general instance of a repository. The repository 201 has two modes of operation; a server mode and a requester mode. When in the server mode, the repository will be receiving and processing access requests to digital works. When in the requester mode, the repository will be initiating requests to access digital works. Repository 201 is general in the sense that its primary purpose is as an exchange medium for digital works. During the course of operation, the repository 201 may communicate with a plurality of other repositories, namely authorization repository 202, rendering repository 203 and master repository 204. Communication between repositories occurs utilizing a repository transaction protocol 205.

Communication with an authorization repository 202 may occur when a digital work being accessed has a condition requiring an authorization. Conceptually, an authorization is a digital certificate such that possession of the certificate is required to gain access to the digital work. An authorization is itself a digital work that can be moved between repositories and subjected to fees and usage rights conditions. An authorization may be required by both repositories involved in an access to a digital work.

Communication with a rendering repository 203 occurs in connection with the rendering of a digital work. As will be described in greater detail below, a rendering repository is coupled with a rendering device (e.g. a printer device) to comprise a rendering system.

Communication with a master repository 205 occurs in connection with obtaining an identification certificate. Identification certificates are the means by which a repository is identified as "trustworthy". The use of identification certificates is described below with respect to the registration transaction.

Figure 3 illustrates the repository 201 coupled to a credit server 301. The credit server 301 is a device which accumulates billing information for the repository 201. The credit server 301 communicates with repository 201 via billing transactions 302 to record billing transactions. Billing transactions are reported to a billing clearinghouse 303 by the credit server 301 on a periodic basis. The credit server 301 communicates to the billing clearinghouse 303 via clearinghouse transactions 304. The clearinghouse transactions 304 enable a secure and encrypted transmission of information to the billing clearinghouse 303.

RENDERING SYSTEMS

A rendering system is generally defined as a system comprising a repository and a rendering device which can render a digital work into its desired form. Examples of a rendering system may be a computer system, a digital audio system, or a printer. A rendering system has the same security features as a repository. The coupling of a rendering repository with the rendering device may occur in a manner suitable for the type of rendering device.

Figure 4a illustrates a printer as an example of a rendering system. Referring to Figure 4, printer system 401 has contained therein a printer repository 402 and a print device 403. It should be noted that the the dashed line defining printer system 401 defines a secure system boundary. Communications within the boundary are assumed to be secure. Depending on the security level, the boundary also represents a barrier intended to provide physical integrity. The printer repository 402 is an instantiation of the rendering repository 205 of Figure 2. The printer repository 402 will in some instances contain an ephemeral copy of a digital work which remains until it is printed out by the print engine 403. In other instances, the printer repository 402 may contain digital works such as fonts, which will remain and can be billed based on use. This design assures that all communication lines between printers and printing devices are encrypted, unless they are within a physically secure boundary. This design feature eliminates a potential "fault" point through which the digital work could be improperly obtained. The printer device 403 represents the printer components used to create the printed output.

Also illustrated in Figure 4a is the repository 404. The repository 404 is coupled to the printer repository 402. The repository 404 represents an external repository which contains digital works.

Figure 4b is an example of a computer system as a rendering system. A computer system may constitute a "multi-

function" device since it may execute digital works (e.g. software programs) and display digital works (e.g. a digitized photograph). Logically, each rendering device can be viewed as having its own repository, although only one physical repository is needed. Referring to Figure 4b, a computer system 410 has contained therein a display/execution repository 411. The display/execution repository 411 is coupled to display device, 412 and execution device 413. The dashed box surrounding the computer system 410 represents a security boundary within which communications are assumed to be secure. The display/execution repository 411 is further coupled to a credit server 414 to report any fees to be billed for access to a digital work and a repository 415 for accessing digital works stored therein.

STRUCTURE OF DIGITAL WORKS

Usage rights are attached directly to digital works. Thus, it is important to understand the structure of a digital work. The structure of a digital work, in particular composite digital works, may be naturally organized into an acyclic structure such as a hierarchy. For example, a magazine has various articles and photographs which may have been created and are owned by different persons. Each of the articles and photographs may represent a node in a hierarchical structure. Consequently, controls, i.e. usage rights, may be placed on each node by the creator. By enabling control and fee billing to be associated with each node, a creator of a work can be assured that the rights and fees are not circumvented.

In the currently preferred embodiment, the file information for a digital work is divided into two files: a "contents" file and a "description tree" file. From the perspective of a repository, the "contents" file is a stream of addressable bytes whose format depends completely on the interpreter used to play, display or print the digital work. The description tree file makes it possible to examine the rights and fees for a work without reference to the content of the digital work. It should be noted that the term description tree as used herein refers to any type of acyclic structure used to represent the relationship between the various components of a digital work.

Figure 5 illustrates the layout of a contents file. Referring to Figure 5, a digital work is comprised of story A 510, advertisement 511, story B 512 and story C 513. It is assumed that the digital work is stored starting at a relative address of 0. Each of the parts of the digital work are stored linearly so that story A 510 is stored at approximately addresses 0-30,000, advertisement 511 at addresses 30,001-40,000, story B 512 at addresses 40,001-60,000 and story C 513 at addresses 60,001-85K. The detail of story A 510 is illustrated in Figure 6. Referring to Figure 6, the story A 510 is further broken down to show text 614 stored at address 0-1500, soldier photo 615 at addresses 1501-10,000, graphics 616 stored at addresses 10,001-25,000 and sidebar 617 stored address 25,001-30,000. Note that the data in the contents file may be compressed (for saving storage) or encrypted (for security).

From Figures 5 and 6 it is readily observed that a digital work can be represented by its component parts as a hierarchy. The description tree for a digital work is comprised of a set of related descriptor blocks (d-blocks). The contents of each d-block is described with respect to Figure 7. Referring to Figure 7, a d-block 700 includes an identifier 701 which is a unique identifier for the work in the repository, a starting address 702 providing the start address of the first byte of the work, a length 703 giving the number of bytes in the work, a rights portion 704 wherein the granted usage rights and their status data are maintained, a parent pointer 705 for pointing to a parent d-block and child pointers 706 for pointing to the child d-blocks. In the currently preferred embodiment, the identifier 701 has two parts. The first part is a unique number assigned to the repository upon manufacture. The second part is a unique number assigned to the work upon creation. The rights portion 704 will contain a data structure, such as a look-up table, wherein the various information associated with a right is maintained. The information required by the respective usage rights is described in more detail below. D-blocks form a strict hierarchy. The top d-block of a work has no parent; all other d-blocks have one parent. The relationship of usage rights between parent and child d-blocks and how conflicts are resolved is described below.

A special type of d-block is a "shell" d-block. A shell d-block adds no new content beyond the content of its parts. A shell d-block is used to add rights and fee information, typically by distributors of digital works.

Figure 8 illustrates a description tree for the digital work of Figure 5. Referring to Figure 8, a top d-block 820 for the digital work points to the various stories and advertisements contained therein. Here, the top d-block 820 points to d-block 821 (representing story A 510), d-block 822 (representing the advertisement 511), d-block 823 (representing story B 512) and and d-block 824 (representing story C 513).

The portion of the description tree for Story A 510 is illustrated in Figure 9. D-block 925 represents text 614, d-block 926 represents photo 615, d-block 927 represents graphics 616 by and d-block 928 represents sidebar 617.

The rights portion 704 of a descriptor block is further illustrated in Figure 10. Figure 10 illustrates a structure which is repeated in the rights portion 704 for each right. Referring to Figure 10, each right will have a right code field 1050 and status information field 1052. The right code field 1050 will contain a unique code assigned to a right. The status information field 1052 will contain information relating to the state of a right and the digital work. Such information is indicated below in Table 1. The rights as stored in the rights portion 704 may typically be in numerical order based on the right code.

TABLE 1

DIGITAL WORK STATE INFORMATION		
Property	**Value**	**Use**
Copies-in-Use	Number	A counter of the number of copies of a work that are in use. Incremented when another copy is used; decremented when use is completed.
Loan-Period	Time-Units	Indicator of the maximum number of time-units that a document can be loaned out
Loaner-Copy	Boolean	Indicator that the current work is a loaned out copy of an authorized digital work.
Remaining-Time	Time-Units	Indicator of the remaining time of use on a metered document right.
Document-Descr	String	A string containing various identifying information about a document. The exact format of this is not specified, but it can include information such as a publisher name, author name, ISBN number, and so on.
Revenue-Owner	RO-Descr	A handle identifying a revenue owner for a digital work. This is used for reporting usage fees.
Publication-Date	Date-Descr	The date that the digital work was published.
History-list	History-Rec	A list of events recording the repostories and dates for operations that copy, transfer, backup, or restore a digital work.

The approach for representing digtal works by separating description data from content assumes that parts of a file are contiguous but takes no position on the actual representation of content. In particular, it is neutral to the question of whether content representation may take an object oriented approach. It would be natural to represent content as objects. In principle, it may be convenient to have content objects that include the billing structure and rights information that is represented in the d-blocks. Such variations in the design of the representation are possible and are viable alternatives but may introduce processing overhead, e.g. the interpretation of the objects.

Digital works are stored in a repository as part of a hierarchical file system. Folders (also termed directories and sub-directories) contain the digital works as well as other folders. Digital works and folders in a folder are ordered in alphabetical order. The digital works are typed to reflect how the files are used. Usage rights can be attached to folders so that the folder itself is treated as a digital work. Access to the folder would then be handled in the same fashion as any other digital work As will be described in more detail below, the contents of the folder are subject to their own rights. Moreover, file management rights may be attached to the folder which define how folder contents can be managed.

ATTACHING USAGE RIGHTS TO A DIGITAL WORK

It is fundamental to the present invention that the usage rights are treated as part of the digital work. As the digital work is distributed, the scope of the granted usage rights will remain the same or may be narrowed. For example, when a digital work is transferred from a document server to a repository, the usage rights may include the right to loan a copy for a predetermined period of time (called the original rights). When the repository loans out a copy of the digital work, the usage rights in the loaner copy (called the next set of rights) could be set to prohibit any further rights to loan out the copy. The basic idea is that one cannot grant more rights than they have.

The attachment of usage rights into a digital work may occur in a variety of ways. If the usage rights will be the same for an entire digital work, they could be attached when the digital work is processed for deposit in the digital work server. In the case of a digital work having different usage rights for the various components, this can be done as the digital work is being created. An authoring tool or digital work assembling tool could be utilized which provides for an automated process of attaching the usage rights.

As will be described below, when a digital work is copied, transferred or loaned, a "next set of rights" can be specified. The"next set of rights" will be attached to the digital work as it is transported.

Resolving Conflicting Rights

Because each part of a digital work may have its own usage rights, there will be instances where the rights of a "contained part" are different from its parent or container part. As a result, conflict rules must be established to dictate when and how a right may be exercised. The hierarchical structure of a digital work facilitates the enforcement of such

rules. A "strict" rule would be as follows: a right for a part in a digital work is sanctioned if and only if it is sanctioned for the part, for ancestor d-blocks containing the part and for all descendent d-blocks. By sanctioned, it is meant that (1) each of the respective parts must have the right, and (2) any conditions for exercising the right are satisfied.

It also possible to implement the present invention using a more lenient rule. In the more lenient rule, access to the part may be enabled to the descendent parts which have the right, but access is denied to the descendents which do not.

An example of applying both the strict rule and lenient is illustrated with reference to Figure 11. Referring to Figure 11, a root d-block 1101 has child d-blocks 1102-1105. In this case, root d-block represents a magazine, and each of the child d-blocks 1102-1105 represent articles in the magazine. Suppose that a request is made to PRINT the digital work represented by root d-block 1101 wherein the strict rule is followed. The rights for the root d-block 1101 and child d-blocks 1102-1105 are then examined. Root d-block 1101 and child d-blocks 1102 and 1105 have been granted PRINT rights. Child d-block 1103 has not been granted PRINT rights and child d-block 1104 has PRINT rights conditioned on payment of a usage fee.

Under the strict rule the PRINT right cannot be exercised because the child d-block does not have the PRINT right. Under the lenient rule, the result would be different. The digital works represented by child d-blocks 1102 and 1105 could be printed and the digital work represented by d-block 1104 could be printed so long as the usage fee is paid. Only the digital work represented by d-block 1103 could not be printed. This same result would be accomplished under the strict rule if the requests were directed to each of the individual digital works.

The present invention supports various combinations of allowing and disallowing access. Moreover, as will be described below, the usage rights grammar permits the owner of a digital work to specify if constraints may be imposed on the work by a container part. The manner in which digital works may be sanctioned because of usage rights conflicts would be implementation specific and would depend on the nature of the digital works.

REPOSITORIES

In the description of Figure 2, it was indicated that repositories come in various forms. All repositories provide a core set of services for the transmission of digital works. The manner in which digital works are exchanged is the basis for all transaction between repositories. The various repository types differ in the ultimate functions that they perform. Repositories may be devices themselves, or they may be incorporated into other systems. An example is the rendering repository 203 of Figure 2.

A repository will have associated with it a repository identifier. Typically, the repository identifier would be a unique number assigned to the repository at the time of manufacture. Each repository will also be classified as being in a particular security class. Certain communications and transactions may be conditioned on a repository being in a particular security class. The various security classes are described in greater detail below.

As a prerequisite to operation, a repository will require possession of an identification certificate. Identification certificates are encrypted to prevent forgery and are issued by a Master repository. A master repository plays the role of an authorization agent to enable repositories to receive digital works. Identification certificates must be updated on a periodic basis. Identification certificates are described in greater detail below with respect to the registration transaction.

A repository has both a hardware and functional embodiment. The functional embodiment is typically software executing on the hardware embodiment. Alternatively, the functional embodiment may be embedded in the hardware embodiment such as an Application Specific Integrated Circuit (ASIC) chip.

The hardware embodiment of a repository will be enclosed in a secure housing which if compromised, may cause the repository to be disabled. The basic components of the hardware embodiment of a repository are described with reference to Figure 12. Referring to Figure 12, a repository is comprised of a processing means 1200, storage system 1207, clock 1205 and external interface 1206. The processing means 1200 is comprised of a processor element 1201 and processor memory 1202. The processing means 1201 provides controller, repository transaction and usage rights transaction functions for the repository. Various functions in the operation of the repository such as decryption and/or decompression of digital works and transaction messages are also performed by the processing means 1200. The processor element 1201 may be a microprocessor or other suitable computing component. The processor memory 1202 would typically be further comprised of Read Only Memories (ROM) and Random Access Memories (RAM). Such memories would contain the software instructions utilized by the processor element 1201 in performing the functions of the repository.

The storage system 1207 is further comprised of descriptor storage 1203 and content storage 1204. The description tree storage 1203 will store the description tree for the digital work and the content storage will store the associated content. The description tree storage 1203 and content storage 1204 need not be of the same type of storage medium, nor are they necessarily on the same physical device. So for example, the descriptor storage 1203 may be stored on a solid state storage (for rapid retrieval of the description tree information), while the content storage 1204 may be on

a high capacity storage such as an optical disk.

The clock 1205 is used to time-stamp various time based conditions for usage rights or for metering usage fees which may be associated with the digital works. The clock 1205 will have an uninterruptable power supply, e.g. a battery, in order to maintain the integrity of the time-stamps. The external interface means 1206 provides for the signal connection to other repositories and to a credit server. The external interface means 1206 provides for the exchange of signals via such standard interfaces such as RS-232 or Personal Computer Manufacturers Card Industry Association (PCMCIA) standards, or FDDI. The external interface means 1206 may also provide network connectivity.

The functional embodiment of a repository is described with reference to Figure 13. Referring to Figure 13, the functional embodiment is comprised of an operating system 1301, core repository services 1302, usage transaction handlers 1303, repository specific functions, 1304 and a user interface 1305. The operating system 1301 is specific to the repository and would typically depend on the type of processor being used. The operating system 1301 would also provide the basic services for controlling and interfacing between the basic components of the repository.

The core repository services 1302 comprise a set of functions required by each and every repository. The core repository services 1302 include the session initiation transactions which are defined in greater detail below. This set of services also includes a generic ticket agent which is used to "punch" a digital ticket and a generic authorization server for processing authorization specifications. Digital tickets and authorizations are specific mechanisms for controlling the distribution and use of digital works and are described in more detail below. Note that coupled to the core repository services are a plurality of identification certificates 1306. The identification certificates 1306 are required to enable the use of the repository.

The usage transactions handlers 1303 comprise functionality for processing access requests to digital works and for billing fees based on access. The usage transactions supported will be different for each repository type. For example, it may not be necessary for some repositories to handle access requests for digital works.

The repository specific functionality 1304 comprises functionality that is unique to a repository. For example, the master repository has special functionality for issuing digital certificates and maintaining encryption keys. The repository specific functionality 1304 would include the user interface implementation for the repository.

Repository Security Classes

For some digital works the losses caused by any individual instance of unauthorized copying is insignificant and the chief economic concern lies in assuring the convenience of access and low-overhead billing. In such cases, simple and inexpensive handheld repositories and network-based workstations may be suitable repositories, even though the measures and guarantees of security are modest.

At the other extreme, some digital works such as a digital copy of a first run movie or a bearer bond or stock certificate would be of very high value so that it is prudent to employ caution and fairly elaborate security measures to ensure that they are not copied or forged. A repository suitable for holding such a digital work could have elaborate measures for ensuring physical integrity and for verifying authorization before use.

By arranging a universal protocol, all kinds of repositories can communicate with each other in principle. However, creators of some works will want to specify that their works will only be transferred to repositories whose level of security is high enough. For this reason, document repositories have a ranking system for classes and levels of security. The security classes in the currently preferred embodiment are described in Table 2.

TABLE 2

REPOSITORY SECURITY LEVELS	
Level	Description of Security
0	Open system. Document transmission is unencrypted. No digital certificate is required for identification. The security of the system depends mostly on user honesty, since only modest knowledge may be needed to circumvent the security measures. The repository has no provisions for preventing unauthorized programs from running and accessing or copying files. The system does not prevent the use of removable storage and does not encrypt stored files.
1	Minimal security. Like the previous class except that stored files are minimally encrypted, including ones on removable storage.
2	Basic security. Like the previous class except that special tools and knowledge are required to compromise the programming, the contents of the repository, or the state of the clock. All digital communications are encrypted. A digital certificate is provided as identification. Medium level encryption is used. Repository identification number is unforgeable.

TABLE 2 (continued)

REPOSITORY SECURITY LEVELS	
Level	Description of Security
3	General security. Like the previous class plus the requirement of special tools are needed to compromise the physical integrity of the repository and that modest encryption is used on all transmissions. Password protection is required to use the local user interface. The digital clock system cannot be reset without authorization. No works would be stored on removable storage. When executing works as programs, it runs them in their own address space and does not give them direct access to any file storage or other memory containing system code or works. They can access works only through the transmission transaction protocol.
4	Like the previous class except that high level encryption is used on all communications. Sensors are used to record attempts at physical and electronic tampering. After such tampering, the repository will not perform other transactions until it has reported such tampering to a designated server.
5	Like the previous class except that if the physical or digital attempts at tampering exceed some preset threshold that threaten the physical integrity of the repository or the integrity of digital and cryptographic barriers, then the repository will save only document description records of history but will erase or destroy any digital identifiers that could be misused if released to an unscrupulous party. It also modifies any certificates of authenticity to indicate that the physical system has been compromised. It also erases the contents of designated documents.
6	Like the previous class except that the repository will attempt wireless communication to report tampering and will employ noisy alarms.
10	This would correspond to a very high level of security. This server would maintain constant communications to remote security systems reporting transactions, sensor readings, and attempts to circumvent security.

The characterization of security levels described in Table 2 is not intended to be fixed. More important is the idea of having different security levels for different repositories. It is anticipated that new security classes and requirements will evolve according to social situations and changes in technology.

Repository User Interface

A user interface is broadly defined as the mechanism by which a user interacts with a repository in order to invoke transactions to gain access to a digital work, or exercise usage rights. As described above, a repository may be embodied in various forms. The user interface for a repository will differ depending on the particular embodiment. The user interface may be a graphical user interface having icons representing the digital works and the various transactions that may be performed. The user interface may be a generated dialog in which a user is prompted for information.

The user interface itself need not be part of the repository. As a repository may be embedded in some other device, the user interface may merely be a part of the device in which the repository is embedded. For example, the repository could be embedded in a "card" that is inserted into an available slot in a computer system. The user interface may be a combination of a display, keyboard, cursor control device and software executing on the computer system.

At a minimum, the user interface must permit a user to input information such as access requests and alpha numeric data and provide feedback as to transaction status. The user interface will then cause the repository to initiate the suitable transactions to service the request. Other facets of a particular user interface will depend on the functionality that a repository will provide.

CREDIT SERVERS

In the present invention, fees may be associated with the exercise of a right. The requirement for payment of fees is described with each version of a usage right in the usage rights language. The recording and reporting of such fees is performed by the credit server. One of the capabilities enabled by associating fees with rights is the possibility of supporting a wide range of charging models. The simplest model, used by conventional software, is that there is a single fee at the time of purchase, after which the purchaser obtains unlimited rights to use the work as often and for as long as he or she wants. Alternative models, include metered use and variable fees. A single work can have different fees for different uses. For example, viewing a photograph on a display could have different fees than making a hardcopy

or including it in a newly created work. A key to these alternative charging models is to have a low overhead means of establishing fees and accounting for credit on these transactions.

A credit server is a computational system that reliably authorizes and records these transactions so that fees are billed and paid. The credit server reports fees to a billing clearinghouse. The billing clearinghouse manages the financial transactions as they occur. As a result, bills may be generated and accounts reconciled. Preferably, the credit server would store the fee transactions and periodically communicate via a network with the billing clearinghouse for reconciliation. In such an embodiment, communications with the billing clearinghouse would be encrypted for integrity and security reasons. In another embodiment, the credit server acts as a "debit card" where transactions occur in "real-time" against a user account.

A credit server is comprised of memory, a processing means, a clock, and interface means for coupling to a repository and a financial institution (e.g. a modem). The credit server will also need to have security and authentication functionality. These elements are essentially the same elements as those of a repository. Thus, a single device can be both a repository and a credit server, provided that it has the appropriate processing elements for carrying out the corresponding functions and protocols. Typically, however, a credit server would be a card-sized system in the possession of the owner of the credit. The credit server is coupled to a repository and would interact via financial transactions as described below. Interactions with a financial institution may occur via protocols established by the financial institutions themselves.

In the currently preferred embodiment credit servers associated with both the server and the repository report the financial transaction to the billing clearinghouse. For example, when a digital work is copied by one repository to another for a fee, credit servers coupled to each of the repositories will report the transaction to the billing clearinghouse. This is desirable in that it insures that a transaction will be accounted for in the event of some break in the communication between a credit server and the billing clearinghouse. However, some implementations may embody only a single credit server reporting the transaction to minimize transaction processing at the risk of losing some transactions.

USAGE RIGHTS LANGUAGE

The present invention uses statements in a high level "usage rights language" to define rights associated with digital works and their parts. Usage rights statements are interpreted by repositories and are used to determine what transactions can be successfully carried out for a digital work and also to determine parameters for those transactions. For example, sentences in the language determine whether a given digital work can be copied, when and how it can be used, and what fees (if any) are to be charged for that use. Once the usage rights statements are generated, they are encoded in a suitable form for accessing during the processing of transactions.

Defining usage rights in terms of a language in combination with the hierarchical representation of a digital work enables the support of a wide variety of distribution and fee schemes. An example is the ability to attach multiple versions of a right to a work. So a creator may attach a PRINT right to make 5 copies for $10.00 and a PRINT right to make unlimited copies for $100.00. A purchaser may then choose which option best fits his needs. Another example is that rights and fees are additive. So in the case of a composite work, the rights and fees of each of the components works is used in determining the rights and fees for the work as a whole.

The basic contents of a right are illustrated in Figure 14. Referring to Figure 14, a right 1450 has a transactional component 1451 and a specifications component 1452. A right 1450 has a label (e.g. COPY or PRINT) which indicates the use or distribution privileges that are embodied by the right. The transactional component 1451 corresponds to a particular way in which a digital work may be used or distributed. The transactional component 1451 is typically embodied in software instructions in a repository which implement the use or distribution privileges for the right. The specifications components 1452 are used to specify conditions which must be satisfied prior to the right being exercised or to designate various transaction related parameters. In the currently preferred embodiment, these specifications include copy count 1453, Fees and Incentives 1454, Time 1455, Access and Security 1456 and Control 1457. Each of these specifications will be described in greater detail below with respect to the language grammar elements.

The usage rights language is based on the grammar described below. A grammar is a convenient means for defining valid sequence of symbols for a language. In describing the grammar the notation "[a|b|c]" is used to indicate distinct choices among alternatives. In this example, a sentence can have either an "a", "b" or "c". It must include exactly one of them. The braces {} are used to indicate optional items. Note that brackets, bars and braces are used to describe the language of usage rights sentences but do not appear in actual sentences in the language.

In contrast, parentheses are part of the usage rights language. Parentheses are used to group items together in lists. The notation (x^*) is used to indicate a variable length list, that is, a list containing one or more items of type x. The notation $(x)^*$ is used to indicate a variable number of lists containing x.

Keywords in the grammar are words followed by colons. Keywords are a common and very special case in the language. They are often used to indicate a single value, typically an identifier. In many cases, the keyword and the parameter are entirely optional. When a keyword is given, it often takes a single identifier as its value. In some cases,

the keyword takes a list of identifiers.

In the usage rights language, time is specified in an hours:minutes:seconds (or hh:mm:ss) representation. Time zone indicators, e.g. PDT for Pacific Daylight Time, may also be specified. Dates are represented as year/ month/day (or YYYY/MMM/DD). Note that these time and date representations may specify moments in time or units of time Money units are specified in terms of dollars.

Finally, in the usage rights language, various "things" will need to interact with each other. For example, an instance of a usage right may specify a bank account, a digital ticke etc.. Such things need to be identified and are specified herein using the suffix "-ID."

The Usage Rights Grammar is listed in its entirety in Figure 15 and is described below.

Grammar element 1501 **"Digital Work Rights:= (Rights*)"** define the digital wor rights as a set of rights. The set of rights attached to a digital work define how that digital work may be transferred, used, performed or played. A set of rights will attach to the entire digital work and in the case of compound digital works, each of the components of the digita work. The usage rights of components of a digital may be different.

Grammar element 1502 **"Right: = (Right-Code {Copy-Count} {Control-Spec} {Time-Spec} {Access-Spec} {Fee-Spec})"** enumerates the content of a right. Each usage right must specify a right code. Each right may also optionally specify conditions which must be satisfied before the right can be exercised. These conditions are copy count, control, time, access and fee conditions. In the currently preferred embodiment, for the optional elements, the following defaults apply: copy count equals 1, no time limit on the use of the right, no access tests or a security level required to use the right and no fee is required. These conditions will each be described in greater detail below.

It is important to note that a digital work may have multiple versions of a right, each having the same right code. The multiple version would provide alternative conditions and fees for accessing the digital work.

Grammar element 1503 **"Right-Code : = Render-Code I Transport-Code I File-Management-CodeI Derivative-Works- Code Configuration-Code"** distinguishes each of the specific rights into a particular right type (although each right is identified by distinct right codes). In this way, the grammar provides a catalog of possible rights that can be associated with parts of digital works. In the following, rights are divided into categories for convenience in describing them.

Grammar element 1504 **"Render-Code : = [Play:{Player:Player-ID}I Print: {Printer: Printer-ID}]"** lists a category of rights all involving the making of ephemeral, transitory, or non-digital copies of the digital work. After use the copies are erased.

- Play A process of rendering or performing a digital work on some processor. This includes such things as playing digital movies, playing digital music, playing a video game, running a computer program, or displaying a document on a display.
- Print To render the work in a medium that is not further protected by usage rights, such as printing on paper.

Grammar element 1505 **"Transport-Code : = [Copy I Transfer I Loan (Remaining-Rights: Next-Set-of-Rights)] {(Next-Copy-Rights: Next-Set of Rights)}"** lists a category of rights involving the making of persistent, usable copies of the digital work on other repositories. The optional Next-Copy-Rights determine the rights on the work after it is transported. If this is not specified, then the rights on the transported copy are the same as on the original. The optional Remaining-Rights specify the rights that remain with a digital work when it is loaned out. If this is not specified, then the default is that no rights can be exercised when it is loaned out.

- Copy Make a new copy of a work
- Transfer Moving a work from one repository to another.
- Loan Temporarily loaning a copy to another repository for a specified period of time.

Grammar element 1506 **"File-Management-Code: = Backup {Back-Up-Copy-Rights: Next-Set -of Rights}I Restore I Delete I Folder I Directory {Name:Hide-Local I Hide - Remote}{Parts:Hide-Local I Hide-Remote}"** lists a category of rights involving operations for file management, such as the making of backup copies to protect the copy owner against catastrophic equipment failure.

Many software licenses and also copyright law give a copy owner the right to make backup copies to protect against catastrophic failure of equipment. However, the making of uncontrolled backup copies is inherently at odds with the ability to control usage, since an uncontrolled backup copy can be kept and then restored even after the authorized copy was sold.

The File management rights enable the making and restoring of backup copies in a way that respects usage rights, honoring the requirements of both the copy owner and the rights grantor and revenue owner. Backup copies of work descriptions (including usage rights and fee data) can be sent under appropriate protocol and usage rights control to other document repositories of sufficiently high security. Further rights permit organization of digital works into folders

which themselves are treated as digital works and whose contents may be "hidden" from a party seeking to determine the contents of a repository.

- Backup To make a backup copy of a digital work as protection against media failure.
- Restore To restore a backup copy of a digital work.
- Delete To delete or erase a copy of a digital work.
- Folder To create and name folders, and to move files and folders between folders.
- Directory To hide a folder or its contents.

Grammar element 1507 **"Derivative-Works-Code: [Extract I Embed I Edit {Process: Process-ID}] {Next-Copy-Rights : Next-Set-of Rights}"** lists a category of rights involving the use of a digital work to create new works.

- Extract To remove a portion of a work, for the purposes of creating a new work.
- Embed To include a work in an existing work.
- Edit To alter a digital work by copying, selecting and modifying portions of an existing digital work.

Grammar element 1508 **"Configuration-Code: = Install I Uninstall"** lists a category of rights for installing and uninstalling software on a repository (typically a rendering repository.) This would typically occur for the installation of a new type of player within the rendering repository.

- Install: To install new software on a repository.
- Uninstall: To remove existing software from a repository.

Grammar element 1509 **"Next-Set-of-Rights: = {(Add: Set-Of-Rights)} {(Delete: Set-Of-Rights)} {(Replace: Set-Of-Rights)} {(Keep: Set-Of-Rights)}"** defines how rights are carried forward for a copy of a digital work. If the Next-Copy-Rights is not specified, the rights for the next copy are the same as those of the current copy. Otherwise, the set of rights for the next copy can be specified. Versions of rights after Add: are added to the current set of rights. Rights after Delete: are deleted from the current set of rights. If only right codes are listed after Delete:, then all versions of rights with those codes are deleted. Versions of rights after Replace: subsume all versions of rights of the same type in the current set of rights.

If Remaining-Rights is not specified, then there are no rights for the original after all Loan copies are loaned out. If Remaining-Rights is specified, then the Keep: token can be used to simplify the expression of what rights to keep behind. A list of right codes following keep means that all of the versions of those listed rights are kept in the remaining copy. This specification can be overridden by subsequent Delete: or Replace: specifications.

Copy Count Specification

For various transactions, it may be desirable to provide some limit as to the number of "copies" of the work which may be exercised simultaneously for the right. For example, it may be desirable to limit the number of copies of a digital work that may be loaned out at a time or viewed at a time.

Grammar element 1510 **"Copy-Count : = (Copies: positive-integer I 0 I unlimited)"** provides a condition which defines the number of "copies" of a work subject to the right . A copy count can be 0, a fixed number, or unlimited. The copy-count is associated with each right, as opposed to there being just a single copy-count for the digital work. The Copy-Count for a right is decremented each time that a right is exercised. When the Copy-Count equals zero, the right can no longer be exercised. If the Copy-Count is not specified, the default is one.

Control Specification

Rights and fees depend in general on rights granted by the creator as well as further restrictions imposed by later distributors. Control specifications deal with interactions between the creators and their distributors governing the imposition of further restrictions and fees. For example, a distributor of a digital work may not want an end consumer of a digital work to add fees or otherwise profit by commercially exploiting the purchased digital work.

Grammar element 1511 **"Control-Spec : = (Control: {Restrictable I Unrestrictable} {Unchargeable I Chargeable})"** provides a condition to specify the effect of usage rights and fees of parents on the exercise of the right. A digital work is restrictable if higher level d-blocks can impose further restrictions (time specifications and access specifications) on the right. It is unrestrictable if no further restrictions can be imposed. The default setting is restrictable. A right is unchargeable if no more fees can be imposed on the use of the right. It is chargeable if more fees can be imposed. The default is chargeable.

Time Specification

It is often desirable to assign a start date or specify some duration as to when a right may be exercised. Grammar element 1512 **"Time-Spec : = ({Fixed-Interval | Sliding-Interval | Meter-Time} Until: Expiration-Date)"** provides for specification of time conditions on the exercise of a right. Rights may be granted for a specified time. Different kinds of time specifications are appropriate for different kinds of rights. Some rights may be exercised during a fixed and predetermined duration. Some rights may be exercised for an interval that starts the first time that the right is invoked by some transaction. Some rights may be exercised or are charged according to some kind of metered time, which may be split into separate intervals. For example, a right to view a picture for an hour might be split into six ten minute viewings or four fifteen minute viewings or twenty three minute viewings.

The terms "time" and "date" are used synonymously to refer to a moment in time. There are several kinds of time specifications. Each specification represents some limitation on the times over which the usage right applies. The Expiration-Date specifies the moment at which the usage right ends. For example, if the Expiration-Date is "Jan 1, 1995," then the right ends at the first moment of 1995. If the Expiration-Date is specified as *forever*, then the rights are interpreted as continuing without end. If only an expiration date is given, then the right can be exercised as often as desired until the expiration date.

Grammar element 1513 **"Fixed-Interval : = From: Start-Time"** is used to define a predetermined interval that runs from the start time to the expiration date.

Grammar element 1514 **"Sliding-Interval : = Interval: Use-Duration"** is used to define an indeterminate (or "open") start time. It sets limits on a continuous period of time over which the contents are accessible. The period starts on the first access and ends after the duration has passed or the expiration date is reached, whichever comes first. For example, if the right gives 10 hours of continuous access, the use-duration would begin when the first access was made and end 10 hours later.

Grammar element 1515 **"Meter-Time: = Time-Remaining: Remaining-Use"** is used to define a "meter time," that is, a measure of the time that the right is actually exercised. It differs from the Sliding-Interval specification in that the time that the digital work is in use need not be continuous. For example, if the rights guarantee three days of access, those days could be spread out over a month. With this specification, the rights can be exercised until the meter time is exhausted or the expiration date is reached, whichever comes first.

Remaining-Use: = Time-Unit
Start-Time: = Time-Unit
Use-Duration: = Time-Unit
All of the time specifications include time-unit specifications in their ultimate instantiation.

Security Class and Authorization Specification

The present invention provides for various security mechanisms to be introduced into a distribution or use scheme. Grammar element 1516 **"Access-Spec : ({SC: Security-Class} {Authorization: Authorization-ID*} {Other-Authorization: Authorization-ID*} {Ticket: Ticket-ID})"** provides a means for restricting access and transmission. Access specifications can specify a required security class for a repository to exercise a right or a required authorization test that must be satisfied.

The keyword **"SC:"** is used to specify a minimum security level for the repositories involved in the access. If **"SC:"** is not specified, the lowest security level is acceptable.

The optional **"Authorization:"** keyword is used to specify required authorizations on the same repository as the work. The optional **"Other-Authorization:"** keyword is used to specify required authorizations on the other repository in the transaction.

The optional **"Ticket:"** keyword specifies the identity of a ticket required for the transaction. A transaction involving digital tickets must locate an appropriate digital ticket agent who can "punch" or otherwise validate the ticket before the transaction can proceed. Tickets are described in greater detail below.

In a transaction involving a repository and a document server, some usage rights may require that the repository have a particular authorization, that the server have some authorization, or that both repositories have (possibly different) authorizations. Authorizations themselves are digital works (hereinafter referred to as an authorization object) that can be moved between repositories in the same manner as other digital works. Their copying and transferring is subject to the same rights and fees as other digital works. A repository is said to have an authorization if that authorization object is contained within the repository.

In some cases, an authorization may be required from a source other than the document server and repository. An authorization object referenced by an Authorization-ID can contain digital address information to be used to set up a communications link between a repository and the authorization source. These are analogous to phone numbers.

For such access tests, the communication would need to be established and authorization obtained before the right could be exercised.

For one-time usage rights, a variant on this scheme is to have a digital ticket. A ticket is presented to a digital ticket agent, whose type is specified on the ticket. In the simplest case, a certified generic ticket agent, available on all repositories, is available to "punch" the ticket. In other cases, the ticket may contain addressing information for locating a "special" ticket agent. Once a ticket has been punched, it cannot be used again for the same kind of transaction (unless it is unpunched or refreshed in the manner described below.) Punching includes marking the ticket with a timestamp of the date and time it was used. Tickets are digital works and can be copied or transferred between repositories according to their usage rights.

In the currently preferred embodiment, a "punched" ticket becomes "unpunched" or "refreshed" when it is copied or extracted. The Copy and Extract operations save the date and time as a property of the digital ticket. When a ticket agent is given a ticket, it can simply check whether the digital copy was made after the last time that it was punched. Of course, the digital ticket must have the copy or extract usage rights attached thereto.

The capability to unpunch a ticket is inportant in the following cases:

- A digital work is circulated at low cost with a limitation that it can be used only once.
- A digital work is circulated with a ticket that can be used once to give discounts on purchases of other works.
- A digital work is circulated with a ticket (included in the purchase price and possibly embedded in the work) that can be used for a future upgrade.

In each of these cases, if a paid copy is made of the digital work (including the ticket) the new owner would expect to get a fresh (unpunched) ticket, whether the copy seller has used the work or not. In contrast, loaning a work or simply transferring it to another repository should not revitalize the ticket.

Usage Fees and Incentives Specification

The billing for use of a digital work is fundamental to a commercial distribution system. Grammar Element 1517 **"Fee-Spec: = {Scheduled-Discount} Regular-Fee-Spec I Scheduled-Fee-Spec I Markup-Spec"** provides a range of options for billing for the use of digital works.

A key feature of this approach is the development of low-overhead billing for transactions in potentially small amounts. Thus, it becomes feasible to collect fees of only a few cents each for thousands of transactions.

The grammar differentiates between uses where the charge is per use from those where it is metered by the time unit. Transactions can support fees that the user pays for using a digital work as well as incentives paid by the right grantor to users to induce them to use or distribute the digital work.

The optional scheduled discount refers to the rest of the fee specification--discounting it by a percentage over time. If it is not specified, then there is no scheduled discount. Regular fee specifications are constant over time. Scheduled fee specifications give a schedule of dates over which the fee specifications change. Markup specifications are used in d-blocks for adding a percentage to the fees already being charged.

Grammar Element 1518 **"Scheduled-Discount: = (Scheduled-Discount: (Time-Spec Percentage)*)"** A Scheduled-Discount is a essentially a scheduled modifier of any other fee specification for this version of the right of the digital work. (It does not refer to children or parent digital works or to other versions of rights.). It is a list of pairs of times and percentages. The most recent time in the list that has not yet passed at the time of the transaction is the one in effect. The percentage gives the discount percentage. For example, the number 10 refers to a 10% discount.

Grammar Element 1519 **"Regular-Fee-Spec : = ({Fee: I Incentive:} [Per-Use-Spec I Metered-Rate-Spec I Best-Price-Spec I Call-For-Price-Spec] {Min: Money-Unit Per: Time-Spec}{Max: Money-Unit Per: Time-Spec} To: Account-ID)"** provides for several kinds of fee specifications.

Fees are paid by the copy-owner/user to the revenue-owner if **Fee:** is specified. Incentives are paid by the revenue-owner to the user if **Incentive:** is specified. If the **Min:** specification is given, then there is a minimum fee to be charged per time-spec unit for its use. If the **Max:** specification is given, then there is a maximum fee to be charged per time-spec for its use. When **Fee:** is specified, **Account-ID** identifies the account to which the fee is to be paid. When **Incentive:** is specified, **Account-ID** identifies the account from which the fee is to be paid.

Grammar element 1520 **"Per-Use-Spec: = Per-Use: Money-unit"** defines a simple fee to be paid every time the right is exercised, regardless of how much time the transaction takes.

Grammar element 1521 **"Metered-Rate-Spec : = Metered: Money-Unit Per: Time-Spec"** defines a metered-rate fee paid according to how long the right is exercised. Thus, the time it takes to complete the transaction determines the fee.

Grammar element 1522 **"Best-Price-Spec : = Best-Price: Money-unit Max: Money-unit"** is used to specify a best-price that is determined when the account is settled. This specification is to accommodate special deals, rebates,

and pricing that depends on information that is not available to the repository. All fee specifications can be combined with tickets or authorizations that could indicate that the consumer is a wholesaler or that he is a preferred customer, or that the seller be authorized in some way. The amount of money in the **Max:** field is the maximum amount that the use will cost. This is the amount that is tentatively debited from the credit server. However, when the transaction is ultimately reconciled, any excess amount will be returned to the consumer in a separate transaction.

Grammar element 1523 **"Call-For-Price-Spec: = Call-For-Price "** is similar to a **"Best-Price-Spec"** in that it is intended to accommodate cases where prices are dynamic. A **Call-For-Price Spec** requires a communication with a dealer to determine the price. This option cannot be exercised if the repository cannot communicate with a dealer at the time that the right is exercised. It is based on a secure transaction whereby the dealer names a price to exercise the right and passes along a deal certificate which is referenced or included in the billing process.

Grammar element 1524 **"Scheduled-Fee-Spec: = (Schedule: (Time-Spec Regular-Fee-Spec)*)"** is used to provide a schedule of dates over which the fee specifications change. The fee specification with the most recent date not in the future is the one that is in effect. This is similar to but more general than the scheduled discount. It is more general, because it provides a means to vary the fee agreement for each time period.

Grammar element 1525 **"Markup-Spec: = Markup: percentage To: Account-ID"** is provided for adding a percentage to the fees already being charged. For example, a 5% markup means that a fee of 5% of cumulative fee so far will be allocated to the distributor. A markup specification can be applied to all of the other kinds of fee specifications. It is typically used in a shell provided by a distributor. It refers to fees associated with d-blocks that are parts of the current d-block. This might be a convenient specification for use in taxes, or in distributor overhead.

REPOSITORY TRANSACTIONS

When a user requests access to a digital work, the repository will initiate various transactions. The combination of transactions invoked will depend on the specifications assigned for a usage right. There are three basic types of transactions, Session Initiation Transactions, Financial Transactions and Usage Transactions. Generally, session initiation transactions are initiated first to establish a valid session. When a valid session is established, transactions corresponding to the various usage rights are invoked. Finally, request specific transactions are performed.

Transactions occur between two repositories (one acting as a server), between a repository and a document playback platform (e.g. for executing or viewing), between a repository and a credit server or between a repository and an authorization server. When transactions occur between more than one repository, it is assumed that there is a reliable communication channel between the repositories. For example, this could be a TCP/IP channel or any other commercially available channel that has built-in capabilities for detecting and correcting transmission errors. However, it is not assumed that the communication channel is secure. Provisions for security and privacy are part of the requirements for specifying and implementing repositories and thus form the need for various transactions.

Message Transmission

Transactions require that there be some communication between repositories. Communication between repositories occurs in units termed as messages. Because the communication line is assumed to be unsecure, all communications with repositories that are above the lowest security class are encrypted utilizing a public key encryption technique. Public key encryption is a well known technique in the encryption arts. The term key refers to a numeric code that is used with encryption and decryption algorithms. Keys come in pairs, where "writing keys" are used to encrypt data and "checking keys" are used to decrypt data. Both writing and checking keys may be public or private. Public keys are those that are distributed to others. Private keys are maintained in confidence.

Key management and security is instrumental in the success of a public key encryption system. In the currently preferred embodiment, one or more master repositories maintain the keys and create the identification certificates used by the repositories.

When a sending repository transmits a message to a receiving repository, the sending repository encrypts all of its data using the public writing key of the receiving repository. The sending repository includes its name, the name of the receiving repository, a session identifier such as a nonce (described below), and a message counter in each message.

In this way, the communication can only be read (to a high probability) by the receiving repository, which holds the private checking key for decryption. The auxiliary data is used to guard against various replay attacks to security. If messages ever arrive with the wrong counter or an old nonce, the repositories can assume that someone is interfering with communication and the transaction terminated.

The respective public keys for the repositories to be used for encryption are obtained in the registration transaction described below.

Session Initiation Transactions

A usage transaction is carried out in a session between repositories. For usage transactions involving more than one repository, or for financial transactions between a repository and a credit server, a registration transaction is performed. A second transaction termed a login transaction, may also be needed to initiate the session. The goal of the registration transaction is to establish a secure channel between two repositories who know each others identities. As it is assumed that the communication channel between the repositories is reliable but not secure, there is a risk that a non-repository may mimic the protocol in order to gain illegitimate access to a repository.

The registration transaction between two repositories is described with respect to Figures 16 and 17. The steps described are from the perspective of a "repository-1" registering its identity with a "repository-2". The registration must be symmetrical so the same set of steps will be repeated for repository-2 registering its identity with repository-1. Referring to Figure 16, repository-1 first generates an encrypted registration identifier, step 1601 and then generates a registration message, step 1602. A registration message is comprised of an identifier of a master repository, the identification certificate for the repository-1 and an encrypted random registration identifier. The identification certificate is encrypted by the master repository in its private key and attests to the fact that the repository (here repository-1) is a bona fide repository. The identification certificate also contains a public key for the repository, the repository security level and a timestamp (indicating a time after which the certificate is no longer valid.) The registration identifier is a number generated by the repository for this registration. The registration identifier is unique to the session and is encrypted in repository-1's private key. The registration identifier is used to improve security of authentication by detecting certain kinds of communications based attacks. Repository-1 then transmits the registration message to repository-2, step 1603.

Upon receiving the registration message, repository-2 determines if it has the needed public key for the master repository, step 1604. If repository-2 does not have the needed public key to decrypt the identification certificate, the registration transaction terminates in an error, step 1618.

Assuming that repository-2 has the proper public key the identification certificate is decrypted, step 1605. Repository-2 saves the encrypted registration identifier, step 1606, and extracts the repository identifier, step 1607. The extracted repository identifier is checked against a "hotlist" of compromised document repositories, step 1608. In the currently preferred embodiment, each repository will contain "hotlists" of compromised repositories. If the repository is on the "hotlist', the registration transaction terminates in an error per step 1618. Repositories can be removed from the hotlist when their certificates expire, so that the list does not need to grow without bound. Also, by keeping a short list of hotlist certificates that it has previously received, a repository can avoid the work of actually going through the list. These lists would be encrypted by a master repository. A minor variation on the approach to improve efficiency would have the repositories first exchange lists of names of hotlist certificates, ultimately exchanging only those lists that they had not previously received. The "hotlists" are maintained and distributed by Master repositories.

Note that rather than terminating in error, the transaction could request that another registration message be sent based on an identification certificate created by another master repository. This may be repeated until a satisfactory identification certificate is found, or it is determined that trust cannot be established.

Assuming that the repository is not on the hotlist, the repository identification needs to be verified. In other words, repository-2 needs to validate that the repository on the other end is really repository-1. This is termed performance testing and is performed in order to avoid invalid access to the repository via a counterfeit repository replaying a recording of a prior session initiation between repository-1 and repository-2. Performance testing is initiated by repository-2 generating a performance message, step 1609. The performance message consists of a nonce, the names of the respective repositories, the time and the registration identifier received from repository-1. A nonce is a generated message based on some random and variable information (e.g. the time or the temperature.) The nonce is used to check whether repository-1 can actually exhibit correct encrypting of a message using the private keys it claims to have, on a message that it has never seen before. The performance message is encrypted using the public key specified in the registration message of repository-1. The performance message is transmitted to repository-1, step 1610, where it is decrypted by repository-1 using its private key, step 1611. Repository-1 then checks to make sure that the names of the two repositories are correct, step 1612, that the time is accurate, step 1613 and that the registration identifier corresponds to the one it sent, step 1614. If any of these tests fails, the transaction is terminated per step 1616. Assuming that the tests are passed, repository-1 transmits the nonce to repository-2 in the clear, step 1615. Repository-2 then compares the received nonce to the original nonce, step 1617. If they are not identical, the registration transaction terminates in an error per step 1618. If they are the same, the registration transaction has successfully completed.

At this point, assuming that the transaction has not terminated, the repositories exchange messages containing session keys to be used in all communications during the session and synchronize their clocks. Figure 17 illustrates the session information exchange and clock synchronization steps (again from the perspective of repository-1.) Referring to Figure 17, repository-1 creates a session key pair, step 1701. A first key is kept private and is used by repository-1 to encrypt messages. The second key is a public key used by repository-2 to decrypt messages. The

second key is encrypted using the public key of repository-2, step 1702 and is sent to repository-2, step 1703. Upon receipt, repository-2 decrypts the second key, step 1704. The second key is used to decrypt messages in subsequent communications. When each repository has completed this step, they are both convinced that the other repository is bona fide and that they are communicating with the original. Each repository has given the other a key to be used in decrypting further communications during the session. Since that key is itself transmitted in the public key of the receiving repository only it will be able to decrypt the key which is used to decrypt subsequent messages.

After the session information is exchanged, the repositories must synchronize their clocks. Clock synchronization is used by the repositories to establish an agreed upon time base for the financial records of their mutual transactions. Referring back to Figure 17, repository-2 initiates clock synchronization by generating a time stamp exchange message, step 1705, and transmits it to repository- 1, step 1706. Upon receipt, repository-1 generates its own time stamp message, step 1707 and transmits it back to repository-2, step 1708. Repository-2 notes the current time, step 1709 and stores the time received from repository-1, step 1710. The current time is compared to the time received from repository-1, step 1711. The difference is then checked to see if it exceeds a predetermined tolerance (e.g. one minute), step 1712. If it does, repository-2 terminates the transaction as this may indicate tampering with the repository, step 1713. If not repository-2 computes an adjusted time delta, step 1714. The adjusted time delta is the difference between the clock time of repository-2 and the average of the times from repository-1 and repository-2.

To achieve greater accuracy, repository-2 can request the time again up to a fixed number of times (e.g. five times), repeat the clock synchronization steps, and average the results.

A second session initiation transaction is a Login transaction. The Login transaction is used to check the authenticity of a user requesting a transaction. A Login transaction is particularly prudent for the authorization of financial transactions that will be charged to a credit server. The Login transaction involves an interaction between the user at a user interface and the credit server associated with a repository. The information exchanged here is a login string supplied by the repository/credit server to identify itself to the user, and a Personal Identification Number (PIN) provided by the user to identify himself to the credit server. In the event that the user is accessing a credit server on a repository different from the one on which the user interface resides, exchange of the information would be encrypted using the public and private keys of the respective repositories.

Billing Transactions

Billing Transactions are concerned with monetary transactions with a credit server. Billing Transactions are carried out when all other conditions are satisfied and a usage fee is required for granting the request. For the most part, billing transactions are well understood in the state of the art. These transactions are between a repository and a credit server, or between a credit server and a billing clearinghouse. Briefly, the required transactions include the following:

- Registration and LOGIN transactions by which the repository and user establish their bona fides to a credit server. These transactions would be entirely internal in cases where the repository and credit server are implemented as a single system.
- Registration and LOGIN transactions, by which a credit server establishes its bona fides to a billing clearinghouse.
- An Assign-fee transaction to assign a charge. The information in this transaction would include a transaction identifier, the identities of the repositories in the transaction, and a list of charges from the parts of the digital work. If there has been any unusual event in the transaction such as an interruption of communications, that information is included as well.
- A Begin-charges transaction to assign a charge. This transaction is much the same as an assign-fee transaction except that it is used for metered use. It includes the same information as the assign-fee transaction as well as the usage fee information. The credit-server is then responsible for running a clock.
- An End-charges transaction to end a charge for metered use. (In a variation on this approach, the repositories would exchange periodic charge information for each block of time.)
- A report-charges transaction between a personal credit server and a billing clearinghouse. This transaction is invoked at least once per billing period. It is used to pass along information about charges. On debit and credit cards, this transaction would also be used to update balance information and credit limits as needed.

All billing transactions are given a transaction ID and are reported to the credit severs by both the server and the client. This reduces possible loss of billing information if one of the parties to a transaction loses a banking card and provides a check against tampering with the system.

Usage Transactions

After the session initiation transactions have been completed, the usage request may then be processed. To sim-

plify the description of the steps carried out in processing a usage request, the term requester is used to refer to a repository in the requester mode which is initiating a request, and the term server is used to refer to a repository in the server mode and which contains the desired digital work. In many cases such as requests to print or view a work, the requester and server may be the same device and the transactions described in the following would be entirely internal. In such instances, certain transaction steps, such as the registration transaction, need not be performed.

There are some common steps that are part of the semantics of all of the usage rights transactions. These steps are referred to as the common transaction steps. There are two sets -- the "opening" steps and the "closing" steps. For simplicity, these are listed here rather than repeating them in the descriptions of all of the usage rights transactions.

Transactions can refer to a part of a digital work, a complete digital work, or a Digital work containing other digital works. Although not described in detail herein, a transaction may even refer to a folder comprised of a plurality of digital works. The term "work" is used to refer to what ever portion or set of digital works is being accessed.

Many of the steps here involve determining if certain conditions are satisfied. Recall that each usage right may have one or more conditions which must be satisfied before the right can be exercised. Digital works have parts and parts have parts. Different parts can have different rights and fees. Thus, it is necessary to verify that the requirements are met for ALL of the parts that are involved in a transaction For brevity, when reference is made to checking whether the rights exist and conditions for exercising are satisfied, it is meant that all such checking takes place for each of the relevant parts of the work.

Figure 18 illustrates the initial common opening and closing steps for a transaction. At this point it is assumed that registration has occurred and that a "trusted" session is in place. General tests are tests on usage rights associated with the folder containing the work or some containing folder higher in the file system hierarchy. These tests correspond to requirements imposed on the work as a consequence of its being on the particular repository, as opposed to being attached to the work itself. Referring to Figure 18, prior to initiating a usage transaction, the requester performs any general tests that are required before the right associated with the transaction can be exercised, step, 1801. For example, install, uninstall and delete rights may be implemented to require that a requester have an authorization certificate before the right can be exercised. Another example is the requirement that a digital ticket be present and punched before a digital work may be copied to a requester. If any of the general tests fail, the transaction is not initiated, step, 1802. Assuming that such required tests are passed, upon receiving the usage request, the server generates a transaction identifier that is used in records or reports of the transaction, step 1803. The server then checks whether the digital work has been granted the right corresponding to the requested transaction, step 1804. If the digital work has not been granted the right corresponding to the request, the transaction terminates, step 1805. If the digital work has been granted the requested right, the server then determines if the various conditions for exercising the right are satisfied. Time based conditions are examined, step 1806. These conditions are checked by examining the time specification for the the version of the right. If any of the conditions are not satisfied, the transaction terminates per step 1805.

Assuming that the time based conditions are satisfied, the server checks security and access conditions, step 1807. Such security and access conditions are satisfied if: 1) the requester is at the specified security class, or a higher security class, 2) the server satisfies any specified authorization test and 3) the requester satisfies any specified authorization tests and has any required digital tickets. If any of the conditions are not satisfied, the transaction terminates per step 1805.

Assuming that the security and access conditions are all satisfied, the server checks the copy count condition, step 1808. If the copy count equals zero, then the transaction cannot be completed and the transaction terminates per step 1805.

Assuming that the copy count does not equal zero, the server checks if the copies in use for the requested right is greater than or equal to any copy count for the requested right (or relevant parts), step 1809. If the copies in use is greater than or equal to the copy count, this indicates that usage rights for the version of the transaction have been exhausted. Accordingly, the server terminates the transaction, step 1805. If the copy count is less than the copies in use for the transaction the transaction can continue, and the copies in use would be incremented by the number of digital works requested in the transaction, step 1810.

The server then checks if the digital work has a "Loan" access right, step 1811. The "Loan" access right is a special case since remaining rights may be present even though all copies are loaned out. If the digital work has the "Loan" access right, a check is made to see if all copies have been loaned out, step 1812. The number of copies that could be loaned is the sum of the Copy-Counts for all of the versions of the loan right of the digital work. For a composite work, the relevant figure is the minimal such sum of each of the components of the composite work. If all copies have been loaned out, the remaining rights are determined, step 1813. The remaining-rights is determined from the remaining rights specifications from the versions of the Loan right. If there is only one version of the Loan right, then the determination is simple. The remaining rights are the ones specified in that version of the Loan right, or none if Remaining-Rights: is not specified. If there are multiple versions of the Loan right and all copies of all of the versions are loaned out, then the remaining rights is taken as the minimum set (intersection) of remaining rights across all of the versions of the loan right. The server then determines if the requested right is in the set of remaining rights, step 1814. If the

requested right is not in the set of remaining rights, the server terminates the transaction, step 1805.

If Loan is not a usage right for the digital work or if all copies have not been loaned out or the requested right is in the set of remaining rights, fee conditions for the right are then checked, step 1815. This will initiate various financial transactions between the repository and associated credit server. Further, any metering of usage of a digital work will commence. If any financial transaction fails, the transaction terminates per step 1805.

It should be noted that the order in which the conditions are checked need not follow the order of steps 1806-1815.

At this point, right specific steps are now performed and are represented here as step 1816. The right specific steps are described in greater detail below.

The common closing transaction steps are now performed. Each of the closing transaction steps are performed by the server after a successful completion of a transaction. Referring back to Figure 18, the copies in use value for the requested right is decremented by the number of copies involved in the transaction, step 1817. Next, if the right had a metered usage fee specification, the server subtracts the elapsed time from the Remaining-Use-Time associated with the right for every part involved in the transaction, step 1818. Finally, if there are fee specifications associated with the right, the server initiates End-Charge financial transaction to confirm billing, step 1819.

Transmission Protocol

An important area to consider is the transmission of the digital work from the server to the requester. The transmission protocol described herein refers to events occurring after a valid session has been created. The transmission protocol must handle the case of disruption in the communications between the repositories. It is assumed that interference such as injecting noise on the communication channel can be detected by the integrity checks (e.g., parity, checksum, etc.) that are built into the transport protocol and are not discussed in detail herein.

The underlying goal in the transmission protocol is to preclude certain failure modes, such as malicious or accidental interference on the communications channel. Suppose, for example, that a user pulls a card with the credit server at a specific time near the end of a transaction. There should not be a vulnerable time at which "pulling the card" causes the repositories to fail to correctly account for the number of copies of the work that have been created. Restated, there should be no time at which a party can break a connection as a means to avoid payment after using a digital work.

If a transaction is interrupted (and fails), both repositories restore the digital works and accounts to their state prior to the failure, modulo records of the failure itself.

Figure 19 is a state diagram showing steps in the process of transmitting information during a transaction. Each box represents a state of a repository in either the server mode (above the central dotted line 1901) or in the requester mode (below the dotted line 1901). Solid arrows stand for transitions between states. Dashed arrows stand for message communications between the repositories. A dashed message arrow pointing to a solid transition arrow is interpreted as meaning that the transition takes place when the message is received. Unlabeled transition arrows take place unconditionally. Other labels on state transition arrows describe conditions that trigger the transition.

Referring now to Figure 19, the server is initially in a state 1902 where a new transaction is initiated via start message 1903. This message includes transaction information including a transaction identifier and a count of the blocks of data to be transferred. The requester, initially in a wait state 1904 then enters a data wait state 1905.

The server enters a data transmit state 1906 and transmits a block of data 1907 and then enters a wait for acknowledgement state 1908. As the data is received, the requester enters a data receive state 1909 and when the data blocks are completely received it enters an acknowledgement state 1910 and transmits an Acknowledgement message 1911 to the server.

If there are more blocks to send, the server waits until receiving an Acknowledgement message from the requester. When an Acknowledgement message is received it sends the next block to the requester and again waits for acknowledgement. The requester also repeats the same cycle of states.

If the server detects a communications failure before sending the last block, it enters a cancellation state 1912 wherein the transaction is cancelled. Similarly, if the requester detects a communications failure before receiving the last block it enters a cancellation state 1913.

If there are no more blocks to send, the server commits to the transaction and waits for the final Acknowledgement in state 1914. If there is a communications failure before the server receives the final Acknowledgement message, it still commits to the transaction but includes a report about the event to its credit server in state 1915. This report serves two purposes. It will help legitimize any claims by a user of having been billed for receiving digital works that were not completely received. Also it helps to identify repositories and communications lines that have suspicious patterns of use and interruption. The server then enters its completion state 1916.

On the requester side, when there are no more blocks to receive, the requester commits to the transaction in state 1917. If the requester detects a communications failure at this state, it reports the failure to its credit server in state 1918, but still commits to the transaction. When it has committed, it sends an acknowledgement message to the server. The server then enters its completion state 1919.

The key property is that both the server and the requester cancel a transaction if it is interrupted before all of the data blocks are delivered, and commits to it if all of the data blocks have been delivered.

There is a possibility that the server will have sent all of the data blocks (and committed) but the requester will not have received all of them and will cancel the transaction. In this case, both repositories will presumably detect a communications failure and report it to their credit server. This case will probably be rare since it depends on very precise timing of the communications failure. The only consequence will be that the user at the requester repository may want to request a refund from the credit services -- and the case for that refund will be documented by reports by both repositories.

To prevent loss of data, the server should not delete any transferred digital work until receiving the final acknowledgement from the requester. But it also should not use the file. A well known way to deal with this situation is called "two-phase commit" or 2PC.

Two-phase commit works as follows. The first phase works the same as the method described above. The server sends all of the data to the requester. Both repositories mark the transaction (and appropriate files) as uncommitted. The server sends a ready-to-commit message to the requester. The requester sends back an acknowledgement. The server then commits and sends the requester a commit message. When the requester receives the commit message, it commits the file.

If there is a communication failure or other crash, the requester must check back with the server to determine the status of the transaction. The server has the last word on this. The requester may have received all of the data, but if it did not get the final message, it has not committed. The server can go ahead and delete files (except for transaction records) once it commits, since the files are known to have been fully transmitted before starting the 2PC cycle.

There are variations known in the art which can be used to achieve the same effect. For example, the server could use an additional level of encryption when transmitting a work to a client. Only after the client sends a message acknowledging receipt does it send the key. The client then agrees to pay for the digital work. The point of this variation is that it provides a clear audit trail that the client received the work. For trusted systems, however, this variation adds a level of encryption for no real gain in accountability.

The transaction for specific usage rights are now discussed.

The Copy Transaction

A Copy transaction is a request to make one or more independent copies of the work with the same or lesser usage rights. Copy differs from the extraction right discussed later in that it refers to entire digital works or entire folders containing digital works. A copy operation cannot be used to remove a portion of a digital work.

- The requester sends the server a message to initiate the Copy Transaction. This message indicates the work to be copied, the version of the copy right to be used for the transaction, the destination address information (location in a folder) for placing the work, the file data for the work (including its size), and the number of copies requested.
- The repositories perform the common opening transaction steps.
- The server transmits the requested contents and data to the client according to the transmission protocol. If a Next-Set-Of-Rights has been provided in the version of the right, those rights are transmitted as the rights for the work. Otherwise, the rights of the original are transmitted. In any event, the Copy-Count field for the copy of the digital work being sent right is set to the number-of-copies requested.
- The requester records the work contents, data, and usage rights and stores the work. It records the date and time that the copy was made in the properties of the digital work.
- The repositories perform the common closing transaction steps.

The Transfer Transaction

A Transfer transaction is a request to move copies of the work with the same or lesser usage rights to another repository. In contrast with a copy transaction, this results in removing the work copies from the server.

- The requester sends the server a message to initiate the Transfer Transaction. This message indicates the work to be transferred, the version of the transfer right to be used in the transaction, the destination address information for placing the work, the file data for the work, and the number of copies involved.
- The repositories perform the common opening transaction steps.
- The server transmits the requested contents and data to the requester according to the transmission protocol. If a Next-Set-Of-Rights has been provided, those rights are transmitted as the rights for the work. Otherwise, the rights of the original are transmitted. In either case, the Copy-Count field for the transmitted rights are set to the number-of-copies requested.

- The requester records the work contents, data, and usage rights and stores the work.
- The server decrements its copy count by the number of copies involved in the transaction.
- The repositories perform the common closing transaction steps.
- If the number of copies remaining in the server is now zero, it erases the digital work from its memory.

The Loan Transaction

A loan transaction is a mechanism for loaning copies of a digital work. The maximum duration of the loan is determined by an internal parameter of the digital work. Works are automatically returned after a predetermined time period.

- The requester sends the server a message to initiate the Transfer Transaction. This message indicates the work to be loaned, the version of the loan right to be used in the transaction, the destination address information for placing the work, the number of copies involved, the file data for the work, and the period of the loan.
- The server checks the validity of the requested loan period, and ends with an error if the period is not valid. Loans for a loaned copy cannot extend beyond the period of the original loan to the server.
- The repositories perform the common opening transaction steps.
- The server transmits the requested contents and data to the requester. If a Next-Set-Of-Rights has been provided, those rights are transmitted as the rights for the work. Otherwise, the rights of the original are transmitted, as modified to reflect the loan period.
- The requester records the digital work contents, data, usage rights, and loan period and stores the work.
- The server updates the usage rights information in the digital work to reflect the number of copies loaned out.
- The repositories perform the common closing transaction steps.
- The server updates the usage rights data for the digital work. This may preclude use of the work until it is returned from the loan. The user on the requester platform can now use the transferred copies of the digital work. A user accessing the original repository cannot use the digital work , unless there are copies remaining. What happens next depends on the order of events in time.

Case 1. If the time of the loan period is not yet exhausted and the requester sends the repository a Return message.

- The return message includes the requester identification, and the transaction ID.
- The server decrements the copies-in-use field by the number of copies that were returned. (If the number of digital works returned is greater than the number actually borrowed, this is treated as an error.) This step may now make the work available at the server for other users.
- The requester deactivates its copies and removes the contents from its memory.

Case 2. If the time of the loan period is exhausted and the requester has not yet sent a Return message.

- The server decrements the copies-in-use field by the number digital works that were borrowed.
- The requester automatically deactivates its copies of the digital work. It terminates all current uses and erases the digital work copies from memory. One question is why a requester would ever return a work earlier than the period of the loan, since it would be returned automatically anyway. One reason for early return is that there may be a metered fee which determines the cost of the loan. Returning early may reduce that fee.

The Play Transaction

A play transaction is a request to use the contents of a work. Typically, to "play" a work is to send the digital work through some kind of transducer, such as a speaker or a display device. The request implies the intention that the contents will not be communicated digitally to any other system. For example, they will not be sent to a printer, recorded on any digital medium, retained after the transaction or sent to another repository.

This term "play" is natural for examples like playing music, playing a movie, or playing a video game. The general form of play means that a "player" is used to use the digital work. However, the term play covers all media and kinds of recordings. Thus one would "play" a digital work, meaning, to render it for reading, or play a computer program, meaning to execute it. For a digital ticket the player would be a digital ticket agent.

- The requester sends the server a message to initiate the play transaction. This message indicates the work to be

played, the version of the play right to be used in the transaction, the identity of the player being used, and the file data for the work.

- The server checks the validity of the player identification and the compatibility of the player identification with the player specification in the right. It ends with an error if these are not satisfactory.
- The repositories perform the common opening transaction steps.
- The server and requester read and write the blocks of data as requested by the player according to the transmission protocol. The requester plays the work contents, using the player.
- When the player is finished, the player and the requester remove the contents from their memory.
- The repositories perform the common closing transaction steps.

The Print Transaction

A Print transaction is a request to obtain the contents of a work for the purpose of rendering them on a "printer." We use the term "printer" to include the common case of writing with ink on paper. However, the key aspect of "printing" in our use of the term is that it makes a copy of the digital work in a place outside of the protection of usage rights. As with all rights, this may require particular authorization certificates.

Once a digital work is printed, the publisher and user are bound by whatever copyright laws are in effect. However, printing moves the contents outside the control of repositories. For example, absent any other enforcement mechanisms, once a digital work is printed on paper, it can be copied on ordinary photocopying machines without intervention by a repository to collect usage fees. If the printer to a digital disk is permitted, then that digital copy is outside of the control of usage rights. Both the creator and the user know this, although the creator does not necessarily give tacit consent to such copying, which may violate copyright laws.

- The requester sends the server a message to initiate a Print transaction. This message indicates the work to be played, the identity of the printer being used, the file data for the work, and the number of copies in the request.
- The server checks the validity of the printer identification and the compatibility of the printer identification with the printer specification in the right. It ends with an error if these are not satisfactory.
- The repositories perform the common opening transaction steps.
- The server transmits blocks of data according to the transmission protocol.
- The requester prints the work contents, using the printer.
- When the printer is finished, the printer and the requester remove the contents from their memory.
- The repositories perform the common closing transaction steps.

The Backup Transaction

A Backup transaction is a request to make a backup copy of a digital work, as a protection against media failure. In the context of repositories, secure backup copies differ from other copies in three ways: (1) they are made under the control of a Backup transaction rather than a Copy transaction, (2) they do not count as regular copies, and (3) they are not usable as regular copies. Generally, backup copies are encrypted.

Although backup copies may be transferred or copied, depending on their assigned rights, the only way to make them useful for playing, printing or embedding is to restore them.

The output of a Backup operation is both an encrypted data file that contains the contents and description of a work, and a restoration file with an encryption key for restoring the encrypted contents. In many cases, the encrypted data file would have rights for " printing" it to a disk outside of the protection system, relying just on its encryption for security. Such files could be stored anywhere that was physically safe and convenient. The restoration file would be held in the repository. This file is necessary for the restoration of a backup copy. It may have rights for transfer between repositories.

- The requester sends the server a message to initiate a backup transaction. This message indicates the work to be backed up, the version of the backup right to be used in the transaction, the destination address information for placing the backup copy, the file data for the work.
- The repositories perform the common opening transaction steps.
- The server transmits the requested contents and data to the requester. If a Next-Set-Of-Rights has been provided, those rights are transmitted as the rights for the work. Otherwise, a set of default rights for backup files of the original are transmitted by the server.
- The requester records the work contents, data, and usage rights. It then creates a one-time key and encrypts the contents file. It saves the key information in a restoration file.
- The repositories perform the common closing transaction steps.

In some cases, it is convenient to be able to archive the large, encrypted contents file to secure offline storage, such as a magneto-optical storage system or magnetic tape. This creation of a non-repository archive file is as secure as the encryption process. Such non-repository archive storage is considered a form of "printing" and is controlled by a print right with a specified "archive-printer." An archive-printer device is programmed to save the encrypted contents file (but not the description file) offline in such a way that it can be retrieved.

The Restore Transaction

A Restore transaction is a request to convert an encrypted backup copy of a digital work into a usable copy. A restore operation is intended to be used to compensate for catastrophic media failure. Like all usage rights, restoration rights can include fees and access tests including authorization checks.

- The requester sends the server a message to initiate a Restore transaction. This message indicates the work to be restored, the version of the restore right for the transaction, the destination address information for placing the work, and the file data for the work.
- The server verifies that the contents file is available (i.e. a digital work corresponding to the request has been backed-up.) If it is not, it ends the transaction with an error.
- The repositories perform the common opening transaction steps.
- The server retrieves the key from the restoration file. It decrypts the work contents, data, and usage rights.
- The server transmits the requested contents and data to the requester according to the transmission protocol. If a Next-Set-Of-Rights has been provided, those rights are transmitted as the rights for the work. Otherwise, a set of default rights for backup files of the original are transmitted by the server.
- The requester stores the digital work.
- The repositories perform the common closing transaction steps.

The Delete Transaction

A Delete transaction deletes a digital work or a number of copies of a digital work from a repository. Practically all digital works would have delete rights.

- The requester sends the server a message to initiate a delete transaction. This message indicates the work to be deleted , the version of the delete right for the transaction.
- The repositories perform the common opening transaction steps.
- The server deletes the file, erasing it from the file system.
- The repositories perform the common closing transaction steps.

The Directory Transaction

A Directory transaction is a request for information about folders, digital works, and their parts. This amounts to roughly the same idea as protection codes in a conventional file system like TENEX, except that it is generalized to the full power of the access specifications of the usage rights language.

The Directory transaction has the important role of passing along descriptions of the rights and fees associated with a digital work. When a user wants to exercise a right, the user interface of his repository implicitly makes a directory request to determine the versions of the right that are available. Typically these are presented to the user -- such as with different choices of billing for exercising a right. Thus, many directory transactions are invisible to the user and are exercised as part of the normal process of exercising all rights.

- The requester sends the server a message to initiate a Directory transaction. This message indicates the file or folder that is the root of the directory request and the version of the di rectory right used for the transaction.
- The server verifies that the information is accessible to the requester. In particular, it does not return the names of any files that have a HIDE-NAME status in their directory specifications, and it does not return the parts of any folders or files that have HIDE-PARTS in their specification. If the information is not accessible, the server ends the transaction with an error.
- The repositories perform the common opening transaction steps.
- The server sends the requested data to the requester according to the transmission protocol.
- The requester records the data.
- The repositories perform the common closing transaction steps.

The Folder Transaction

A Folder transaction is a request to create or rename a folder, or to move a work between folders. Together with Directory rights, Folder rights control the degree to which organization of a repository can be accessed or modified from another repository.

- The requester sends the server a message to initiate a Folder transaction. This message indicates the folder that is the root of the folder request, the version of the folder right for the transaction, an operation, and data. The operation can be one of create, rename, and move file. The data are the specifications required for the operation, such as a specification of a folder or digital work and a name.
- The repositories perform the common opening transaction steps.
- The server performs the requested operation -- creating a folder, renaming a folder, or moving a work between folders.
- The repositories perform the common closing transaction steps.

The Extract Transaction

A extract transaction is a request to copy a part of a digital work and to create a new work containing it. The extraction operation differs from copying in that it can be used to separate a part of a digital work from d-blocks or shells that place additional restrictions or fees on it. The extraction operation differs from the edit operation in that it does not change the contents of a work, only its embedding in d-blocks. Extraction creates a new digital work.

- The requester sends the server a message to initiate an Extract transaction. This message indicates the part of the work to be extracted, the version of the extract right to be used in the transaction, the destination address information for placing the part as a new work, the file data for the work, and the number of copies involved.
- The repositories perform the common opening transaction steps.
- The server transmits the requested contents and data to the requester according to the transmission protocol. If a Next-Set-Of-Rights has been provided, those rights are transmitted as the rights for the new work. Otherwise, the rights of the original are transmitted. The Copy-Count field for this right is set to the number-of-copies requested.
- The requester records the contents, data, and usage rights and stores the work. It records the date and time that new work was made in the properties of the work.
- The repositories perform the common closing transaction steps.

The Embed Transaction

An embed transaction is a request to make a digital work become a part of another digital work or to add a shell d-block to enable the adding of fees by a distributor of the work.

- The requester sends the server a message to initiate an Embed transaction. This message indicates the work to be embedded, the version of the embed right to be used in the transaction, the destination address information for placing the part as a a work, the file data for the work, and the number of copies involved.
- The server checks the control specifications for all of the rights in the part and the destination. If they are incompatible, the server ends the transaction with an error.
- The repositories perform the common opening transaction steps.
- The server transmits the requested contents and data to the requester according to the transmission protocol. If a Next-Set-Of-Rights has been provided, those rights are transmitted as the rights for the new work. Otherwise, the rights of the original are transmitted. The Copy-Count field for this right is set to the number-of-copies requested.
- The requester records the contents, data, and usage rights and embeds the work in the destination file.
- The repositories perform the common closing transaction steps.

The Edit Transaction

An Edit transaction is a request to make a new digital work by copying, selecting and modifying portions of an existing digital work. This operation can actually change the contents of a digital work. The kinds of changes that are permitted depend on the process being used. Like the extraction operation, edit operates on portions of a digital work. In contrast with the extract operation, edit does not affect the rights or location of the work. It only changes the contents. The kinds of changes permitted are determined by the type specification of the processor specified in the rights. In the currently preferred embodiment, an edit transaction changes the work itself and does not make a new work. However,

it would be a reasonable variation to cause a new copy of the work to be made.

- The requester sends the server a message to initiate an Edit transaction. This message indicates the work to be edited, the version of the edit right to be used in the transaction, the file data for the work (including its size), the process-ID for the process, and the number of copies involved.
- The server checks the compatibility of the process-ID to be used by the requester against any process-ID specification in the right. If they are incompatible, it ends the transaction with an error.
- The repositories perform the common opening transaction steps.
- The requester uses the process to change the contents of the digital work as desired. (For example, it can select and duplicate parts of it; combine it with other information; or compute functions based on the information. This can amount to editing text, music, or pictures or taking whatever other steps are useful in creating a derivative work.)
- The repositories perform the common closing transaction steps.

The edit transaction is used to cover a wide range of kinds of works. The category describes a process that takes as its input any portion of a digital work and then modifies the input in some way. For example, for text, a process for editing the text would require edit rights. A process for "summarizing" or counting words in the text would also be considered editing. For a music file, processing could involve changing the pitch or tempo, or adding reverberations, or any other audio effect. For digital video works, anything which alters the image would require edit rights. Examples would be colorizing, scaling, extracting still photos, selecting and combining frames into story boards, sharpening with signal processing, and so on.

Some creators may want to protect the authenticity of their works by limiting the kinds of processes that can be performed on them. If there are no edit rights, then no processing is allowed at all. A processor identifier can be included to specify what kind of process is allowed. If no process identifier is specified, then arbitrary processors can be used. For an example of a specific process, a photographer may want to allow use of his photograph but may not want it to be colorized. A musician may want to allow extraction of portions of his work but not changing of the tonality.

Authorization Transactions

There are many ways that authorization transactions can be defined. In the following, our preferred way is to simply define them in terms of other transactions that we already need for repositories. Thus, it is convenient sometimes to speak of "authorization transactions," but they are actually made up of other transactions that repositories already have.

A usage right can specify an authorization-ID, which identifies an authorization object (a digital work in a file of a standard format) that the repository must have and which it must process. The authorization is given to the generic authorization (or ticket) server of the repository which begins to interpret the authorization.

As described earlier, the authorization contains a server identifier, which may just be the generic authorization server or it may be another server. When a remote authorization server is required, it must contain a digital address. It may also contain a digital certificate.

If a remote authorization server is required, then the authorization process first performs the following steps:

- The generic authorization server attempts to set up the communications channel. (If the channel cannot be set up, then authorization fails with an error.)
- When the channel is set up, it performs a registration process with the remote repository. (If registration fails, then the authorization fails with an error.)
- When registration is complete, the generic authorization server invokes a "Play" transaction with the remote repository, supplying the authorization document as the digital work to be played, and the remote authorization server (a program) as the "player." (If the player cannot be found or has some other error, then the authorization fails with an error.)
- The authorization server then "plays" the authorization. This involves decrypting it using either the public key of the master repository that issued the certificate or the session key from the repository that transmitted it. The authorization server then performs various tests. These tests vary according to the authorization server. They include such steps as checking issue and validity dates of the authorization and checking any hot-lists of known invalid authorizations. The authorization server may require carrying out any other transactions on the repository as well, such as checking directories, getting some person to supply a password, or playing some other digital work. It may also invoke some special process for checking information about locations or recent events. The "script" for such steps is contained within the authorization server.
- If all of the required steps are completed satisfactorily, the authorization server completes the transaction normally, signaling that authorization is granted.

The Install Transaction

An Install transaction is a request to install a digital work as runnable software on a repository. In a typical case, the requester repository is a rendering repository and the software would be a new kind or new version of a player. Also in a typical case, the software would be copied to file system of the requester repository before it is installed.

- The requester sends the server an Install message. This message indicates the work to be installed, the version of the Install right being invoked, and the file data for the work (including its size).
- The repositories perform the common opening transaction steps.
- The requester extracts a copy of the digital certificate for the software. If the certificate cannot be found or the master repository for the certificate is not known to the requester, the transaction ends with an error.
- The requester decrypts the digital certificate using the public key of the master repository, recording the identity of the supplier and creator, a key for decrypting the software, the compatibility information, and a tamper-checking code. (This step certifies the software.)
- The requester decrypts the software using the key from the certificate and computes a check code on it using a 1-way hash function. If the check-code does not match the tamper-checking code from the certificate, the installation transaction ends with an error. (This step assures that the contents of the software, including the various scripts, have not been tampered with.)
- The requester retrieves the instructions in the compatibility-checking script and follows them. If the software is not compatible with the repository, the installation transaction ends with an error. (This step checks platform compatibility.)
- The requester retrieves the instructions in the installation script and follows them. If there is an error in this process (such as insufficient resources), then the transaction ends with an error. Note that the installation process puts the runnable software in a place in the repository where it is no longer accessible as a work for exercising any usage rights other than the execution of the software as part of repository operations in carrying out other transactions.
- The repositories perform the common closing transaction steps.

The Uninstall Transaction

An Uninstall transaction is a request to remove software from a repository. Since uncontrolled or incorrect removal of software from a repository could compromise its behavioral integrity, this step is controlled.

- The requester sends the server an Uninstall message. This message indicates the work to be uninstalled, the version of the Uninstall right being invoked, and the file data for the work (including its size).
- The repositories perform the common opening transaction steps.
- The requester extracts a copy of the digital certificate for the software. If the certificate cannot be found or the master repository for the certificate is not known to the requester, the transaction ends with an error.
- The requester checks whether the software is installed. If the software is not installed, the transaction ends with an error.
- The requester decrypts the digital certificate using the public key of the master repository, recording the identity of the supplier and creator, a key for decrypting the software, the compatibility information, and a tamper-checking code. (This step authenticates the certification of the software, including the script for uninstalling it.)
- The requester decrypts the software using the key from the certificate and computes a check code on it using a 1-way hash function. If the check-code does not match the tamper-checking code from the certificate, the installation transaction ends with an error. (This step assures that the contents of the software, including the various scripts, have not been tampered with.)
- The requester retrieves the instructions in the uninstallation script and follows them. If there is an error in this process (such as insufficient resources), then the transaction ends with an error.
- The repositories perform the common closing transaction steps.

Claims

1. A distribution system for distributing digital works, said digital works having one or more usage rights attached thereto, said distribution system comprising:

   a grammar for creating instances of usage rights indicating a manner by which a possessor of an associated digital work may transport said associated digital work;

means for creating usage rights from said grammar;
means for attaching created usage rights to a digital work;
a requester repository for accessing digital works, said requester repository having means for generating usage transactions, each said usage transaction specifying a usage right;
a server repository for storing digital works with attached created usage rights, said server repository having means for processing usage transactions from said requester repository to determine if access to a digital work may be granted.

2. The distribution system as recited in Claim 1 wherein said grammar further specifies a default plurality of conditions for an instance of a usage right, wherein said one or more conditions must be satisfied before said usage right may be exercised.

3. The distribution system as recited in Claim 2 wherein said means for creating usage rights from said grammar is further comprised of means for changing said default plurality of conditions for an instance of a usage right.

4. The distribution system as recited in Claim 1 wherein said digital work is a software program.

5. The distribution system as recited in Claim 1 wherein said grammar is further for creating a first version of a usage right having a first set of conditions and a second version of said usage right having a second set of conditions.

6. A computer based system for controlling distribution and use of digital works comprising:

a usage rights grammar for creating instances of usages rights which define how a digital work may be used or distributed, said usage rights grammar comprising a first plurality of grammar elements for defining transport usage rights and a second plurality of grammar elements for defining rendering usage rights;
means for attaching usage rights to digital works;
a plurality of repositories for storing and exchanging digital works, each of said plurality of repositories comprising :
means for storing digital works and their attached usage rights;
transaction processing means having a requester mode of operation for requesting access to a requested digital work, said request specifying a usage right, and a server mode of operation for processing requests to access said requested digital work based on said usage right specified in said request and the usage rights attached to said requested digital work; and
a coupling means for coupling to another of said plurality of repositories across a communications medium.

7. The computer based system for controlling distribution and use of digital works as recited in Claim 6 wherein said first plurality of grammar elements is comprised of:

a loan grammar element for enabling a digital work to be loaned to another repository;
a copy grammar element for enabling a copy of a digital work to be made and transported to another repository; and
a transfer grammar element for enabling a digital work to be transferred to another repository.

8. The computer based system for controlling distribution and use of digital works as recited in Claim 6 or Claim 7 wherein said second plurality of grammar elements is comprised of:

a play grammar element for enabling a digital work to be rendered on a specified class of player device; and
a print grammar element for enabling a digital work to be printed on a specified class of printer device.

9. The computer based system for controlling distribution and use of digital works as recited in any one of Claims 6 to 8 wherein said grammar comprises one or more further pluralities of grammar elements, for defining file management usage rights, for enabling a digital work to be used in the creation of a new digital work,for enabling the secure installation and uninstallation of digital works comprising of software programs, or for providing a set of creator specified conditions which must be satisfied for each instantiation of a usage right defined by a grammar element.

10. A method for controlling distribution and use of digital works comprising the steps of:

a) creating a set of usage rights from a usage rights grammar, each of said usage rights defining a specific instance of how a digital work may be used or distributed, each of said usage rights specifying one or more conditions which must be satisfied in order for said usage right to be exercised;
b) attaching said set of usage rights to a digital work;
c) storing said digital work and its attached usage rights in a first repository;
d) a second repository initiating a request to access said digital work in said first repository, said request specifying a usage right;
e) said first repository receiving said request from said second repository;
f) said first repository determining if said specified usage right is attached to said digital work;
g) said first repository denying access to said digital work if said identified usage right is not attached to said digital work;
h) if said identified usage right is attached to said digital work, said first repository determining if conditions specified by said usage right are satisfied;
i) if said conditions are not satisfied, said first repository denying access to said digital work;
j) if said conditions are satisfied, said first repository transmitting said digital work to said second repository.

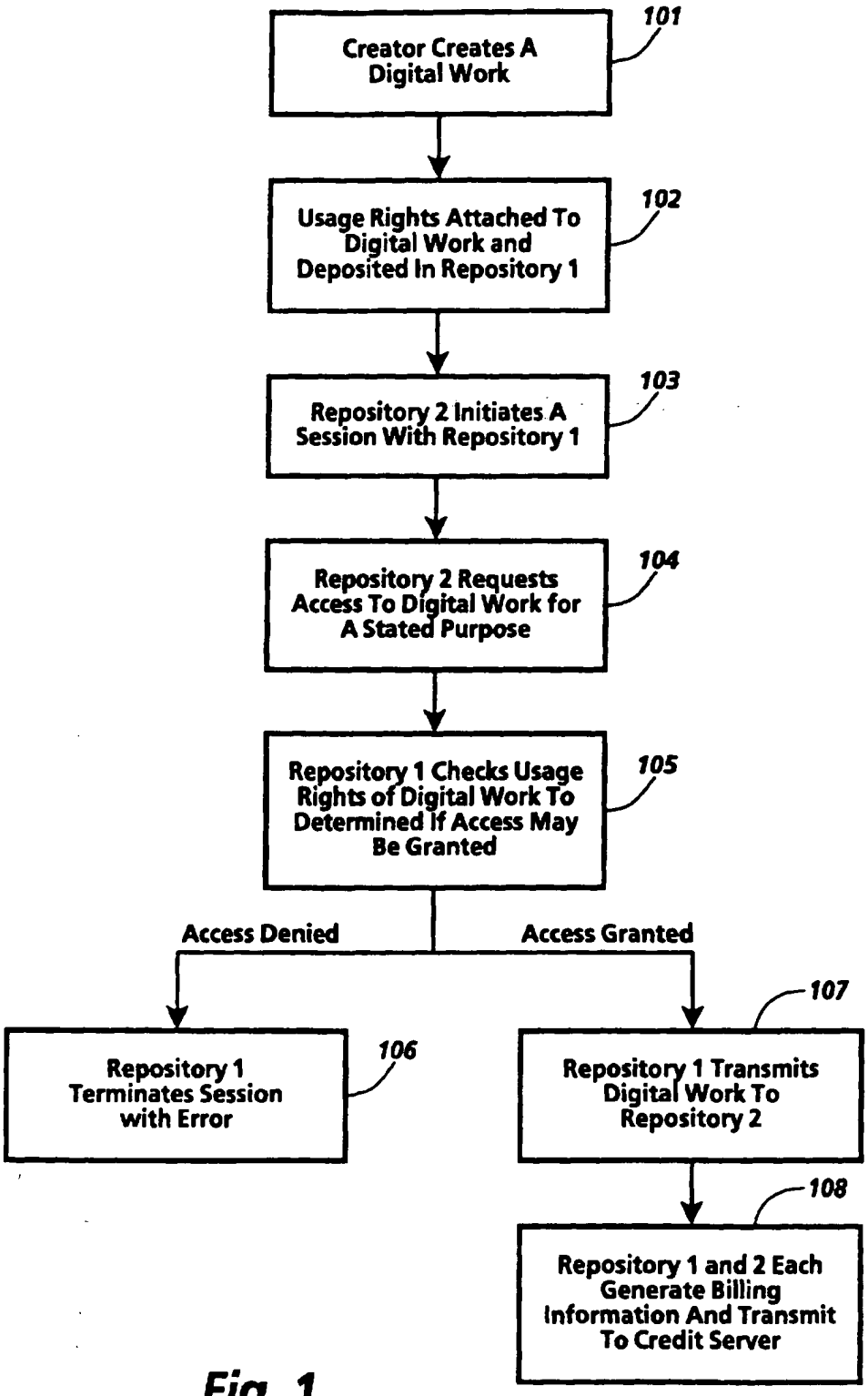


Fig. 1

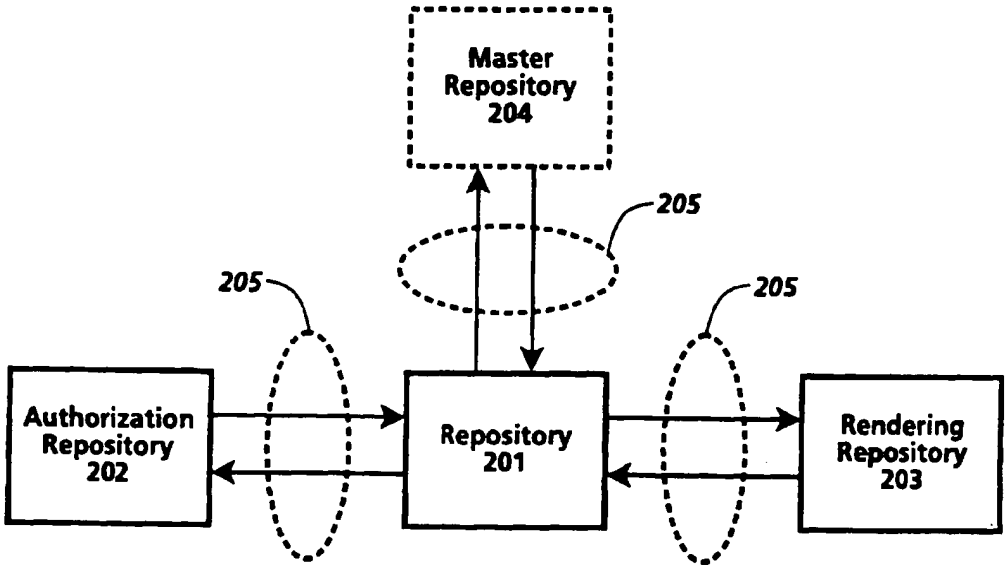


Fig. 2

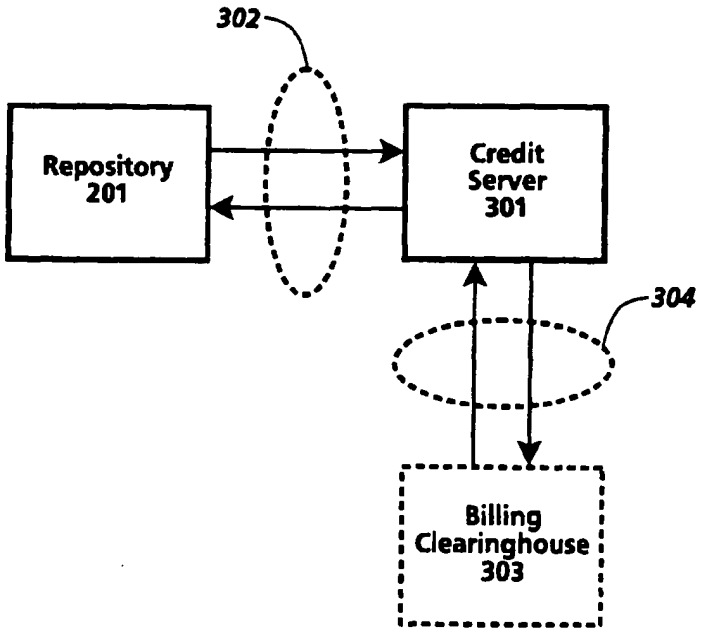


Fig. 3

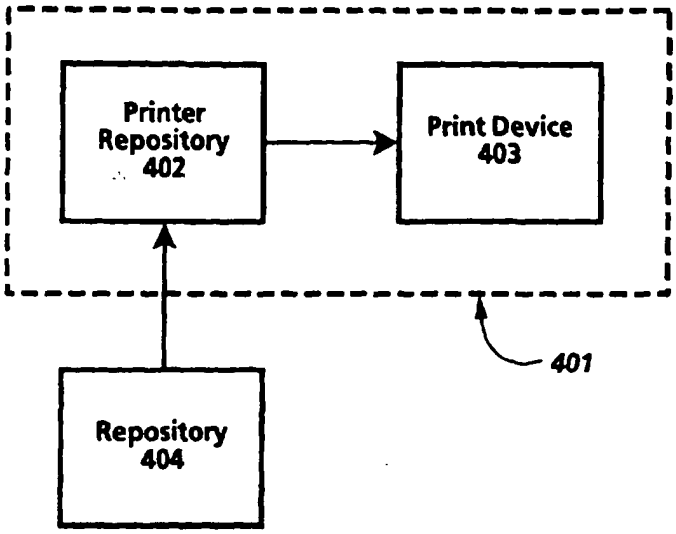


Fig. 4a

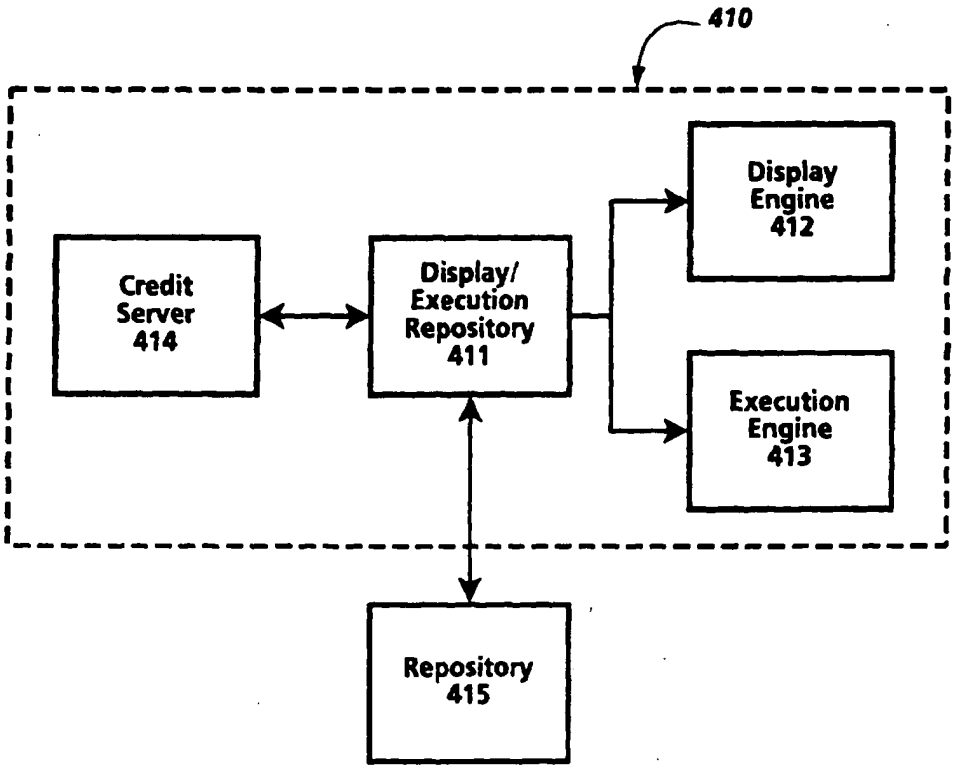


Fig. 4b

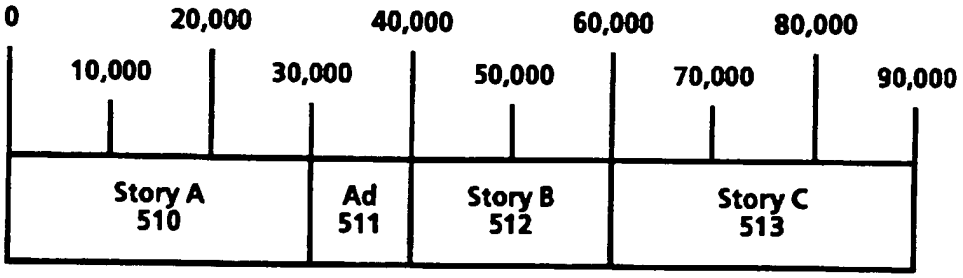


Fig. 5

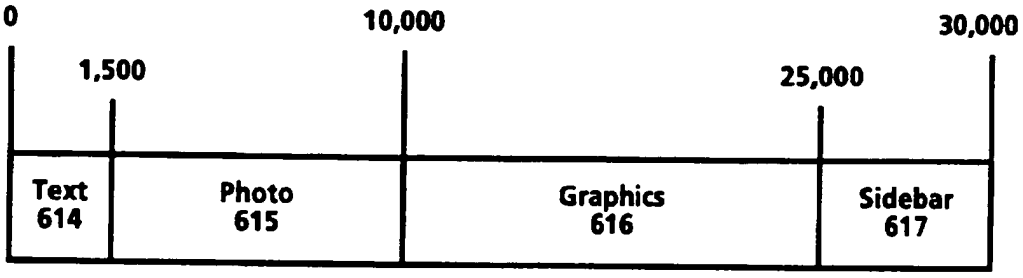


Fig. 6

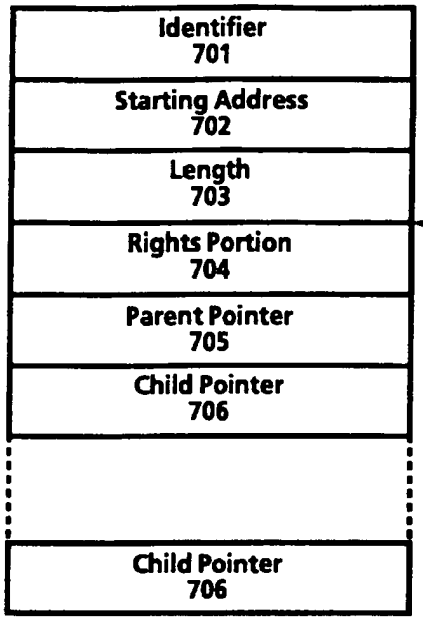


Fig. 7

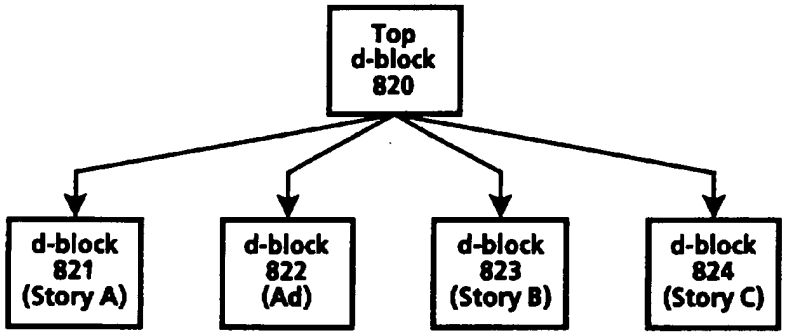


Fig. 8

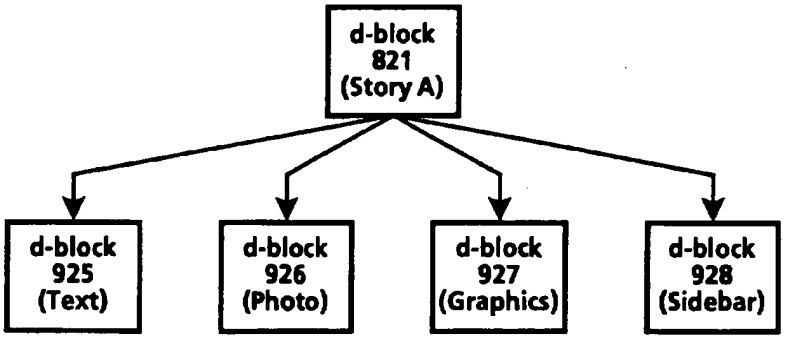


Fig. 9

Fig.10

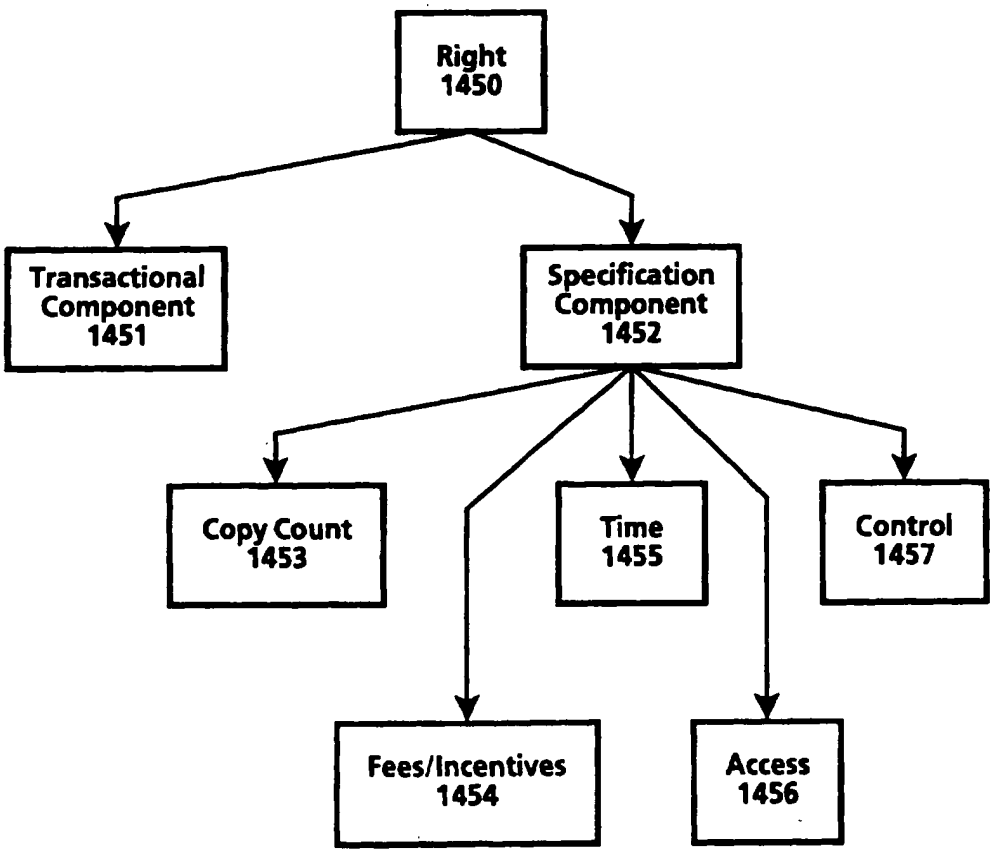


Fig.14

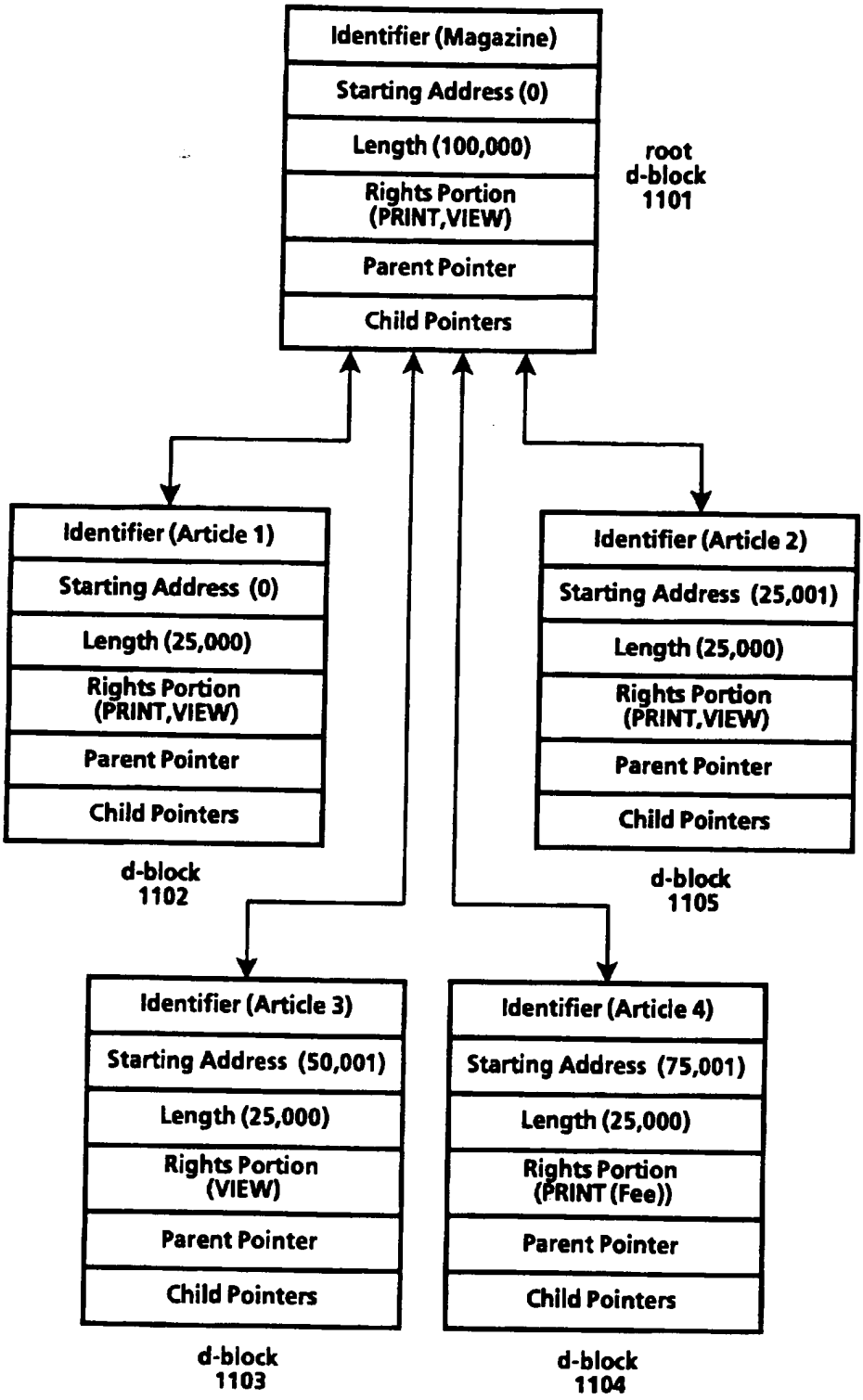


Fig.11

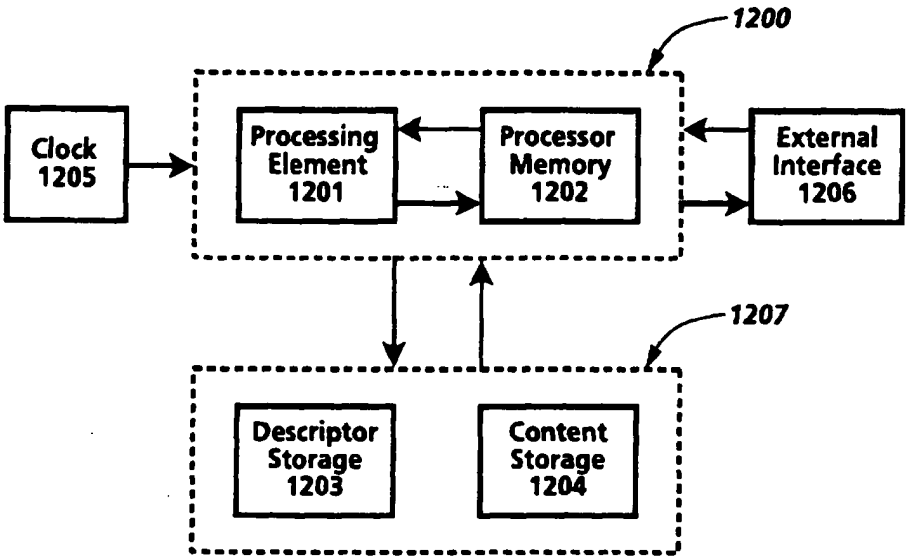


Fig.12

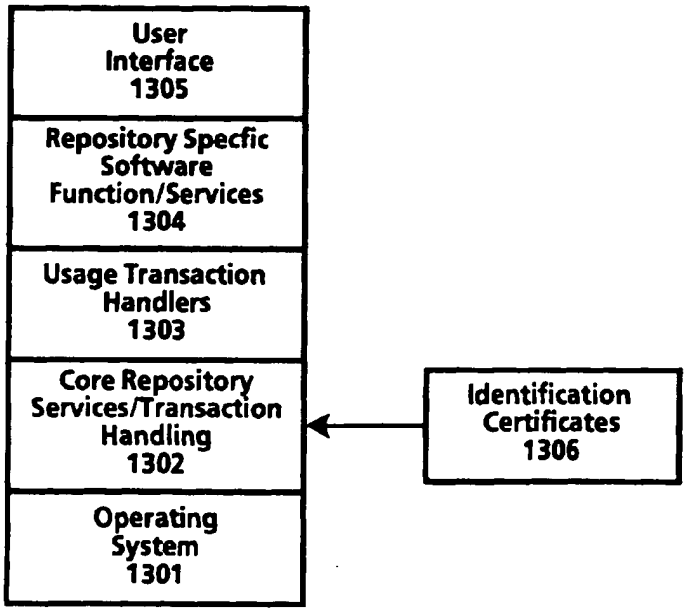


Fig.13

1501 ~ **Digital Work Rights:** = (Rights*)

1502 ~ **Right** := (Right-Code {Copy-Count} {Control-Spec} {Time-Spec } {Access-Spec} {Fee-Spec})

1503 ~ **Right-Code** := Render-Code | Transport-Code | File-Management-Code| Derivative-Works- Code | Configuration-Code

1504 ~ **Render-Code** := [Play : {Player: Player-ID} | Print: {Printer: Printer-ID}]

1505 ~ **Transport-Code** := [Copy | Transfer | Loan {Remaining-Rights: Next-Set-of-Rights}]{(Next-Copy-Rights: Next-Set-of-Rights)}

1506 ~ **File-Management-Code** := Backup {Back-Up-Copy-Rights: Next-Set-of-Rights} | Restore | Delete | Folder | Directory {Name: Hide-Local | Hide-Remote} {Parts: Hide-Local | Hide-Remote}

1507 ~ **Derivative-Works-Code** := [Extract | Embed | Edit{Process: Process-ID}] {Next-Copy-Rights : Next-Set-of Rights}

1508 ~ **Configuration-Code** := Install | Uninstall

1509 ~ **Next-Set-of-Rights** := {(Add: Set-Of-Rights)} {(Delete: Set-Of-Rights)} {(Replace: Set-Of-Rights)}{(Keep: Set-Of-Rights)}

1510 ~ **Copy-Count** := (Copies:positive-integer | 0 | Unlimited)

1511 ~ **Control-Spec** := (Control: {Restrictable | Unrestrictable} {Unchargeable | Chargeable})

1512 ~ **Time-Spec** := ({Fixed-Interval | Sliding-Interval | Meter-Time} Until: Expiration-Date)

1513 ~ **Fixed-Interval** := From: Start-Time

1514 ~ **Sliding-Interval** := Interval: Use-Duration

1515 ~ **Meter-Time:** = Time-Remaining: Remaining-Use

1516 ~ **Access-Spec** := ({SC: Security-Class} {Authorization: Authorization-ID*} {Other-Authorization: Authorization-ID*} {Ticket: Ticket-ID})

1517 ~ **Fee-Spec:** = {Scheduled-Discount} Regular-Fee-Spec | Scheduled-Fee-Spec | Markup-Spec

1518 ~ **Scheduled-Discount:** = Scheduled-Discount: (Scheduled-Discount: (Time-Spec Percentage)*)

1519 ~ **Regular-Fee-Spec** := ({Fee: | Incentive: } [Per-Use-Spec | Metered-Rate-Spec | Best-Price-Spec | Call-For-Price-Spec] {Min: Money-Unit Per: Time-Spec}{Max: Money-Unit Per: Time-Spec} To: Account-ID)

1520 ~ **Per-Use-Spec:** = Per-Use: Money-unit

1521 ~ **Metered-Rate-Spec** := Metered: Money-Unit Per: Time-Spec

1522 ~ **Best-Price-Spec** := Best-Price: Money-unit Max: Money-unit

1523 ~ **Call-For-Price-Spec** := Call-For -Price

1524 ~ **Scheduled-Fee-Spec:** = (Schedule: (Time-Spec Regular-Fee-Spec)*)

1525 ~ **Markup-Spec:** = Markup: percentage To: Account-ID

Fig.15

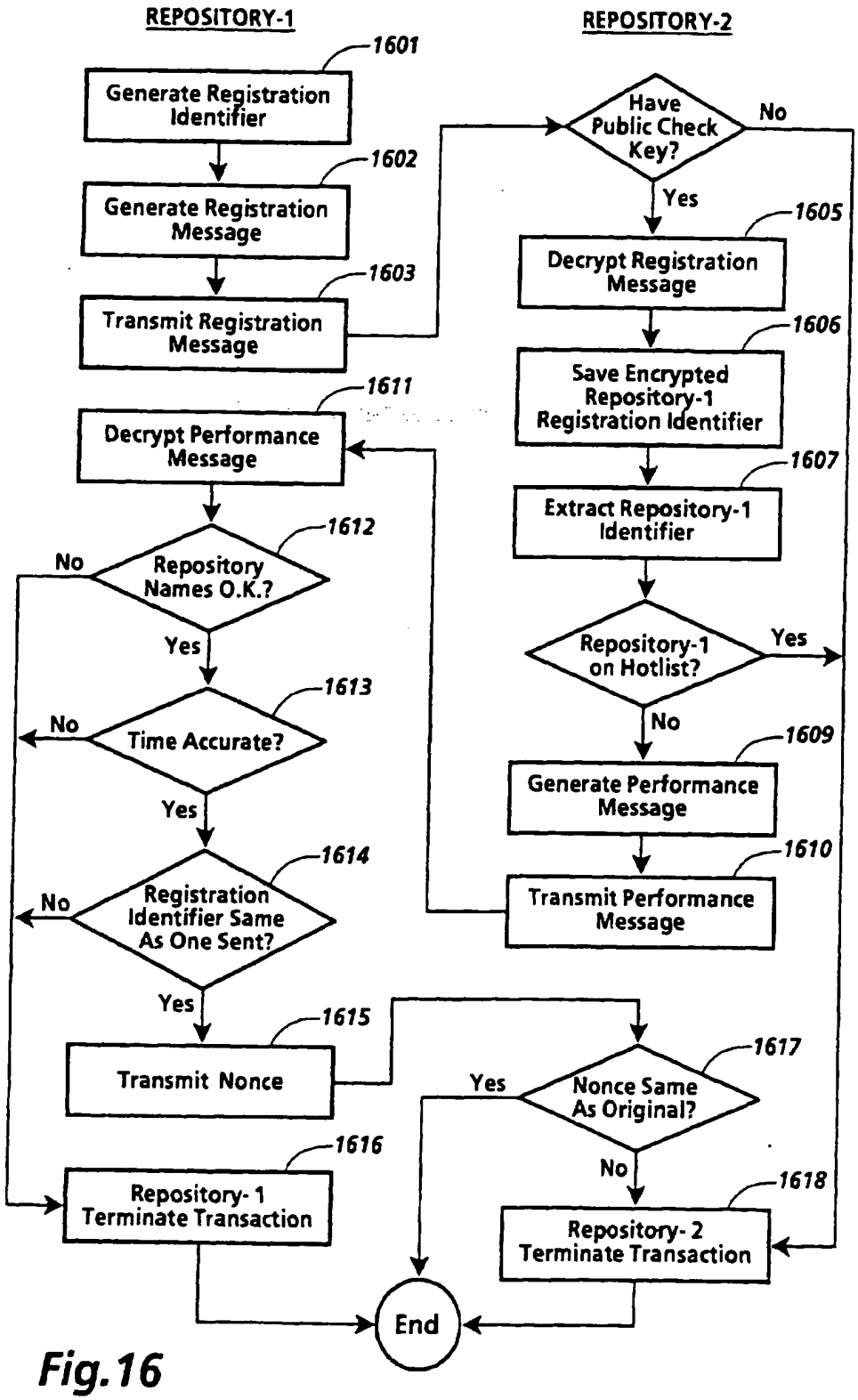


Fig.16

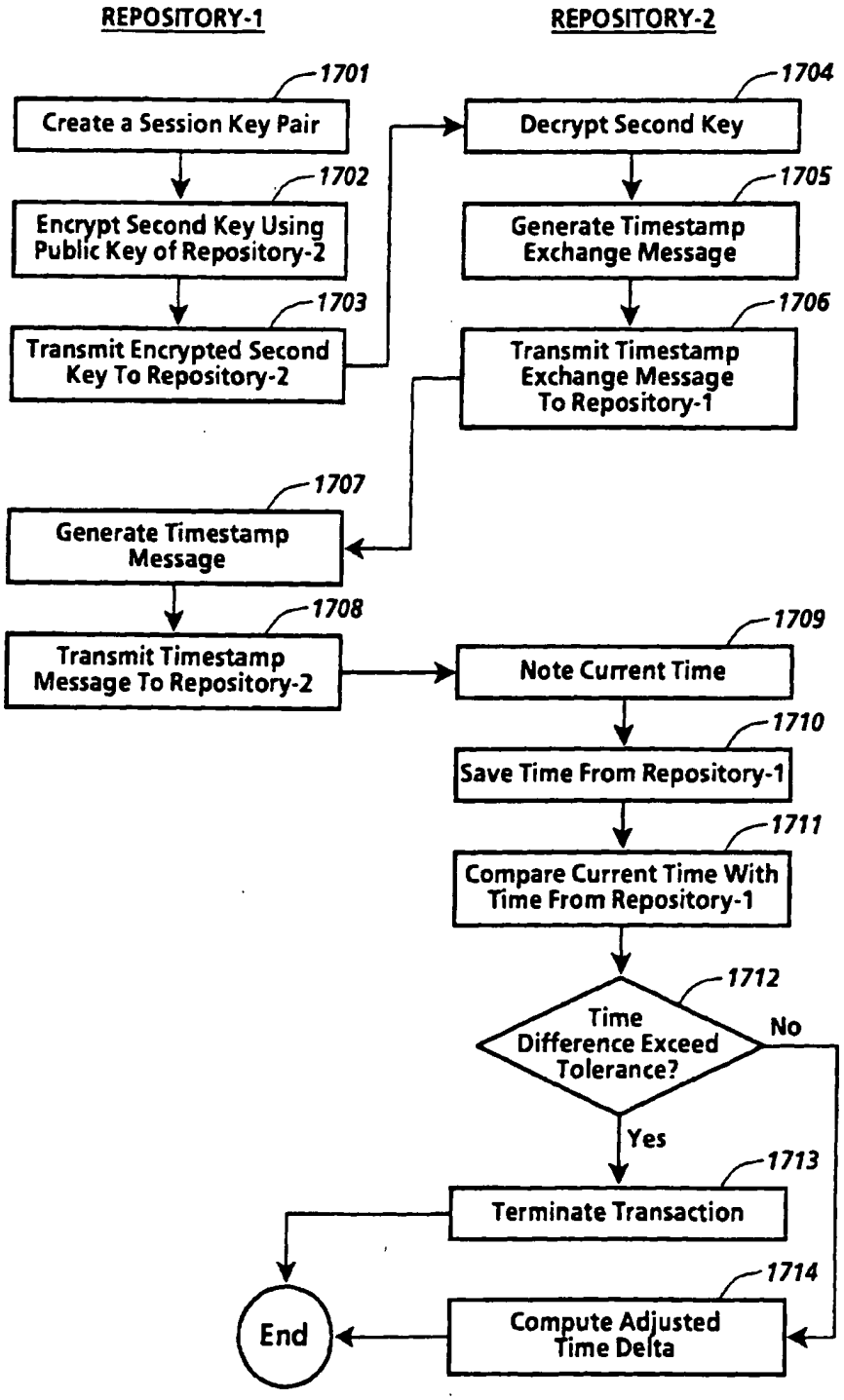


Fig.17

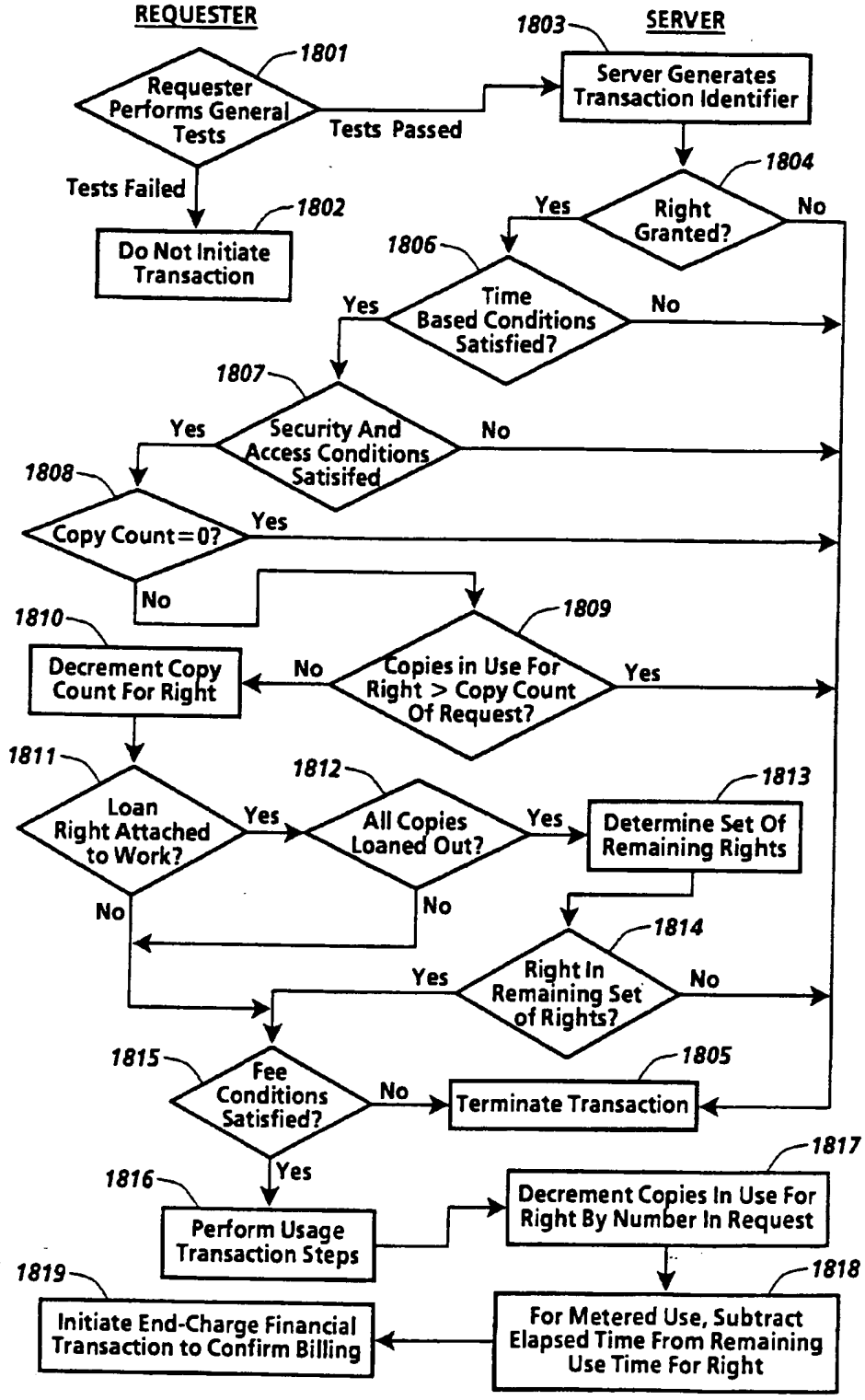


Fig.18

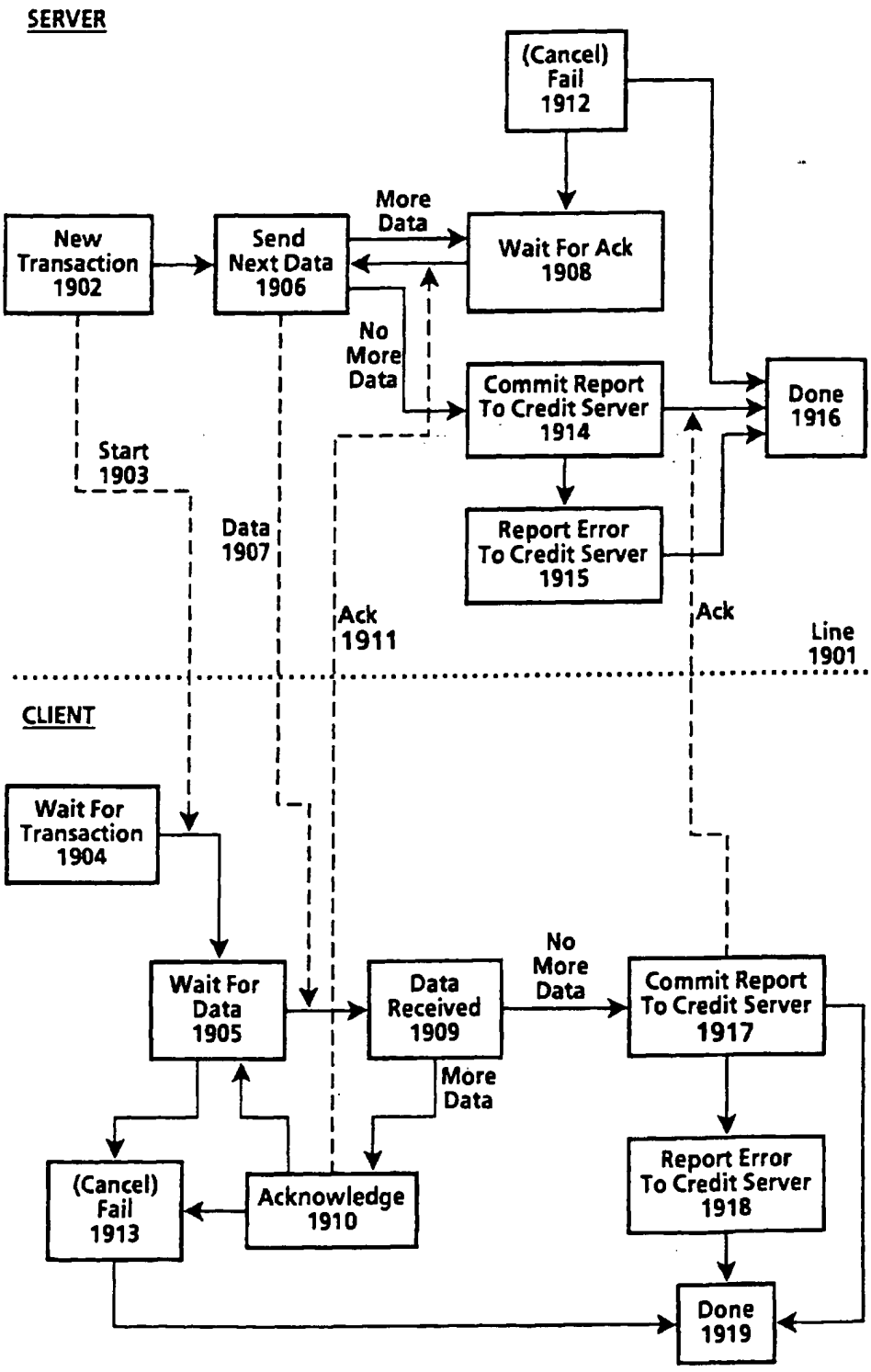


Fig.19

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 95 30 8417

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
A	WO-A-92 20022 (DIGITAL EQUIPMENT CORP.) * page 45, line 10 - page 80, line 19; figures 1-43 *	1,6,10	G06F1/00
A	US-A-5 291 596 (MIITA) * the whole document *	1,6,10	
A	GB-A-2 236 604 (SUN MICROSYSTEMS INC) * page 9, line 11 - page 20, line 15 *	1,6,10	

TECHNICAL FIELDS SEARCHED (Int.Cl.6)

G06F

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	1 April 1996	Moens, R

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 715 245 A1

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
05.06.1996 Bulletin 1996/23

(51) Int Cl.[6]: **G06F 1/00**

(21) Application number: **95308420.9**

(22) Date of filing: **23.11.1995**

(84) Designated Contracting States:
DE FR GB

(30) Priority: **23.11.1994 US 344042**

(71) Applicant: **XEROX CORPORATION**
Rochester New York 14644 (US)

(72) Inventors:
- **Stefik, Mark J.**
  Woodside, California 94062 (US)
- **Casey, Michalene M.**
  Morgan Hill, California 95037 (US)

(74) Representative: **Goode, Ian Roy**
Rank Xerox Ltd
Patent Department
Parkway
Marlow Buckinghamshire SL7 1YL (GB)

(54) System for controlling the distribution and use of digital works

(57) A system for controlling use and distribution of digital works, in which the owner of a digital work (101) attaches usage rights (102) to that work. Usage rights are granted by the "owner" of a digital work to "buyers" of the digital work. The usage rights define how a digital work may be used and further distributed by the buyer. Each right has associated with it certain optional specifications which outline the conditions and fees upon which the right may be exercised. Digital works are stored in a repository. A repository will process each request (103,104) to access a digital work by examining the corresponding usage rights (105). Digital work playback devices, coupled to the repository containing the work, are used to play, display or print the work. Access to digital works for the purposes of transporting between repositories (e.g. copying, borrowing or transfer) is carried out using a digital work transport protocol. Access to digital works for the purposes of replay by a digital work playback device(e.g. printing, displaying or executing) is carried out using a digital work playback protocol. Access is denied (106) or granted (107) depending whether the requesting repository has the required usage rights.

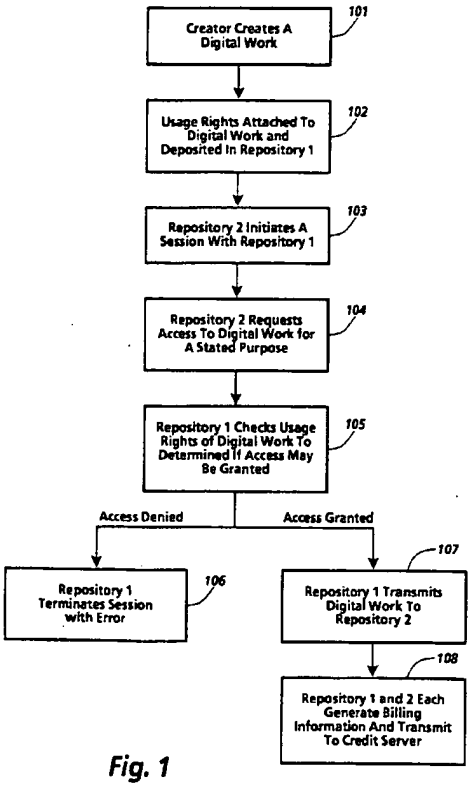


Fig. 1

EP 0 715 245 A1

Printed by Jouve, 75001 PARIS (FR)

Description

The present invention relates to the field of distribution and usage rights enforcement for digitally encoded works.

A fundamental issue facing the publishing and information industries as they consider electronic publishing is how to prevent the unauthorized and unaccounted distribution or usage of electronically published materials. Electronically published materials are typically distributed in a digital form and recreated on a computer based system having the capability to recreate the materials. Audio and video recordings, software, books and multimedia works are all being electronically published. Companies in these industries receive royalties for each accounted for delivery of the materials, e.g. the sale of an audio CD at a retail outlet. Any unaccounted distribution of a work results in an unpaid royalty (e.g. copying the audio recording CD to another digital medium.)

The ease in which electronically published works can be "perfectly" reproduced and distributed is a major concern. The transmission of digital works over networks is commonplace. One such widely used network is the Internet. The Internet is a widespread network facility by which computer users in many universities, corporations and government entities communicate and trade ideas and information. Computer bulletin boards found on the Internet and commercial networks such as CompuServ and Prodigy allow for the posting and retrieving of digital information. Information services such as Dialog and LEXIS/NEXIS provide databases of current information on a wide variety of topics. Another factor which will exacerbate the situation is the development and expansion of the National Information Infrastructure (the NII). It is anticipated that, as the NII grows, the transmission of digital works over networks will increase many times over. It would be desirable to utilize the NII for distribution of digital works without the fear of widespread unauthorized copying.

The most straightforward way to curb unaccounted distribution is to prevent unauthorized copying and transmission. For existing materials that are distributed in digital form, various safeguards are used. In the case of software, copy protection schemes which limit the number of copies that can be made or which corrupt the output when copying is detected have been employed. Another scheme causes software to become disabled after a predetermined period of time has lapsed. A technique used for workstation based software is to require that a special hardware device must be present on the workstation in order for the software to run, e.g., see US-A-4,932,054 entitled "Method and Apparatus for Protecting Computer Software Utilizing Coded Filter Network in Conjunction with an Active Coded Hardware Device." Such devices are provided with the software and are commonly referred to as dongles.

Yet another scheme is to distribute software, but which requires a "key" to enable its use. This is employed in distribution schemes where "demos" of the software are provided on a medium along with the entire product. The demos can be freely used, but in order to use the actual product, the key must be purchased. These schemes do not hinder copying of the software once the key is initially purchased.

It is an object of the present invention to provide an improved system and method for controlling the use and distribution of digital works.

The invention accordingly provides a system and method as claimed in the accompanying claims.

A system for controlling use and distribution of digital works is disclosed. A digital work is any written, aural, graphical or video based work including computer programs that has been translated to or created in a digital form, and which can be recreated using suitable rendering means such as software programs. The present invention allows the owner of a digital work to attach usage rights to the work. The usage rights for the work define how it may be used and distributed. Digital works and their usage rights are stored in a secure repository. Digital works may only be accessed by other secure repositories.

Usage rights for a digital work are embodied in a flexible and extensible usage rights grammar. Conceptually, a right in the usage rights grammar is a label attached to a predetermined behavior and conditions to exercising the right. For example, a COPY right denotes that a copy of the digital work may be made. A condition to exercising the right is the requester must pass certain security criteria. Conditions may also be attached to limit the right itself. For example, a LOAN right may be defined so as to limit the duration of which a work may be LOANed. Conditions may also include requirements that fees be paid.

A repository is comprised of a storage means for storing a digital work and its attached usage rights, an external interface for receiving and transmitting data, a processor and a clock. A repository has two primary operating modes, a server mode and a requester mode. When operating in a server mode, the repository is responding to requests to access digital works. When operating in requester mode, the repository is requesting access to a digital work.

Generally, a repository will process each request to access a digital work by examining the work's usage rights. For example, in a request to make a copy of a digital work, the digital work is examined to see if rights have been granted which would allow copies to be given out. If such a right has been granted, then conditions to exercise of the right are checked (e.g. a right to make 2 copies). If conditions associated with the right are satisfied, the copy can be made. Before transporting the digital work, any specified changes to the set of usage rights in the copy are attached to the copy of the digital work.

Repositories communicate utilizing a set of repository transactions. The repository transactions embody a set of

protocols for establishing secure sessions connections between repositories, and for processing access requests to the digital works.

Digital works are recreated on rendering systems. A rendering system is comprised of at least a rendering repository and a rendering device (e.g. a printer, display or audio system.) Rendering systems are internally secure. Access to digital works not contained within the rendering repository is accomplished via repository transactions with an external repository containing the desired digital work.

A system and method in accordance with the invention will now be described, by way of example, with reference to the accompanying drawings, in which:-

Figure 1 is a flowchart illustrating a simple instantiation of the operation of the currently preferred embodiment of the present invention.

Figure 2 is a block diagram illustrating the various repository types and the repository transaction flow between them in the currently preferred embodiment of the present invention.

Figure 3 is a block diagram of a repository coupled with a credit server in the currently preferred embodiment of the present invention.

Figures 4a and 4b are examples of rendering systems as may be utilized in the currently preferred embodiment of the present invention.

Figure 5 illustrates a contents file layout for a digital work as may be utilized in the currently preferred embodiment of the present invention.

Figure 6 illustrates a contents file layout for an individual digital work of the digital work of Figure 5 as may be utilized in the currently preferred embodiment of the present invention.

Figure 7 illustrates the components of a description block of the currently preferred embodiment of the present invention.

Figure 8 illustrates a description tree for the contents file layout of the digital work illustrated in Figure 5.

Figure 9 illustrates a portion of a description tree corresponding to the individual digital work illustrated in Figure 6.

Figure 10 illustrates a layout for the rights portion of a description block as may be utilized in the currently preferred embodiment of the present invention.

Figure 11 is a description tree wherein certain d-blocks have PRINT usage rights and is used to illustrate "strict" and "lenient" rules for resolving usage rights conflicts.

Figure 12 is a block diagram of the hardware components of a repository as are utilized in the currently preferred embodiment of the present invention.

Figure 13 is a block diagram of the functional (logical) components of a repository as are utilized in the currently preferred embodiment of the present invention.

Figure 14 is diagram illustrating the basic components of a usage right in the currently preferred embodiment of the present invention.

Figure 15 lists the usage rights grammar of the currently preferred embodiment of the present invention.

Figure 16 is a flowchart illustrating the steps of certificate delivery, hotlist checking and performance testing as performed in a registration transaction as may be performed in the currently preferred embodiment of the present invention.

Figure 17 is a flowchart illustrating the steps of session information exchange and clock synchronization as may be performed in the currently preferred embodiment of the present invention, after each repository in the registration transaction has successfully completed the steps described in Figure 16.

Figure 18 is a flowchart illustrating the basic flow for a usage transaction, including the common opening and closing step, as may be performed in the currently preferred embodiment of the present invention.

Figure 19 is a state diagram of server and client repositories in accordance with a transport protocol followed when moving a digital work from the server to the client repositories, as may be performed in the currently preferred embodiment of the present invention.

OVERVIEW

A system for controlling use and distribution of digital works is disclosed. The present invention is directed to supporting commercial transactions involving digital works.

Herein the terms "digital work", "work" and "content" refer to any work that has been reduced to a digital representation. This would include any audio, video, text, or multimedia work and any accompanying interpreter (e.g. software) that may be required for recreating the work. The term composite work refers to a digital work comprised of a collection of other digital works. The term "usage rights" or "rights" is a term which refers to rights granted to a recipient of a digital work. Generally, these rights define how a digital work can be used and if it can be further distributed. Each usage right may have one or more specified conditions which must be satisfied before the right may be exercised.

Figure 1 is a high level flowchart omitting various details but which demonstrates the basic operation of the present

invention. Referring to Figure 1, a creator creates a digital work, step 101. The creator will then determine appropriate usage rights and fees, attach them to the digital work, and store them in Repository 1, step 102. The determination of appropriate usage rights and fees will depend on various economic factors. The digital work remains securely in Repository 1 until a request for access is received. The request for access begins with a session initiation by another repository. Here a Repository 2 initiates a session with Repository 1, step 103. As will be described in greater detail below, this session initiation includes steps which helps to insure that the respective repositories are trustworthy. Assuming that a session can be established, Repository 2 may then request access to the Digital Work for a stated purpose, step 104. The purpose may be, for example, to print the digital work or to obtain a copy of the digital work. The purpose will correspond to a specific usage right. In any event, Repository 1 checks the usage rights associated with the digital work to determine if the access to the digital work may be granted, step 105. The check of the usage rights essentially involves a determination of whether a right associated with the access request has been attached to the digital work and if all conditions associated with the right are satisfied. If the access is denied, repository 1 terminates the session with an error message, step 106. If access is granted, repository 1 transmits the digital work to repository 2, step 107. Once the digital work has been transmitted to repository 2, repository 1 and 2 each generate billing information for the access which is transmitted to a credit server, step 108. Such double billing reporting is done to insure against attempts to circumvent the billing process.

Figure 2 illustrates the basic interactions between repository types in the present invention. As will become apparent from Figure 2, the various repository types will serve different functions. It is fundamental that repositories will share a core set of functionality which will enable secure and trusted communications. Referring to Figure 2, a repository 201 represents the general instance of a repository. The repository 201 has two modes of operation; a server mode and a requester mode. When in the server mode, the repository will be receiving and processing access requests to digital works. When in the requester mode, the repository will be initiating requests to access digital works. Repository 201 is general in the sense that its primary purpose is as an exchange medium for digital works. During the course of operation, the repository 201 may communicate with a plurality of other repositories, namely authorization repository 202, rendering repository 203 and master repository 204. Communication between repositories occurs utilizing a repository transaction protocol 205.

Communication with an authorization repository 202 may occur when a digital work being accessed has a condition requiring an authorization. Conceptually, an authorization is a digital certificate such that possession of the certificate is required to gain access to the digital work. An authorization is itself a digital work that can be moved between repositories and subjected to fees and usage rights conditions. An authorization may be required by both repositories involved in an access to a digital work.

Communication with a rendering repository 203 occurs in connection with the rendering of a digital work. As will be described in greater detail below, a rendering repository is coupled with a rendering device (e.g. a printer device) to comprise a rendering system.

Communication with a master repository 205 occurs in connection with obtaining an identification certificate. Identification certificates are the means by which a repository is identified as "trustworthy". The use of identification certificates is described below with respect to the registration transaction.

Figure 3 illustrates the repository 201 coupled to a credit server 301. The credit server 301 is a device which accumulates billing information for the repository 201. The credit server 301 communicates with repository 201 via billing transactions 302 to record billing transactions. Billing transactions are reported to a billing clearinghouse 303 by the credit server 301 on a periodic basis. The credit server 301 communicates to the billing clearinghouse 303 via clearinghouse transactions 304. The clearinghouse transactions 304 enable a secure and encrypted transmission of information to the billing clearinghouse 303.

RENDERING SYSTEMS

A rendering system is generally defined as a system comprising a repository and a rendering device which can render a digital work into its desired form. Examples of a rendering system may be a computer system, a digital audio system, or a printer. A rendering system has the same security features as a repository. The coupling of a rendering repository with the rendering device may occur in a manner suitable for the type of rendering device.

Figure 4a illustrates a printer as an example of a rendering system. Referring to Figure 4, printer system 401 has contained therein a printer repository 402 and a print device 403. It should be noted that the the dashed line defining printer system 401 defines a secure system boundary. Communications within the boundary are assumed to be secure. Depending on the security level, the boundary also represents a barrier intended to provide physical integrity. The printer repository 402 is an instantiation of the rendering repository 205 of Figure 2. The printer repository 402 will in some instances contain an ephemeral copy of a digital work which remains until it is printed out by the print engine 403. In other instances, the printer repository 402 may contain digital works such as fonts, which will remain and can be billed based on use. This design assures that all communication lines between printers and printing devices are

encrypted, unless they are within a physically secure boundary. This design feature eliminates a potential "fault" point through which the digital work could be improperly obtained. The printer device 403 represents the printer components used to create the printed output.

Also illustrated in Figure 4a is the repository 404. The repository 404 is coupled to the printer repository 402. The repository 404 represents an external repository which contains digital works.

Figure 4b is an example of a computer system as a rendering system. A computer system may constitute a "multi-function" device since it may execute digital works (e.g. software programs) and display digital works (e.g. a digitized photograph). Logically, each rendering device can be viewed as having its own repository, although only one physical repository is needed. Referring to Figure 4b, a computer system 410 has contained therein a display/execution repository 411. The display/execution repository 411 is coupled to display device, 412 and execution device 413. The dashed box surrounding the computer system 410 represents a security boundary within which communications are assumed to be secure. The display/execution repository 411 is further coupled to a credit server 414 to report any fees to be billed for access to a digital work and a repository 415 for accessing digital works stored therein.

STRUCTURE OF DIGITAL WORKS

Usage rights are attached directly to digital works. Thus, it is important to understand the structure of a digital work. The structure of a digital work, in particular composite digital works, may be naturally organized into an acyclic structure such as a hierarchy. For example, a magazine has various articles and photographs which may have been created and are owned by different persons. Each of the articles and photographs may represent a node in a hierarchical structure. Consequently, controls, i.e. usage rights, may be placed on each node by the creator. By enabling control and fee billing to be associated with each node, a creator of a work can be assured that the rights and fees are not circumvented.

In the currently preferred embodiment, the file information for a digital work is divided into two files: a "contents" file and a "description tree" file. From the perspective of a repository, the "contents" file is a stream of addressable bytes whose format depends completely on the interpreter used to play, display or print the digital work. The description tree file makes it possible to examine the rights and fees for a work without reference to the content of the digital work. It should be noted that the term description tree as used herein refers to any type of acyclic structure used to represent the relationship between the various components of a digital work.

Figure 5 illustrates the layout of a contents file. Referring to Figure 5, a digital work is comprised of story A 510, advertisement 511, story B 512 and story C 513. It is assumed that the digital work is stored starting at a relative address of 0. Each of the parts of the digital work are stored linearly so that story A 510 is stored at approximately addresses 0-30,000, advertisement 511 at addresses 30,001-40,000, story B 512 at addresses 40,001-60,000 and story C 513 at addresses 60,001-85K. The detail of story A 510 is illustrated in Figure 6. Referring to Figure 6, the story A 510 is further broken down to show text 614 stored at address 0-1500, soldier photo 615 at addresses 1501-10,000, graphics 616 stored at addresses 10,001-25,000 and sidebar 617 stored address 25,001-30,000. Note that the data in the contents file may be compressed (for saving storage) or encrypted (for security).

From Figures 5 and 6 it is readily observed that a digital work can be represented by its component parts as a hierarchy. The description tree for a digital work is comprised of a set of related descriptor blocks (d-blocks). The contents of each d-block is described with respect to Figure 7. Referring to Figure 7, a d-block 700 includes an identifier 701 which is a unique identifier for the work in the repository, a starting address 702 providing the start address of the first byte of the work, a length 703 giving the number of bytes in the work, a rights portion 704 wherein the granted usage rights and their status data are maintained, a parent pointer 705 for pointing to a parent d-block and child pointers 706 for pointing to the child d-blocks. In the currently preferred embodiment, the identifier 701 has two parts. The first part is a unique number assigned to the repository upon manufacture. The second part is a unique number assigned to the work upon creation. The rights portion 704 will contain a data structure, such as a look-up table, wherein the various information associated with a right is maintained. The information required by the respective usage rights is described in more detail below. D-blocks form a strict hierarchy. The top d-block of a work has no parent; all other d-blocks have one parent. The relationship of usage rights between parent and child d-blocks and how conflicts are resolved is described below.

A special type of d-block is a "shell" d-block. A shell d-block adds no new content beyond the content of its parts. A shell d-block is used to add rights and fee information, typically by distributors of digital works.

Figure 8 illustrates a description tree for the digital work of Figure 5. Referring to Figure 8, a top d-block 820 for the digital work points to the various stories and advertisements contained therein. Here, the top d-block 820 points to d-block 821 (representing story A 510), d-block 822 (representing the advertisement 511), d-block 823 (representing story B 512) and and d-block 824 (representing story C 513).

The portion of the description tree for Story A 510 is illustrated in Figure 9. D-block 925 represents text 614, d-block 926 represents photo 615, d-block 927 represents graphics 616 by and d-block 928 represents sidebar 617.

The rights portion 704 of a descriptor block is further illustrated in Figure 10. Figure 10 illustrates a structure which is repeated in the rights portion 704 for each right. Referring to Figure 10, each right will have a right code field 1050 and status information field 1052. The right code field 1050 will contain a unique code assigned to a right. The status information field 1052 will contain information relating to the state of a right and the digital work. Such information is indicated below in Table 1. The rights as stored in the rights portion 704 may typically be in numerical order based on the right code.

TABLE 1

DIGITAL WORK STATE INFORMATION		
Property	**Value**	**Use**
Copies-in-Use	Number	A counter of the number of copies of a work that are in use. Incremented when another copy is used; decremented when use is completed.
Loan-Period	Time-Units	Indicator of the maximum number of time-units that a document can be loaned out
Loaner-Copy	Boolean	Indicator that the current work is a loaned out copy of an authorized digital work.
Remaining-Time	Time-Units	Indicator of the remaining time of use on a metered document right.
Document-Descr	String	A string containing various identifying information about a document. The exact format of this is not specified, but it can include information such as a publisher name, author name, ISBN number, and so on.
Revenue-Owner	RO-Descr	A handle identifying a revenue owner for a digital work. This is used for reporting usage fees.
Publication-Date	Date-Descr	The date that the digital work was published.
History-list	History-Rec	A list of events recording the repostories and dates for operations that copy, transfer, backup, or restore a digital work.

The approach for representing digtal works by separating description data from content assumes that parts of a file are contiguous but takes no position on the actual representation of content. In particular, it is neutral to the question of whether content representation may take an object oriented approach. It would be natural to represent content as objects. In principle, it may be convenient to have content objects that include the billing structure and rights information that is represented in the d-blocks. Such variations in the design of the representation are possible and are viable alternatives but may introduce processing overhead, e.g. the interpretation of the objects.

Digital works are stored in a repository as part of a hierarchical file system. Folders (also termed directories and sub-directories) contain the digital works as well as other folders. Digital works and folders in a folder are ordered in alphabetical order. The digital works are typed to reflect how the files are used. Usage rights can be attached to folders so that the folder itself is treated as a digital work. Access to the folder would then be handled in the same fashion as any other digital work As will be described in more detail below, the contents of the folder are subject to their own rights. Moreover, file management rights may be attached to the folder which define how folder contents can be managed.

ATTACHING USAGE RIGHTS TO A DIGITAL WORK

It is fundamental to the present invention that the usage rights are treated as part of the digital work. As the digital work is distributed, the scope of the granted usage rights will remain the same or may be narrowed. For example, when a digital work is transferred from a document server to a repository, the usage rights may include the right to loan a copy for a predetermined period of time (called the original rights). When the repository loans out a copy of the digital work, the usage rights in the loaner copy (called the next set of rights) could be set to prohibit any further rights to loan out the copy. The basic idea is that one cannot grant more rights than they have.

The attachment of usage rights into a digital work may occur in a variety of ways. If the usage rights will be the same for an entire digital work, they could be attached when the digital work is processed for deposit in the digital work server. In the case of a digital work having different usage rights for the various components, this can be done as the digital work is being created. An authoring tool or digital work assembling tool could be utilized which provides for an automated process of attaching the usage rights.

As will be described below, when a digital work is copied, transferred or loaned, a "next set of rights" can be specified. The"next set of rights" will be attached to the digital work as it is transported.

Resolving Conflicting Rights

Because each part of a digital work may have its own usage rights, there will be instances where the rights of a "contained part" are different from its parent or container part. As a result, conflict rules must be established to dictate when and how a right may be exercised. The hierarchical structure of a digital work facilitates the enforcement of such rules. A "strict" rule would be as follows: a right for a part in a digital work is sanctioned if and only if it is sanctioned for the part, for ancestor d-blocks containing the part and for all descendent d-blocks. By sanctioned, it is meant that (1) each of the respective parts must have the right, and (2) any conditions for exercising the right are satisfied.

It also possible to implement the present invention using a more lenient rule. In the more lenient rule, access to the part may be enabled to the descendent parts which have the right, but access is denied to the descendents which do not.

An example of applying both the strict rule and lenient is illustrated with reference to Figure 11. Referring to Figure 11, a root d-block 1101 has child d-blocks 1102-1105. In this case, root d-block represents a magazine, and each of the child d-blocks 1102-1105 represent articles in the magazine. Suppose that a request is made to PRINT the digital work represented by root d-block 1101 wherein the strict rule is followed. The rights for the root d-block 1101 and child d-blocks 1102-1105 are then examined. Root d-block 1101 and child d-blocks 1102 and 1105 have been granted PRINT rights. Child d-block 1103 has not been granted PRINT rights and child d-block 1104 has PRINT rights conditioned on payment of a usage fee.

Under the strict rule the PRINT right cannot be exercised because the child d-block does not have the PRINT right. Under the lenient rule, the result would be different. The digital works represented by child d-blocks 1102 and 1105 could be printed and the digital work represented by d-block 1104 could be printed so long as the usage fee is paid. Only the digital work represented by d-block 1103 could not be printed. This same result would be accomplished under the strict rule if the requests were directed to each of the individual digital works.

The present invention supports various combinations of allowing and disallowing access. Moreover, as will be described below, the usage rights grammar permits the owner of a digital work to specify if constraints may be imposed on the work by a container part. The manner in which digital works may be sanctioned because of usage rights conflicts would be implementation specific and would depend on the nature of the digital works.

REPOSITORIES

In the description of Figure 2, it was indicated that repositories come in various forms. All repositories provide a core set of services for the transmission of digital works. The manner in which digital works are exchanged is the basis for all transaction between repositories. The various repository types differ in the ultimate functions that they perform. Repositories may be devices themselves, or they may be incorporated into other systems. An example is the rendering repository 203 of Figure 2.

A repository will have associated with it a repository identifier. Typically, the repository identifier would be a unique number assigned to the repository at the time of manufacture. Each repository will also be classified as being in a particular security class. Certain communications and transactions may be conditioned on a repository being in a particular security class. The various security classes are described in greater detail below.

As a prerequisite to operation, a repository will require possession of an identification certificate. Identification certificates are encrypted to prevent forgery and are issued by a Master repository. A master repository plays the role of an authorization agent to enable repositories to receive digital works. Identification certificates must be updated on a periodic basis. Identification certificates are described in greater detail below with respect to the registration transaction.

A repository has both a hardware and functional embodiment. The functional embodiment is typically software executing on the hardware embodiment. Alternatively, the functional embodiment may be embedded in the hardware embodiment such as an Application Specific Integrated Circuit (ASIC) chip.

The hardware embodiment of a repository will be enclosed in a secure housing which if compromised, may cause the repository to be disabled. The basic components of the hardware embodiment of a repository are described with reference to Figure 12. Referring to Figure 12, a repository is comprised of a processing means 1200, storage system 1207, clock 1205 and external interface 1206. The processing means 1200 is comprised of a processor element 1201 and processor memory 1202. The processing means 1201 provides controller, repository transaction and usage rights transaction functions for the repository. Various functions in the operation of the repository such as decryption and/or decompression of digital works and transaction messages are also performed by the processing means 1200. The processor element 1201 may be a microprocessor or other suitable computing component. The processor memory 1202 would typically be further comprised of Read Only Memories (ROM) and Random Access Memories (RAM). Such memories would contain the software instructions utilized by the processor element 1201 in performing the functions of the repository.

The storage system 1207 is further comprised of descriptor storage 1203 and content storage 1204. The description tree storage 1203 will store the description tree for the digital work and the content storage will store the associated content. The description tree storage 1203 and content storage 1204 need not be of the same type of storage medium, nor are they necessarily on the same physical device. So for example, the descriptor storage 1203 may be stored on a solid state storage (for rapid retrieval of the description tree information), while the content storage 1204 may be on a high capacity storage such as an optical disk.

The clock 1205 is used to time-stamp various time based conditions for usage rights or for metering usage fees which may be associated with the digital works. The clock 1205 will have an uninterruptable power supply, e.g. a battery, in order to maintain the integrity of the time-stamps. The external interface means 1206 provides for the signal connection to other repositories and to a credit server. The external interface means 1206 provides for the exchange of signals via such standard interfaces such as RS-232 or Personal Computer Manufacturers Card Industry Association (PCMCIA) standards, or FDDI. The external interface means 1206 may also provide network connectivity.

The functional embodiment of a repository is described with reference to Figure 13. Referring to Figure 13, the functional embodiment is comprised of an operating system 1301, core repository services 1302, usage transaction handlers 1303, repository specific functions, 1304 and a user interface 1305. The operating system 1301 is specific to the repository and would typically depend on the type of processor being used. The operating system 1301 would also provide the basic services for controlling and interfacing between the basic components of the repository.

The core repository services 1302 comprise a set of functions required by each and every repository. The core repository services 1302 include the session initiation transactions which are defined in greater detail below. This set of services also includes a generic ticket agent which is used to "punch" a digital ticket and a generic authorization server for processing authorization specifications. Digital tickets and authorizations are specific mechanisms for controlling the distribution and use of digital works and are described in more detail below. Note that coupled to the core repository services are a plurality of identification certificates 1306. The identification certificates 1306 are required to enable the use of the repository.

The usage transactions handlers 1303 comprise functionality for processing access requests to digital works and for billing fees based on access. The usage transactions supported will be different for each repository type. For example, it may not be necessary for some repositories to handle access requests for digital works.

The repository specific functionality 1304 comprises functionality that is unique to a repository. For example, the master repository has special functionality for issuing digital certificates and maintaining encryption keys. The repository specific functionality 1304 would include the user interface implementation for the repository.

Repository Security Classes

For some digital works the losses caused by any individual instance of unauthorized copying is insignificant and the chief economic concern lies in assuring the convenience of access and low-overhead billing. In such cases, simple and inexpensive handheld repositories and network-based workstations may be suitable repositories, even though the measures and guarantees of security are modest.

At the other extreme, some digital works such as a digital copy of a first run movie or a bearer bond or stock certificate would be of very high value so that it is prudent to employ caution and fairly elaborate security measures to ensure that they are not copied or forged. A repository suitable for holding such a digital work could have elaborate measures for ensuring physical integrity and for verifying authorization before use.

By arranging a universal protocol, all kinds of repositories can communicate with each other in principle. However, creators of some works will want to specify that their works will only be transferred to repositories whose level of security is high enough. For this reason, document repositories have a ranking system for classes and levels of security. The security classes in the currently preferred embodiment are described in Table 2.

TABLE 2

REPOSITORY SECURITY LEVELS	
Level	Description of Security
0	Open system. Document transmission is unencrypted. No digital certificate is required for identification. The security of the system depends mostly on user honesty, since only modest knowledge may be needed to circumvent the security measures. The repository has no provisions for preventing unauthorized programs from running and accessing or copying files. The system does not prevent the use of removable storage and does not encrypt stored files.
1	Minimal security. Like the previous class except that stored files are minimally encrypted, including ones on removable storage.

TABLE 2 (continued)

REPOSITORY SECURITY LEVELS	
Level	Description of Security
2	Basic security. Like the previous class except that special tools and knowledge are required to compromise the programming, the contents of the repository, or the state of the clock. All digital communications are encrypted. A digital certificate is provided as identification. Medium level encryption is used. Repository identification number is unforgeable.
3	General security. Like the previous class plus the requirement of special tools are needed to compromise the physical integrity of the repository and that modest encryption is used on all transmissions. Password protection is required to use the local user interface. The digital clock system cannot be reset without authorization. No works would be stored on removable storage. When executing works as programs, it runs them in their own address space and does not give them direct access to any file storage or other memory containing system code or works. They can access works only through the transmission transaction protocol.
4	Like the previous class except that high level encryption is used on all communications. Sensors are used to record attempts at physical and electronic tampering. After such tampering, the repository will not perform other transactions until it has reported such tampering to a designated server.
5	Like the previous class except that if the physical or digital attempts at tampering exceed some preset thresholds that threaten the physical integrity of the repository or the integrity of digital and cryptographic barriers, then the repository will save only document description records of history but will erase or destroy any digital identifiers that could be misused if released to an unscrupulous party. It also modifies any certificates of authenticity to indicate that the physical system has been compromised. It also erases the contents of designated documents.
6	Like the previous class except that the repository will attempt wireless communication to report tampering and will employ noisy alarms.
10	This would correspond to a very high level of security. This server would maintain constant communications to remote security systems reporting transactions, sensor readings, and attempts to circumvent security.

The characterization of security levels described in Table 2 is not intended to be fixed. More important is the idea of having different security levels for different repositories. It is anticipated that new security classes and requirements will evolve according to social situations and changes in technology.

Repository User Interface

A user interface is broadly defined as the mechanism by which a user interacts with a repository in order to invoke transactions to gain access to a digital work, or exercise usage rights. As described above, a repository may be embodied in various forms. The user interface for a repository will differ depending on the particular embodiment. The user interface may be a graphical user interface having icons representing the digital works and the various transactions that may be performed. The user interface may be a generated dialog in which a user is prompted for information.

The user interface itself need not be part of the repository. As a repository may be embedded in some other device, the user interface may merely be a part of the device in which the repository is embedded. For example, the repository could be embedded in a "card" that is inserted into an available slot in a computer system. The user interface may be a combination of a display, keyboard, cursor control device and software executing on the computer system.

At a minimum, the user interface must permit a user to input information such as access requests and alpha numeric data and provide feedback as to transaction status. The user interface will then cause the repository to initiate the suitable transactions to service the request. Other facets of a particular user interface will depend on the functionality that a repository will provide.

CREDIT SERVERS

In the present invention, fees may be associated with the exercise of a right. The requirement for payment of fees is described with each version of a usage right in the usage rights language. The recording and reporting of such fees is performed by the credit server. One of the capabilities enabled by associating fees with rights is the possibility of

supporting a wide range of charging models. The simplest model, used by conventional software, is that there is a single fee at the time of purchase, after which the purchaser obtains unlimited rights to use the work as often and for as long as he or she wants. Alternative models, include metered use and variable fees. A single work can have different fees for different uses. For example, viewing a photograph on a display could have different fees than making a hardcopy or including it in a newly created work. A key to these alternative charging models is to have a low overhead means of establishing fees and accounting for credit on these transactions.

A credit server is a computational system that reliably authorizes and records these transactions so that fees are billed and paid. The credit server reports fees to a billing clearinghouse. The billing clearinghouse manages the financial transactions as they occur. As a result, bills may be generated and accounts reconciled. Preferably, the credit server would store the fee transactions and periodically communicate via a network with the billing clearinghouse for reconciliation. In such an embodiment, communications with the billing clearinghouse would be encrypted for integrity and security reasons. In another embodiment, the credit server acts as a "debit card" where transactions occur in "real-time" against a user account.

A credit server is comprised of memory, a processing means, a clock, and interface means for coupling to a repository and a financial institution (e.g. a modem). The credit server will also need to have security and authentication functionality. These elements are essentially the same elements as those of a repository. Thus, a single device can be both a repository and a credit server, provided that it has the appropriate processing elements for carrying out the corresponding functions and protocols. Typically, however, a credit server would be a card-sized system in the possession of the owner of the credit. The credit server is coupled to a repository and would interact via financial transactions as described below. Interactions with a financial institution may occur via protocols established by the financial institutions themselves.

In the currently preferred embodiment credit servers associated with both the server and the repository report the financial transaction to the billing clearinghouse. For example, when a digital work is copied by one repository to another for a fee, credit servers coupled to each of the repositories will report the transaction to the billing clearinghouse. This is desirable in that it insures that a transaction will be accounted for in the event of some break in the communication between a credit server and the billing clearinghouse. However, some implementations may embody only a single credit server reporting the transaction to minimize transaction processing at the risk of losing some transactions.

USAGE RIGHTS LANGUAGE

The present invention uses statements in a high level "usage rights language" to define rights associated with digital works and their parts. Usage rights statements are interpreted by repositories and are used to determine what transactions can be successfully carried out for a digital work and also to determine parameters for those transactions. For example, sentences in the language determine whether a given digital work can be copied, when and how it can be used, and what fees (if any) are to be charged for that use. Once the usage rights statements are generated, they are encoded in a suitable form for accessing during the processing of transactions.

Defining usage rights in terms of a language in combination with the hierarchical representation of a digital work enables the support of a wide variety of distribution and fee schemes. An example is the ability to attach multiple versions of a right to a work. So a creator may attach a PRINT right to make 5 copies for $10.00 and a PRINT right to make unlimited copies for $100.00. A purchaser may then choose which option best fits his needs. Another example is that rights and fees are additive. So in the case of a composite work, the rights and fees of each of the components works is used in determining the rights and fees for the work as a whole.

The basic contents of a right are illustrated in Figure 14. Referring to Figure 14, a right 1450 has a transactional component 1451 and a specifications component 1452. A right 1450 has a label (e.g. COPY or PRINT) which indicates the use or distribution privileges that are embodied by the right. The transactional component 1451 corresponds to a particular way in which a digital work may be used or distributed. The transactional component 1451 is typically embodied in software instructions in a repository which implement the use or distribution privileges for the right. The specifications components 1452 are used to specify conditions which must be satisfied prior to the right being exercised or to designate various transaction related parameters. In the currently preferred embodiment, these specifications include copy count 1453, Fees and Incentives 1454, Time 1455, Access and Security 1456 and Control 1457. Each of these specifications will be described in greater detail below with respect to the language grammar elements.

The usage rights language is based on the grammar described below. A grammar is a convenient means for defining valid sequence of symbols for a language. In describing the grammar the notation "[a | b | c] is used to indicate distinct choices among alternatives. In this example, a sentence can have either an "a", "b" or "c". It must include exactly one of them. The braces { } are used to indicate optional items. Note that brackets, bars and braces are used to describe the language of usage rights sentences but do not appear in actual sentences in the language.

In contrast, parentheses are part of the usage rights language. Parentheses are used to group items together in lists. The notation (x*) is used to indicate a variable length list, that is, a list containing one or more items of type x.

The notation (x)* is used to indicate a variable number of lists containing x.

Keywords in the grammar are words followed by colons. Keywords are a common and very special case in the language. They are often used to indicate a single value, typically an identifier. In many cases, the keyword and the parameter are entirely optional. When a keyword is given, it often takes a single identifier as its value. In some cases, the keyword takes a list of identifiers.

In the usage rights language, time is specified in an hours:minutes:seconds (or hh:mm:ss) representation. Time zone indicators, e.g. PDT for Pacific Daylight Time, may also be specified. Dates are represented as year/ month/day (or YYYY/MMM/DD). Note that these time and date representations may specify moments in time or units of time Money units are specified in terms of dollars.

Finally, in the usage rights language, various "things" will need to interact with each other. For example, an instance of a usage right may specify a bank account, a digital ticket, etc.. Such things need to be identified and are specified herein using the suffix "-ID."

The Usage Rights Grammar is listed in its entirety in Figure 15 and is described below.

Grammar element 1501 **"Digital Work Rights: = (Rights*)"** define the digital work rights as a set of rights. The set of rights attached to a digital work define how that digital work may be transferred, used, performed or played. A set of rights will attach to the entire digital work and in the case of compound digital works, each of the components of the digital work. The usage rights of components of a digital may be different.

Grammar element 1502 **"Right : = (Right-Code {Copy-Count} {Control-Spec} {Time-Spec} {Access-Spec} {Fee-Spec})"** enumerates the content of a right. Each usage right must specify a right code. Each right may also optionally specify conditions which must be satisfied before the right can be exercised. These conditions are copy count, control, time, access and fee conditions. In the currently preferred embodiment, for the optional elements, the following defaults apply: copy count equals 1, no time limit on the use of the right, no access tests or a security level required to use the right and no fee is required. These conditions will each be described in greater detail below.

It is important to note that a digital work may have multiple versions of a right, each having the same right code. The multiple version would provide alternative conditions and fees for accessing the digital work.

Grammar element 1503 **"Right-Code : = Render-Code I Transport-Code I File-Management-CodeI Derivative-Works- Code Configuration-Code"** distinguishes each of the specific rights into a particular right type (although each right is identified by distinct right codes). In this way, the grammar provides a catalog of possible rights that can be associated with parts of digital works. In the following, rights are divided into categories for convenience in describing them.

Grammar element 1504 **"Render-Code : = [Play: {Player: Player-ID} I Print: {Printer: Printer-ID}]"** lists a category of rights all involving the making of ephemeral, transitory, or non-digital copies of the digital work. After use the copies are erased.

- Play — A process of rendering or performing a digital work on some processor. This includes such things as playing digital movies, playing digital music, playing a video game, running a computer program, or displaying a document on a display.
- Print — To render the work in a medium that is not further protected by usage rights, such as printing on paper.

Grammar element 1505 **"Transport-Code : = [Copy I Transfer I Loan {Remaining-Rights: Next-Set-of-Rights}] {(Next-Copy-Rights: Next-Set of Rights)}"** lists a category of rights involving the making of persistent, usable copies of the digital work on other repositories. The optional Next-Copy-Rights determine the rights on the work after it is transported. If this is not specified, then the rights on the transported copy are the same as on the original. The optional Remaining-Rights specify the rights that remain with a digital work when it is loaned out. If this is not specified, then the default is that no rights can be exercised when it is loaned out.

- Copy — Make a new copy of a work
- Transfer — Moving a work from one repository to another.
- Loan — Temporarily loaning a copy to another repository for a specified period of time.

Grammar element 1506 **"File-Management-Code : = Backup {Back-Up-Copy-Rights: Next-Set -of Rights}I Restore I Delete I Folder I Directory {Name:Hide-Local I Hide - Remote}{Parts:Hide-Local I Hide-Remote}"** lists a category of rights involving operations for file management, such as the making of backup copies to protect the copy owner against catastrophic equipment failure.

Many software licenses and also copyright law give a copy owner the right to make backup copies to protect against catastrophic failure of equipment. However, the making of uncontrolled backup copies is inherently at odds with the ability to control usage, since an uncontrolled backup copy can be kept and then restored even after the authorized copy was sold.

The File management rights enable the making and restoring of backup copies in a way that respects usage rights, honoring the requirements of both the copy owner and the rights grantor and revenue owner. Backup copies of work descriptions (including usage rights and fee data) can be sent under appropriate protocol and usage rights control to other document repositories of sufficiently high security. Further rights permit organization of digital works into folders which themselves are treated as digital works and whose contents may be "hidden" from a party seeking to determine the contents of a repository.

- Backup To make a backup copy of a digital work as protection against media failure.
- Restore To restore a backup copy of a digital work.
- Delete To delete or erase a copy of a digital work.
- Folder To create and name folders, and to move files and folders between folders.
- Directory To hide a folder or its contents.

Grammar element 1507 **"Derivative-Works-Code: [Extract I Embed I Edit {Process: Process-ID}] {Next-Copy-Rights : Next-Set-of Rights}"** lists a category of rights involving the use of a digital work to create new works.

- Extract To remove a portion of a work, for the purposes of creating a new work.
- Embed To include a work in an existing work.
- Edit To alter a digital work by copying, selecting and modifying portions of an existing digital work.

Grammar element 1508 **"Configuration-Code : = Install I Uninstall"** lists a category of rights for installing and uninstalling software on a repository (typically a rendering repository.) This would typically occur for the installation of a new type of player within the rendering repository.

- Install: To install new software on a repository.
- Uninstall: To remove existing software from a repository.

Grammar element 1509 **"Next-Set-of-Rights : = {(Add : Set-Of-Rights)} {(Delete: Set-Of-Rights)} {(Replace: Set-Of-Rights)} {(Keep: Set-Of-Rights)}"** defines how rights are carried forward for a copy of a digital work. If the Next-Copy-Rights is not specified, the rights for the next copy are the same as those of the current copy. Otherwise, the set of rights for the next copy can be specified. Versions of rights after Add: are added to the current set of rights. Rights after Delete: are deleted from the current set of rights. If only right codes are listed after Delete:, then all versions of rights with those codes are deleted. Versions of rights after Replace: subsume all versions of rights of the same type in the current set of rights.

If Remaining-Rights is not specified, then there are no rights for the original after all Loan copies are loaned out. If Remaining-Rights is specified, then the Keep: token can be used to simplify the expression of what rights to keep behind. A list of right codes following keep means that all of the versions of those listed rights are kept in the remaining copy. This specification can be overridden by subsequent Delete: or Replace: specifications.

Copy Count Specification

For various transactions, it may be desirable to provide some limit as to the number of "copies" of the work which may be exercised simultaneously for the right. For example, it may be desirable to limit the number of copies of a digital work that may be loaned out at a time or viewed at a time.

Grammar element 1510 **"Copy-Count : = (Copies: positive-integer I 0 I unlimited)"** provides a condition which defines the number of "copies" of a work subject to the right . A copy count can be 0, a fixed number, or unlimited. The copy-count is associated with each right, as opposed to there being just a single copy-count for the digital work. The Copy-Count for a right is decremented each time that a right is exercised. When the Copy-Count equals zero, the right can no longer be exercised. If the Copy-Count is not specified, the default is one.

Control Specification

Rights and fees depend in general on rights granted by the creator as well as further restrictions imposed by later distributors. Control specifications deal with interactions between the creators and their distributors governing the imposition of further restrictions and fees. For example, a distributor of a digital work may not want an end consumer of a digital work to add fees or otherwise profit by commercially exploiting the purchased digital work.

Grammar element 1511 **"Control-Spec : = (Control: {Restrictable I Unrestrictable} {Unchargeable I Chargeable})"** provides a condition to specify the effect of usage rights and fees of parents on the exercise of the right. A

digital work is restrictable if higher level d-blocks can impose further restrictions (time specifications and access specifications) on the right. It is unrestrictable if no further restrictions can be imposed. The default setting is restrictable. A right is unchargeable if no more fees can be imposed on the use of the right. It is chargeable if more fees can be imposed. The default is chargeable.

Time Specification

It is often desirable to assign a start date or specify some duration as to when a right may be exercised. Grammar element 1512 **"Time-Spec : = ({Fixed-Interval | Sliding-Interval | Meter-Time} Until: Expiration-Date)"** provides for specification of time conditions on the exercise of a right. Rights may be granted for a specified time. Different kinds of time specifications are appropriate for different kinds of rights. Some rights may be exercised during a fixed and predetermined duration. Some rights may be exercised for an interval that starts the first time that the right is invoked by some transaction. Some rights may be exercised or are charged according to some kind of metered time, which may be split into separate intervals. For example, a right to view a picture for an hour might be split into six ten minute viewings or four fifteen minute viewings or twenty three minute viewings.

The terms "time" and "date" are used synonymously to refer to a moment in time. There are several kinds of time specifications. Each specification represents some limitation on the times over which the usage right applies. The Expiration-Date specifies the moment at which the usage right ends. For example, if the Expiration-Date is "Jan 1, 1995," then the right ends at the first moment of 1995. If the Expiration-Date is specified as "forever", then the rights are interpreted as continuing without end. If only an expiration date is given, then the right can be exercised as often as desired until the expiration date.

Grammar element 1513 **"Fixed-Interval := From: Start-Time"** is used to define a predetermined interval that runs from the start time to the expiration date.

Grammar element 1514 **"Sliding-Interval : = Interval: Use-Duration"** is used to define an indeterminate (or "open") start time. It sets limits on a continuous period of time over which the contents are accessible. The period starts on the first access and ends after the duration has passed or the expiration date is reached, whichever comes first. For example, if the right gives 10 hours of continuous access, the use-duration would begin when the first access was made and end 10 hours later.

Grammar element 1515 **"Meter-Time: = Time-Remaining: Remaining-Use"** is used to define a "meter time," that is, a measure of the time that the right is actually exercised. It differs from the Sliding-Interval specification in that the time that the digital work is in use need not be continuous. For example, if the rights guarantee three days of access, those days could be spread out over a month. With this specification, the rights can be exercised until the meter time is exhausted or the expiration date is reached, whichever comes first.

Remaining-Use: = Time-Unit
Start-Time: = Time-Unit
Use-Duration: = Time-Unit
All of the time specifications include time-unit specifications in their ultimate instantiation.

Security Class and Authorization Specification

The present invention provides for various security mechanisms to be introduced into a distribution or use scheme. Grammar element 1516 **"Access-Spec : = ({SC: Security-Class} {Authorization: Authorization-ID*} {Other-Authorization: Authorization-ID*} {Ticket: Ticket-ID})"** provides a means for restricting access and transmission. Access specifications can specify a required security class for a repository to exercise a right or a required authorization test that must be satisfied.

The keyword **"SC:"** is used to specify a minimum security level for the repositories involved in the access. If **"SC:"** is not specified, the lowest security level is acceptable.

The optional **"Authorization:"** keyword is used to specify required authorizations on the same repository as the work. The optional **"Other-Authorization:"** keyword is used to specify required authorizations on the other repository in the transaction.

The optional **"Ticket:"** keyword specifies the identity of a ticket required for the transaction. A transaction involving digital tickets must locate an appropriate digital ticket agent who can "punch" or otherwise validate the ticket before the transaction can proceed. Tickets are described in greater detail below.

In a transaction involving a repository and a document server, some usage rights may require that the repository have a particular authorization, that the server have some authorization, or that both repositories have (possibly different) authorizations. Authorizations themselves are digital works (hereinafter referred to as an authorization object) that can be moved between repositories in the same manner as other digital works. Their copying and transferring is

subject to the same rights and fees as other digital works. A repository is said to have an authorization if that authorization object is contained within the repository.

In some cases, an authorization may be required from a source other than the document server and repository. An authorization object referenced by an Authorization-ID can contain digital address information to be used to set up a communications link between a repository and the authorization source. These are analogous to phone numbers. For such access tests, the communication would need to be established and authorization obtained before the right could be exercised.

For one-time usage rights, a variant on this scheme is to have a digital ticket. A ticket is presented to a digital ticket agent, whose type is specified on the ticket. In the simplest case, a certified generic ticket agent, available on all repositories, is available to "punch" the ticket. In other cases, the ticket may contain addressing information for locating a "special" ticket agent. Once a ticket has been punched, it cannot be used again for the same kind of transaction (unless it is unpunched or refreshed in the manner described below.) Punching includes marking the ticket with a timestamp of the date and time it was used. Tickets are digital works and can be copied or transferred between repositories according to their usage rights.

In the currently preferred embodiment, a "punched" ticket becomes "unpunched" or "refreshed" when it is copied or extracted. The Copy and Extract operations save the date and time as a property of the digital ticket. When a ticket agent is given a ticket, it can simply check whether the digital copy was made after the last time that it was punched. Of course, the digital ticket must have the copy or extract usage rights attached thereto.

The capability to unpunch a ticket is inportant in the following cases:

- A digital work is circulated at low cost with a limitation that it can be used only once.
- A digital work is circulated with a ticket that can be used once to give discounts on purchases of other works.
- A digital work is circulated with a ticket (included in the purchase price and possibly embedded in the work) that can be used for a future upgrade.

In each of these cases, if a paid copy is made of the digital work (including the ticket) the new owner would expect to get a fresh (unpunched) ticket, whether the copy seller has used the work or not. In contrast, loaning a work or simply transferring it to another repository should not revitalize the ticket.

Usage Fees and Incentives Specification

The billing for use of a digital work is fundamental to a commercial distribution system. Grammar Element 1517 **"Fee-Spec: = {Scheduled-Discount} Regular-Fee-Spec | Scheduled-Fee-Spec | Markup-Spec"** provides a range of options for billing for the use of digital works.

A key feature of this approach is the development of low-overhead billing for transactions in potentially small amounts. Thus, it becomes feasible to collect fees of only a few cents each for thousands of transactions.

The grammar differentiates between uses where the charge is per use from those where it is metered by the time unit. Transactions can support fees that the user pays for using a digital work as well as incentives paid by the right grantor to users to induce them to use or distribute the digital work.

The optional scheduled discount refers to the rest of the fee specification--discounting it by a percentage over time. If it is not specified, then there is no scheduled discount. Regular fee specifications are constant over time. Scheduled fee specifications give a schedule of dates over which the fee specifications change. Markup specifications are used in d-blocks for adding a percentage to the fees already being charged.

Grammar Element 1518 **"Scheduled-Discount: = (Scheduled-Discount: (Time-Spec Percentage)*)"** A Scheduled-Discount is a essentially a scheduled modifier of any other fee specification for this version of the right of the digital work. (It does not refer to children or parent digital works or to other versions of rights.). It is a list of pairs of times and percentages. The most recent time in the list that has not yet passed at the time of the transaction is the one in effect. The percentage gives the discount percentage. For example, the number 10 refers to a 10% discount.

Grammar Element 1519 **"Regular-Fee-Spec : = ({Fee: | Incentive: } [Per-Use-Spec | Metered-Rate-Spec | Best-Price-Spec | Call-For-Price-Spec] {Min: Money-Unit Per: Time-Spec)(Max: Money-Unit Per: Time-Spec} To: Account-ID)"** provides for several kinds of fee specifications.

Fees are paid by the copy-owner/user to the revenue-owner if Fee: is specified. Incentives are paid by the revenue-owner to the user if Incentive: is specified. If the Min: specification is given, then there is a minimum fee to be charged per time-spec unit for its use. If the Max: specification is given, then there is a maximum fee to be charged per time-spec for its use. When Fee: is specified, Account-ID identifies the account to which the fee is to be paid. When Incentive: is specified, Account-ID identifies the account from which the fee is to be paid.

Grammar element 1520 **"Per-Use-Spec: = Per-Use: Money-unit"** defines a simple fee to be paid every time the right is exercised, regardless of how much time the transaction takes.

Grammar element 1521 **"Metered-Rate-Spec : = Metered: Money-Unit Per: Time-Spec"** defines a metered-rate fee paid according to how long the right is exercised. Thus, the time it takes to complete the transaction determines the fee.

Grammar element 1522 **"Best-Price-Spec := Best-Price: Money-unit Max: Money-unit"** is used to specify a best-price that is determined when the account is settled. This specification is to accommodate special deals, rebates, and pricing that depends on information that is not available to the repository. All fee specifications can be combined with tickets or authorizations that could indicate that the consumer is a wholesaler or that he is a preferred customer, or that the seller be authorized in some way. The amount of money in the Max: field is the maximum amount that the use will cost. This is the amount that is tentatively debited from the credit server. However, when the transaction is ultimately reconciled, any excess amount will be returned to the consumer in a separate transaction.

Grammar element 1523 **"Call-For-Price-Spec:= Call-For-Price"** is similar to a **"Best-Price-Spec"** in that it is intended to accommodate cases where prices are dynamic. **A Call-For-Price Spec** requires a communication with a dealer to determine the price. This option cannot be exercised if the repository cannot communicate with a dealer at the time that the right is exercised. It is based on a secure transaction whereby the dealer names a price to exercise the right and passes along a deal certificate which is referenced or included in the billing process.

Grammar element 1524 **"Scheduled-Fee-Spec: = (Schedule: (Time-Spec Regular-Fee-Spec)*)"** is used to provide a schedule of dates over which the fee specifications change. The fee specification with the most recent date not in the future is the one that is in effect. This is similar to but more general than the scheduled discount. It is more general, because it provides a means to vary the fee agreement for each time period.

Grammar element 1525 **"Markup-Spec: = Markup: percentage To: Account-ID"** is provided for adding a percentage to the fees already being charged. For example, a 5% markup means that a fee of 5% of cumulative fee so far will be allocated to the distributor. A markup specification can be applied to all of the other kinds of fee specifications. It is typically used in a shell provided by a distributor. It refers to fees associated with d-blocks that are parts of the current d-block. This might be a convenient specification for use in taxes, or in distributor overhead.

REPOSITORY TRANSACTIONS

When a user requests access to a digital work, the repository will initiate various transactions. The combination of transactions invoked will depend on the specifications assigned for a usage right. There are three basic types of transactions, Session Initiation Transactions, Financial Transactions and Usage Transactions. Generally, session initiation transactions are initiated first to establish a valid session. When a valid session is established, transactions corresponding to the various usage rights are invoked. Finally, request specific transactions are performed.

Transactions occur between two repositories (one acting as a server), between a repository and a document playback platform (e.g. for executing or viewing), between a repository and a credit server or between a repository and an authorization server. When transactions occur between more than one repository, it is assumed that there is a reliable communication channel between the repositories. For example, this could be a TCP/IP channel or any other commercially available channel that has built-in capabilities for detecting and correcting transmission errors. However, it is not assumed that the communication channel is secure. Provisions for security and privacy are part of the requirements for specifying and implementing repositories and thus form the need for various transactions.

Message Transmission

Transactions require that there be some communication between repositories. Communication between repositories occurs in units termed as messages. Because the communication line is assumed to be unsecure, all communications with repositories that are above the lowest security class are encrypted utilizing a public key encryption technique. Public key encryption is a well known technique in the encryption arts. The term key refers to a numeric code that is used with encryption and decryption algorithms. Keys come in pairs, where "writing keys" are used to encrypt data and "checking keys" are used to decrypt data. Both writing and checking keys may be public or private. Public keys are those that are distributed to others. Private keys are maintained in confidence.

Key management and security is instrumental in the success of a public key encryption system. In the currently preferred embodiment, one or more master repositories maintain the keys and create the identification certificates used by the repositories.

When a sending repository transmits a message to a receiving repository, the sending repository encrypts all of its data using the public writing key of the receiving repository. The sending repository includes its name, the name of the receiving repository, a session identifier such as a nonce (described below), and a message counter in each message.

In this way, the communication can only be read (to a high probability) by the receiving repository, which holds the private checking key for decryption. The auxiliary data is used to guard against various replay attacks to security. If

messages ever arrive with the wrong counter or an old nonce, the repositories can assume that someone is interfering with communication and the transaction terminated.

The respective public keys for the repositories to be used for encryption are obtained in the registration transaction described below.

Session Initiation Transactions

A usage transaction is carried out in a session between repositories. For usage transactions involving more than one repository, or for financial transactions between a repository and a credit server, a registration transaction is performed. A second transaction termed a login transaction, may also be needed to initiate the session. The goal of the registration transaction is to establish a secure channel between two repositories who know each others identities. As it is assumed that the communication channel between the repositories is reliable but not secure, there is a risk that a non-repository may mimic the protocol in order to gain illegitimate access to a repository.

The registration transaction between two repositories is described with respect to Figures 16 and 17. The steps described are from the perspective of a "repository-1" registering its identity with a "repository-2". The registration must be symmetrical so the same set of steps will be repeated for repository-2 registering its identity with repository-1. Referring to Figure 16, repository-1 first generates an encrypted registration identifier, step 1601 and then generates a registration message, step 1602. A registration message is comprised of an identifier of a master repository, the identification certificate for the repository-1 and an encrypted random registration identifier. The identification certificate is encrypted by the master repository in its private key and attests to the fact that the repository (here repository-1) is a bona fide repository. The identification certificate also contains a public key for the repository, the repository security level and a timestamp (indicating a time after which the certificate is no longer valid.) The registration identifier is a number generated by the repository for this registration. The registration identifier is unique to the session and is encrypted in repository-1's private key. The registration identifier is used to improve security of authentication by detecting certain kinds of communications based attacks. Repository-1 then transmits the registration message to repository-2, step 1603.

Upon receiving the registration message, repository-2 determines if it has the needed public key for the master repository, step 1604. If repository-2 does not have the needed public key to decrypt the identification certificate, the registration transaction terminates in an error, step 1618.

Assuming that repository-2 has the proper public key the identification certificate is decrypted, step 1605. Repository-2 saves the encrypted registration identifier, step 1606, and extracts the repository identifier, step 1607. The extracted repository identifier is checked against a "hotlist" of compromised document repositories, step 1608. In the currently preferred embodiment, each repository will contain "hotlists" of compromised repositories. If the repository is on the "hotlist', the registration transaction terminates in an error per step 1618. Repositories can be removed from the hotlist when their certificates expire, so that the list does not need to grow without bound. Also, by keeping a short list of hotlist certificates that it has previously received, a repository can avoid the work of actually going through the list. These lists would be encrypted by a master repository. A minor variation on the approach to improve efficiency would have the repositories first exchange lists of names of hotlist certificates, ultimately exchanging only those lists that they had not previously received. The "hotlists" are maintained and distributed by Master repositories.

Note that rather than terminating in error, the transaction could request that another registration message be sent based on an identification certificate created by another master repository. This may be repeated until a satisfactory identification certificate is found, or it is determined that trust cannot be established.

Assuming that the repository is not on the hotlist, the repository identification needs to be verified. In other words, repository-2 needs to validate that the repository on the other end is really repository-1. This is termed performance testing and is performed in order to avoid invalid access to the repository via a counterfeit repository replaying a recording of a prior session initiation between repository-1 and repository-2. Performance testing is initiated by repository-2 generating a performance message, step 1609. The performance message consists of a nonce, the names of the respective repositories, the time and the registration identifier received from repository-1. A nonce is a generated message based on some random and variable information (e.g. the time or the temperature.) The nonce is used to check whether repository-1 can actually exhibit correct encrypting of a message using the private keys it claims to have, on a message that it has never seen before. The performance message is encrypted using the public key specified in the registration message of repository-1. The performance message is transmitted to repository-1, step 1610, where it is decrypted by repository-1 using its private key, step 1611. Repository-1 then checks to make sure that the names of the two repositories are correct, step 1612, that the time is accurate, step 1613 and that the registration identifier corresponds to the one it sent, step 1614. If any of these tests fails, the transaction is terminated per step 1616. Assuming that the tests are passed, repository-1 transmits the nonce to repository-2 in the clear, step 1615. Repository-2 then compares the received nonce to the original nonce, step 1617. If they are not identical, the registration transaction terminates in an error per step 1618. If they are the same, the registration transaction has successfully completed.

At this point, assuming that the transaction has not terminated, the repositories exchange messages containing session keys to be used in all communications during the session and synchronize their clocks. Figure 17 illustrates the session information exchange and clock synchronization steps (again from the perspective of repository-1.) Referring to Figure 17, repository-1 creates a session key pair, step 1701. A first key is kept private and is used by repository-1 to encrypt messages. The second key is a public key used by repository-2 to decrypt messages. The second key is encrypted using the public key of repository-2, step 1702 and is sent to repository-2, step 1703. Upon receipt, repository-2 decrypts the second key, step 1704. The second key is used to decrypt messages in subsequent communications. When each repository has completed this step, they are both convinced that the other repository is bona fide and that they are communicating with the original. Each repository has given the other a key to be used in decrypting further communications during the session. Since that key is itself transmitted in the public key of the receiving repository only it will be able to decrypt the key which is used to decrypt subsequent messages.

After the session information is exchanged, the repositories must synchronize their clocks. Clock synchronization is used by the repositories to establish an agreed upon time base for the financial records of their mutual transactions. Referring back to Figure 17, repository-2 initiates clock synchronization by generating a time stamp exchange message, step 1705, and transmits it to repository-1, step 1706. Upon receipt, repository-1 generates its own time stamp message, step 1707 and transmits it back to repository-2, step 1708. Repository-2 notes the current time, step 1709 and stores the time received from repository-1, step 1710. The current time is compared to the time received from repository-1, step 1711. The difference is then checked to see if it exceeds a predetermined tolerance (e.g. one minute), step 1712. If it does, repository-2 terminates the transaction as this may indicate tampering with the repository, step 1713. If not repository-2 computes an adjusted time delta, step 1714. The adjusted time delta is the difference between the clock time of repository-2 and the average of the times from repository-1 and repository-2.

To achieve greater accuracy, repository-2 can request the time again up to a fixed number of times (e.g. five times), repeat the clock synchronization steps, and average the results.

A second session initiation transaction is a Login transaction. The Login transaction is used to check the authenticity of a user requesting a transaction. A Login transaction is particularly prudent for the authorization of financial transactions that will be charged to a credit server. The Login transaction involves an interaction between the user at a user interface and the credit server associated with a repository. The information exchanged here is a login string supplied by the repository/credit server to identify itself to the user, and a Personal Identification Number (PIN) provided by the user to identify himself to the credit server. In the event that the user is accessing a credit server on a repository different from the one on which the user interface resides, exchange of the information would be encrypted using the public and private keys of the respective repositories.

Billing Transactions

Billing Transactions are concerned with monetary transactions with a credit server. Billing Transactions are carried out when all other conditions are satisfied and a usage fee is required for granting the request. For the most part, billing transactions are well understood in the state of the art. These transactions are between a repository and a credit server, or between a credit server and a billing clearinghouse. Briefly, the required transactions include the following:

- Registration and LOGIN transactions by which the repository and user establish their bona fides to a credit server. These transactions would be entirely internal in cases where the repository and credit server are implemented as a single system.
- Registration and LOGIN transactions, by which a credit server establishes its bona fides to a billing clearinghouse.
- An Assign-fee transaction to assign a charge. The information in this transaction would include a transaction identifier, the identities of the repositories in the transaction, and a list of charges from the parts of the digital work. If there has been any unusual event in the transaction such as an interruption of communications, that information is included as well.
- A Begin-charges transaction to assign a charge. This transaction is much the same as an assign-fee transaction except that it is used for metered use. It includes the same information as the assign-fee transaction as well as the usage fee information. The credit-server is then responsible for running a clock.
- An End-charges transaction to end a charge for metered use. (In a variation on this approach, the repositories would exchange periodic charge information for each block of time.)
- A report-charges transaction between a personal credit server and a billing clearinghouse. This transaction is invoked at least once per billing period. It is used to pass along information about charges. On debit and credit cards, this transaction would also be used to update balance information and credit limits as needed.

All billing transactions are given a transaction ID and are reported to the credit severs by both the server and the client. This reduces possible loss of billing information if one of the parties to a transaction loses a banking card and

provides a check against tampering with the system.

Usage Transactions

After the session initiation transactions have been completed, the usage request may then be processed. To simplify the description of the steps carried out in processing a usage request, the term requester is used to refer to a repository in the requester mode which is initiating a request, and the term server is used to refer to a repository in the server mode and which contains the desired digital work. In many cases such as requests to print or view a work, the requester and server may be the same device and the transactions described in the following would be entirely internal. In such instances, certain transaction steps, such as the registration transaction, need not be performed.

There are some common steps that are part of the semantics of all of the usage rights transactions. These steps are referred to as the common transaction steps. There are two sets -- the "opening" steps and the "closing" steps. For simplicity, these are listed here rather than repeating them in the descriptions of all of the usage rights transactions.

Transactions can refer to a part of a digital work, a complete digital work, or a Digital work containing other digital works. Although not described in detail herein, a transaction may even refer to a folder comprised of a plurality of digital works. The term "work" is used to refer to what ever portion or set of digital works is being accessed.

Many of the steps here involve determining if certain conditions are satisfied. Recall that each usage right may have one or more conditions which must be satisfied before the right can be exercised. Digital works have parts and parts have parts. Different parts can have different rights and fees. Thus, it is necessary to verify that the requirements are met for ALL of the parts that are involved in a transaction For brevity, when reference is made to checking whether the rights exist and conditions for exercising are satisfied, it is meant that all such checking takes place for each of the relevant parts of the work.

Figure 18 illustrates the initial common opening and closing steps for a transaction. At this point it is assumed that registration has occurred and that a "trusted" session is in place. General tests are tests on usage rights associated with the folder containing the work or some containing folder higher in the file system hierarchy. These tests correspond to requirements imposed on the work as a consequence of its being on the particular repository, as opposed to being attached to the work itself. Referring to Figure 18, prior to initiating a usage transaction, the requester performs any general tests that are required before the right associated with the transaction can be exercised, step, 1801. For example, install, uninstall and delete rights may be implemented to require that a requester have an authorization certificate before the right can be exercised. Another example is the requirement that a digital ticket be present and punched before a digital work may be copied to a requester. If any of the general tests fail, the transaction is not initiated, step, 1802. Assuming that such required tests are passed, upon receiving the usage request, the server generates a transaction identifier that is used in records or reports of the transaction, step 1803. The server then checks whether the digital work has been granted the right corresponding to the requested transaction, step 1804. If the digital work has not been granted the right corresponding to the request, the transaction terminates, step 1805. If the digital work has been granted the requested right, the server then determines if the various conditions for exercising the right are satisfied. Time based conditions are examined, step 1806. These conditions are checked by examining the time specification for the the version of the right. If any of the conditions are not satisfied, the transaction terminates per step 1805.

Assuming that the time based conditions are satisfied, the server checks security and access conditions, step 1807. Such security and access conditions are satisfied if: 1) the requester is at the specified security class, or a higher security class, 2) the server satisfies any specified authorization test and 3) the requester satisfies any specified authorization tests and has any required digital tickets. If any of the conditions are not satisfied, the transaction terminates per step 1805.

Assuming that the security and access conditions are all satisfied, the server checks the copy count condition, step 1808. If the copy count equals zero, then the transaction cannot be completed and the transaction terminates per step 1805.

Assuming that the copy count does not equal zero, the server checks if the copies in use for the requested right is greater than or equal to any copy count for the requested right (or relevant parts), step 1809. If the copies in use is greater than or equal to the copy count, this indicates that usage rights for the version of the transaction have been exhausted. Accordingly, the server terminates the transaction, step 1805. If the copy count is less than the copies in use for the transaction the transaction can continue, and the copies in use would be incremented by the number of digital works requested in the transaction, step 1810.

The server then checks if the digital work has a "Loan" access right, step 1811. The "Loan" access right is a special case since remaining rights may be present even though all copies are loaned out. If the digital work has the "Loan" access right, a check is made to see if all copies have been loaned out, step 1812. The number of copies that could be loaned is the sum of the Copy-Counts for all of the versions of the loan right of the digital work. For a composite work, the relevant figure is the minimal such sum of each of the components of the composite work. If all copies have been loaned out, the remaining rights are determined, step 1813. The remaining-rights is determined from the remaining

rights specifications from the versions of the Loan right. If there is only one version of the Loan right, then the determination is simple. The remaining rights are the ones specified in that version of the Loan right, or none if Remaining-Rights: is not specified. If there are multiple versions of the Loan right and all copies of all of the versions are loaned out, then the remaining rights is taken as the minimum set (intersection) of remaining rights across all of the versions of the loan right. The server then determines if the requested right is in the set of remaining rights, step 1814. If the requested right is not in the set of remaining rights, the server terminates the transaction, step 1805.

If Loan is not a usage right for the digital work or if all copies have not been loaned out or the requested right is in the set of remaining rights, fee conditions for the right are then checked, step 1815. This will initiate various financial transactions between the repository and associated credit server. Further, any metering of usage of a digital work will commence. If any financial transaction fails, the transaction terminates per step 1805.

It should be noted that the order in which the conditions are checked need not follow the order of steps 1806-1815.

At this point, right specific steps are now performed and are represented here as step 1816. The right specific steps are described in greater detail below.

The common closing transaction steps are now performed. Each of the closing transaction steps are performed by the server after a successful completion of a transaction. Referring back to Figure 18, the copies in use value for the requested right is decremented by the number of copies involved in the transaction, step 1817. Next, if the right had a metered usage fee specification, the server subtracts the elapsed time from the Remaining-Use-Time associated with the right for every part involved in the transaction, step 1818. Finally, if there are fee specifications associated with the right, the server initiates End-Charge financial transaction to confirm billing, step 1819.

Transmission Protocol

An important area to consider is the transmission of the digital work from the server to the requester. The transmission protocol described herein refers to events occurring after a valid session has been created. The transmission protocol must handle the case of disruption in the communications between the repositories. It is assumed that interference such as injecting noise on the communication channel can be detected by the integrity checks (e.g., parity, checksum, etc.) that are built into the transport protocol and are not discussed in detail herein.

The underlying goal in the transmission protocol is to preclude certain failure modes, such as malicious or accidental interference on the communications channel. Suppose, for example, that a user pulls a card with the credit server at a specific time near the end of a transaction. There should not be a vulnerable time at which "pulling the card" causes the repositories to fail to correctly account for the number of copies of the work that have been created. Restated, there should be no time at which a party can break a connection as a means to avoid payment after using a digital work.

If a transaction is interrupted (and fails), both repositories restore the digital works and accounts to their state prior to the failure, modulo records of the failure itself.

Figure 19 is a state diagram showing steps in the process of transmitting information during a transaction. Each box represents a state of a repository in either the server mode (above the central dotted line 1901) or in the requester mode (below the dotted line 1901). Solid arrows stand for transitions between states. Dashed arrows stand for message communications between the repositories. A dashed message arrow pointing to a solid transition arrow is interpreted as meaning that the transition takes place when the message is received. Unlabeled transition arrows take place unconditionally. Other labels on state transition arrows describe conditions that trigger the transition.

Referring now to Figure 19, the server is initially in a state 1902 where a new transaction is initiated via start message 1903. This message includes transaction information including a transaction identifier and a count of the blocks of data to be transferred. The requester, initially in a wait state 1904 then enters a data wait state 1905.

The server enters a data transmit state 1906 and transmits a block of data 1907 and then enters a wait for acknowledgement state 1908. As the data is received, the requester enters a data receive state 1909 and when the data blocks are completely received it enters an acknowledgement state 1910 and transmits an Acknowledgement message 1911 to the server.

If there are more blocks to send, the server waits until receiving an Acknowledgement message from the requester. When an Acknowledgement message is received it sends the next block to the requester and again waits for acknowledgement. The requester also repeats the same cycle of states.

If the server detects a communications failure before sending the last block, it enters a cancellation state 1912 wherein the transaction is cancelled. Similarly, if the requester detects a communications failure before receiving the last block it enters a cancellation state 1913.

If there are no more blocks to send, the server commits to the transaction and waits for the final Acknowledgement in state 1914. If there is a communications failure before the server receives the final Acknowledgement message, it still commits to the transaction but includes a report about the event to its credit server in state 1915. This report serves two purposes. It will help legitimize any claims by a user of having been billed for receiving digital works that were not completely received. Also it helps to identify repositories and communications lines that have suspicious patterns of

use and interruption. The server then enters its completion state 1916.

On the requester side, when there are no more blocks to receive, the requester commits to the transaction in state 1917. If the requester detects a communications failure at this state, it reports the failure to its credit server in state 1918, but still commits to the transaction. When it has committed, it sends an acknowledgement message to the server. The server then enters its completion state 1919.

The key property is that both the server and the requester cancel a transaction if it is interrupted before all of the data blocks are delivered, and commits to it if all of the data blocks have been delivered.

There is a possibility that the server will have sent all of the data blocks (and committed) but the requester will not have received all of them and will cancel the transaction. In this case, both repositories will presumably detect a communications failure and report it to their credit server. This case will probably be rare since it depends on very precise timing of the communications failure. The only consequence will be that the user at the requester repository may want to request a refund from the credit services -- and the case for that refund will be documented by reports by both repositories.

To prevent loss of data, the server should not delete any transferred digital work until receiving the final acknowledgement from the requester. But it also should not use the file. A well known way to deal with this situation is called "two-phase commit" or 2PC.

Two-phase commit works as follows. The first phase works the same as the method described above. The server sends all of the data to the requester. Both repositories mark the transaction (and appropriate files) as uncommitted. The server sends a ready-to-commit message to the requester. The requester sends back an acknowledgement. The server then commits and sends the requester a commit message. When the requester receives the commit message, it commits the file.

If there is a communication failure or other crash, the requester must check back with the server to determine the status of the transaction. The server has the last word on this. The requester may have received all of the data, but if it did not get the final message, it has not committed. The server can go ahead and delete files (except for transaction records) once it commits, since the files are known to have been fully transmitted before starting the 2PC cycle.

There are variations known in the art which can be used to achieve the same effect. For example, the server could use an additional level of encryption when transmitting a work to a client. Only after the client sends a message acknowledging receipt does it send the key. The client then agrees to pay for the digital work. The point of this variation is that it provides a clear audit trail that the client received the work. For trusted systems, however, this variation adds a level of encryption for no real gain in accountability.

The transaction for specific usage rights are now discussed.

The Copy Transaction

A Copy transaction is a request to make one or more independent copies of the work with the same or lesser usage rights. Copy differs from the extraction right discussed later in that it refers to entire digital works or entire folders containing digital works. A copy operation cannot be used to remove a portion of a digital work.

- The requester sends the server a message to initiate the Copy Transaction. This message indicates the work to be copied, the version of the copy right to be used for the transaction, the destination address information (location in a folder) for placing the work, the file data for the work (including its size), and the number of copies requested.
- The repositories perform the common opening transaction steps.
- The server transmits the requested contents and data to the client according to the transmission protocol. If a Next-Set-Of-Rights has been provided in the version of the right, those rights are transmitted as the rights for the work. Otherwise, the rights of the original are transmitted. In any event, the Copy-Count field for the copy of the digital work being sent right is set to the number-of-copies requested.
- The requester records the work contents, data, and usage rights and stores the work. It records the date and time that the copy was made in the properties of the digital work.
- The repositories perform the common closing transaction steps.

The Transfer Transaction

A Transfer transaction is a request to move copies of the work with the same or lesser usage rights to another repository. In contrast with a copy transaction, this results in removing the work copies from the server.

- The requester sends the server a message to initiate the Transfer Transaction. This message indicates the work to be transferred, the version of the transfer right to be used in the transaction, the destination address information for placing the work, the file data for the work, and the number of copies involved.

- The repositories perform the common opening transaction steps.
- The server transmits the requested contents and data to the requester according to the transmission protocol. If a Next-Set-Of-Rights has been provided, those rights are transmitted as the rights for the work. Otherwise, the rights of the original are transmitted.

In either case, the Copy-Count field for the transmitted rights are set to the number-of-copies requested.

- The requester records the work contents, data, and usage rights and stores the work.
- The server decrements its copy count by the number of copies involved in the transaction.
- The repositories perform the common closing transaction steps.
- If the number of copies remaining in the server is now zero, it erases the digital work from its memory.

The Loan Transaction

A loan transaction is a mechanism for loaning copies of a digital work. The maximum duration of the loan is determined by an internal parameter of the digital work. Works are automatically returned after a predetermined time period.

- The requester sends the server a message to initiate the Transfer Transaction. This message indicates the work to be loaned, the version of the loan right to be used in the transaction, the destination address information for placing the work, the number of copies involved, the file data for the work, and the period of the loan.
- The server checks the validity of the requested loan period, and ends with an error if the period is not valid. Loans for a loaned copy cannot extend beyond the period of the original loan to the server.
- The repositories perform the common opening transaction steps.
- The server transmits the requested contents and data to the requester. If a Next-Set-Of-Rights has been provided, those rights are transmitted as the rights for the work. Otherwise, the rights of the original are transmitted, as modified to reflect the loan period.
- The requester records the digital work contents, data, usage rights, and loan period and stores the work.
- The server updates the usage rights information in the digital work to reflect the number of copies loaned out.
- The repositories perform the common closing transaction steps.
- The server updates the usage rights data for the digital work. This may preclude use of the work until it is returned from the loan. The user on the requester platform can now use the transferred copies of the digital work. A user accessing the original repository cannot use the digital work , unless there are copies remaining. What happens next depends on the order of events in time.

Case 1. If the time of the loan period is not yet exhausted and the requester sends the repository a Return message.

- The return message includes the requester identification, and the transaction ID.
- The server decrements the copies-in-use field by the number of copies that were returned. (If the number of digital works returned is greater than the number actually borrowed, this is treated as an error.) This step may now make the work available at the server for other users.
- The requester deactivates its copies and removes the contents from its memory.

Case 2. If the time of the loan period is exhausted and the requester has not yet sent a Return message.

- The server decrements the copies-in-use field by the number digital works that were borrowed.
- The requester automatically deactivates its copies of the digital work. It terminates all current uses and erases the digital work copies from memory. One question is why a requester would ever return a work earlier than the period of the loan, since it would be returned automatically anyway. One reason for early return is that there may be a metered fee which determines the cost of the loan. Returning early may reduce that fee.

The Play Transaction

A play transaction is a request to use the contents of a work. Typically, to "play" a work is to send the digital work through some kind of transducer, such as a speaker or a display device. The request implies the intention that the contents will not be communicated digitally to any other system. For example, they will not be sent to a printer, recorded on any digital medium, retained after the transaction or sent to another repository.

This term "play" is natural for examples like playing music, playing a movie, or playing a video game. The general

form of play means that a "player" is used to use the digital work. However, the term play covers all media and kinds of recordings. Thus one would "play" a digital work, meaning, to render it for reading, or play a computer program, meaning to execute it. For a digital ticket the player would be a digital ticket agent.

- The requester sends the server a message to initiate the play transaction. This message indicates the work to be played, the version of the play right to be used in the transaction, the identity of the player being used, and the file data for the work.
- The server checks the validity of the player identification and the compatibility of the player identification with the player specification in the right. It ends with an error if these are not satisfactory.
- The repositories perform the common opening transaction steps.
- The server and requester read and write the blocks of data as requested by the player according to the transmission protocol. The requester plays the work contents, using the player.
- When the player is finished, the player and the requester remove the contents from their memory.
- The repositories perform the common closing transaction steps.

The Print Transaction

A Print transaction is a request to obtain the contents of a work for the purpose of rendering them on a "printer." We use the term "printer" to include the common case of writing with ink on paper. However, the key aspect of "printing" in our use of the term is that it makes a copy of the digital work in a place outside of the protection of usage rights. As with all rights, this may require particular authorization certificates.

Once a digital work is printed, the publisher and user are bound by whatever copyright laws are in effect. However, printing moves the contents outside the control of repositories. For example, absent any other enforcement mechanisms, once a digital work is printed on paper, it can be copied on ordinary photocopying machines without intervention by a repository to collect usage fees. If the printer to a digital disk is permitted, then that digital copy is outside of the control of usage rights. Both the creator and the user know this, although the creator does not necessarily give tacit consent to such copying, which may violate copyright laws.

- The requester sends the server a message to initiate a Print transaction. This message indicates the work to be played, the identity of the printer being used, the file data for the work, and the number of copies in the request.
- The server checks the validity of the printer identification and the compatibility of the printer identification with the printer specification in the right. It ends with an error if these are not satisfactory.
- The repositories perform the common opening transaction steps.
- The server transmits blocks of data according to the transmission protocol.
- The requester prints the work contents, using the printer.
- When the printer is finished, the printer and the requester remove the contents from their memory.
- The repositories perform the common closing transaction steps.

The Backup Transaction

A Backup transaction is a request to make a backup copy of a digital work, as a protection against media failure. In the context of repositories, secure backup copies differ from other copies in three ways: (1) they are made under the control of a Backup transaction rather than a Copy transaction, (2) they do not count as regular copies, and (3) they are not usable as regular copies. Generally, backup copies are encrypted.

Although backup copies may be transferred or copied, depending on their assigned rights, the only way to make them useful for playing, printing or embedding is to restore them.

The output of a Backup operation is both an encrypted data file that contains the contents and description of a work, and a restoration file with an encryption key for restoring the encrypted contents. In many cases, the encrypted data file would have rights for "printing" it to a disk outside of the protection system, relying just on its encryption for security. Such files could be stored anywhere that was physically safe and convenient. The restoration file would be held in the repository. This file is necessary for the restoration of a backup copy. It may have rights for transfer between repositories.

- The requester sends the server a message to initiate a backup transaction. This message indicates the work to be backed up, the version of the backup right to be used in the transaction, the destination address information for placing the backup copy, the file data for the work.
- The repositories perform the common opening transaction steps.
- The server transmits the requested contents and data to the requester. If a Next-Set-Of-Rights has been provided,

those rights are transmitted as the rights for the work. Otherwise, a set of default rights for backup files of the original are transmitted by the server.

- The requester records the work contents, data, and usage rights. It then creates a one-time key and encrypts the contents file. It saves the key information in a restoration file.
- The repositories perform the common closing transaction steps.

In some cases, it is convenient to be able to archive the large, encrypted contents file to secure offline storage, such as a magneto-optical storage system or magnetic tape. This creation of a non-repository archive file is as secure as the encryption process. Such non-repository archive storage is considered a form of "printing" and is controlled by a print right with a specified "archive-printer." An archive-printer device is programmed to save the encrypted contents file (but not the description file) offline in such a way that it can be retrieved.

The Restore Transaction

A Restore transaction is a request to convert an encrypted backup copy of a digital work into a usable copy. A restore operation is intended to be used to compensate for catastrophic media failure. Like all usage rights, restoration rights can include fees and access tests including authorization checks.

- The requester sends the server a message to initiate a Restore transaction. This message indicates the work to be restored, the version of the restore right for the transaction, the destination address information for placing the work, and the file data for the work.
- The server verifies that the contents file is available (i.e. a digital work corresponding to the request has been backed-up.) If it is not, it ends the transaction with an error.
- The repositories perform the common opening transaction steps.
- The server retrieves the key from the restoration file. It decrypts the work contents, data, and usage rights.
- The server transmits the requested contents and data to the requester according to the transmission protocol. If a Next-Set-Of-Rights has been provided, those rights are transmitted as the rights for the work. Otherwise, a set of default rights for backup files of the original are transmitted by the server.
- The requester stores the digital work.
- The repositories perform the common closing transaction steps.

The Delete Transaction

A Delete transaction deletes a digital work or a number of copies of a digital work from a repository. Practically all digital works would have delete rights.

- The requester sends the server a message to initiate a delete transaction. This message indicates the work to be deleted, the version of the delete right for the transaction.
- The repositories perform the common opening transaction steps.
- The server deletes the file, erasing it from the file system.
- The repositories perform the common closing transaction steps.

The Directory Transaction

A Directory transaction is a request for information about folders, digital works, and their parts. This amounts to roughly the same idea as protection codes in a conventional file system like TENEX, except that it is generalized to the full power of the access specifications of the usage rights language.

The Directory transaction has the important role of passing along descriptions of the rights and fees associated with a digital work. When a user wants to exercise a right, the user interface of his repository implicitly makes a directory request to determine the versions of the right that are available. Typically these are presented to the user -- such as with different choices of billing for exercising a right. Thus, many directory transactions are invisible to the user and are exercised as part of the normal process of exercising all rights.

- The requester sends the server a message to initiate a Directory transaction. This message indicates the file or folder that is the root of the directory request and the version of the directory right used for the transaction.
- The server verifies that the information is accessible to the requester. In particular, it does not return the names of any files that have a HIDE-NAME status in their directory specifications, and it does not return the parts of any folders or files that have HIDE-PARTS in their specification. If the information is not accessible, the server ends

the transaction with an error.

- The repositories perform the common opening transaction steps.
- The server sends the requested data to the requester according to the transmission protocol.
- The requester records the data.
- The repositories perform the common closing transaction steps.

The Folder Transaction

A Folder transaction is a request to create or rename a folder, or to move a work between folders. Together with Directory rights, Folder rights control the degree to which organization of a repository can be accessed or modified from another repository.

- The requester sends the server a message to initiate a Folder transaction. This message indicates the folder that is the root of the folder request, the version of the folder right for the transaction, an operation, and data. The operation can be one of create, rename, and move file. The data are the specifications required for the operation, such as a specification of a folder or digital work and a name.
- The repositories perform the common opening transaction steps.
- The server performs the requested operation -- creating a folder, renaming a folder, or moving a work between folders.
- The repositories perform the common closing transaction steps.

The Extract Transaction

A extract transaction is a request to copy a part of a digital work and to create a new work containing it. The extraction operation differs from copying in that it can be used to separate a part of a digital work from d-blocks or shells that place additional restrictions or fees on it. The extraction operation differs from the edit operation in that it does not change the contents of a work, only its embedding in d-blocks. Extraction creates a new digital work.

- The requester sends the server a message to initiate an Extract transaction. This message indicates the part of the work to be extracted, the version of the extract right to be used in the transaction, the destination address information for placing the part as a new work, the file data for the work, and the number of copies involved.
- The repositories perform the common opening transaction steps.
- The server transmits the requested contents and data to the requester according to the transmission protocol. If a Next-Set-Of-Rights has been provided, those rights are transmitted as the rights for the new work. Otherwise, the rights of the original are transmitted. The Copy-Count field for this right is set to the number-of-copies requested.
- The requester records the contents, data, and usage rights and stores the work. It records the date and time that new work was made in the properties of the work.
- The repositories perform the common closing transaction steps.

The Embed Transaction

An embed transaction is a request to make a digital work become a part of another digital work or to add a shell d-block to enable the adding of fees by a distributor of the work.

- The requester sends the server a message to initiate an Embed transaction. This message indicates the work to be embedded, the version of the embed right to be used in the transaction, the destination address information for placing the part as a a work, the file data for the work, and the number of copies involved.
- The server checks the control specifications for all of the rights in the part and the destination. If they are incompatible, the server ends the transaction with an error.
- The repositories perform the common opening transaction steps.
- The server transmits the requested contents and data to the requester according to the transmission protocol. If a Next-Set-Of-Rights has been provided, those rights are transmitted as the rights for the new work. Otherwise, the rights of the original are transmitted. The Copy-Count field for this right is set to the number-of-copies requested.
- The requester records the contents, data, and usage rights and embeds the work in the destination file.
- The repositories perform the common closing transaction steps.

The Edit Transaction

An Edit transaction is a request to make a new digital work by copying, selecting and modifying portions of an existing digital work. This operation can actually change the contents of a digital work. The kinds of changes that are permitted depend on the process being used. Like the extraction operation, edit operates on portions of a digital work. In contrast with the extract operation, edit does not affect the rights or location of the work. It only changes the contents. The kinds of changes permitted are determined by the type specification of the processor specified in the rights. In the currently preferred embodiment, an edit transaction changes the work itself and does not make a new work. However, it would be a reasonable variation to cause a new copy of the work to be made.

- The requester sends the server a message to initiate an Edit transaction. This message indicates the work to be edited, the version of the edit right to be used in the transaction, the file data for the work (including its size), the process-ID for the process, and the number of copies involved.
- The server checks the compatibility of the process-ID to be used by the requester against any process-ID specification in the right. If they are incompatible, it ends the transaction with an error.
- The repositories perform the common opening transaction steps.
- The requester uses the process to change the contents of the digital work as desired. (For example, it can select and duplicate parts of it; combine it with other information; or compute functions based on the information. This can amount to editing text, music, or pictures or taking whatever other steps are useful in creating a derivative work.)
- The repositories perform the common closing transaction steps.

The edit transaction is used to cover a wide range of kinds of works. The category describes a process that takes as its input any portion of a digital work and then modifies the input in some way. For example, for text, a process for editing the text would require edit rights. A process for "summarizing" or counting words in the text would also be considered editing. For a music file, processing could involve changing the pitch or tempo, or adding reverberations, or any other audio effect. For digital video works, anything which alters the image would require edit rights. Examples would be colorizing, scaling, extracting still photos, selecting and combining frames into story boards, sharpening with signal processing, and so on.

Some creators may want to protect the authenticity of their works by limiting the kinds of processes that can be performed on them. If there are no edit rights, then no processing is allowed at all. A processor identifier can be included to specify what kind of process is allowed. If no process identifier is specified, then arbitrary processors can be used. For an example of a specific process, a photographer may want to allow use of his photograph but may not want it to be colorized. A musician may want to allow extraction of portions of his work but not changing of the tonality.

Authorization Transactions

There are many ways that authorization transactions can be defined. In the following, our preferred way is to simply define them in terms of other transactions that we already need for repositories. Thus, it is convenient sometimes to speak of "authorization transactions," but they are actually made up of other transactions that repositories already have.

A usage right can specify an authorization-ID, which identifies an authorization object (a digital work in a file of a standard format) that the repository must have and which it must process. The authorization is given to the generic authorization (or ticket) server of the repository which begins to interpret the authorization.

As described earlier, the authorization contains a server identifier, which may just be the generic authorization server or it may be another server. When a remote authorization server is required, it must contain a digital address. It may also contain a digital certificate.

If a remote authorization server is required, then the authorization process first performs the following steps:

- The generic authorization server attempts to set up the communications channel. (If the channel cannot be set up, then authorization fails with an error.)
- When the channel is set up, it performs a registration process with the remote repository. (If registration fails, then the authorization fails with an error.)
- When registration is complete, the generic authorization server invokes a "Play" transaction with the remote repository, supplying the authorization document as the digital work to be played, and the remote authorization server (a program) as the "player." (If the player cannot be found or has some other error, then the authorization fails with an error.)
- The authorization server then "plays" the authorization. This involves decrypting it using either the public key of the master repository that issued the certificate or the session key from the repository that transmitted it. The authorization server then performs various tests. These tests vary according to the authorization server. They

include such steps as checking issue and validity dates of the authorization and checking any hot-lists of known invalid authorizations. The authorization server may require carrying out any other transactions on the repository as well, such as checking directories, getting some person to supply a password, or playing some other digital work. It may also invoke some special process for checking information about locations or recent events. The "script" for such steps is contained within the authorization server.

- If all of the required steps are completed satisfactorily, the authorization server completes the transaction normally, signaling that authorization is granted.

The Install Transaction

An Install transaction is a request to install a digital work as runnable software on a repository. In a typical case, the requester repository is a rendering repository and the software would be a new kind or new version of a player. Also in a typical case, the software would be copied to file system of the requester repository before it is installed.

- The requester sends the server an Install message. This message indicates the work to be installed, the version of the Install right being invoked, and the file data for the work (including its size).
- The repositories perform the common opening transaction steps.
- The requester extracts a copy of the digital certificate for the software. If the certificate cannot be found or the master repository for the certificate is not known to the requester, the transaction ends with an error.
- The requester decrypts the digital certificate using the public key of the master repository, recording the identity of the supplier and creator, a key for decrypting the software, the compatibility information, and a tamper-checking code. (This step certifies the software.)
- The requester decrypts the software using the key from the certificate and computes a check code on it using a 1-way hash function. If the check-code does not match the tamper-checking code from the certificate, the installation transaction ends with an error. (This step assures that the contents of the software, including the various scripts, have not been tampered with.)
- The requester retrieves the instructions in the compatibility-checking script and follows them. If the software is not compatible with the repository, the installation transaction ends with an error. (This step checks platform compatibility.)
- The requester retrieves the instructions in the installation script and follows them. If there is an error in this process (such as insufficient resources), then the transaction ends with an error. Note that the installation process puts the runnable software in a place in the repository where it is no longer accessible as a work for exercising any usage rights other than the execution of the software as part of repository operations in carrying out other transactions.
- The repositories perform the common closing transaction steps.

The Uninstall Transaction

An Uninstall transaction is a request to remove software from a repository. Since uncontrolled or incorrect removal of software from a repository could compromise its behavioral integrity, this step is controlled.

- The requester sends the server an Uninstall message. This message indicates the work to be uninstalled, the version of the Uninstall right being invoked, and the file data for the work (including its size).
- The repositories perform the common opening transaction steps.
- The requester extracts a copy of the digital certificate for the software. If the certificate cannot be found or the master repository for the certificate is not known to the requester, the transaction ends with an error.
- The requester checks whether the software is installed. If the software is not installed, the transaction ends with an error.
- The requester decrypts the digital certificate using the public key of the master repository, recording the identity of the supplier and creator, a key for decrypting the software, the compatibility information, and a tamper-checking code. (This step authenticates the certification of the software, including the script for uninstalling it.)
- The requester decrypts the software using the key from the certificate and computes a check code on it using a 1-way hash function. If the check-code does not match the tamper-checking code from the certificate, the installation transaction ends with an error. (This step assures that the contents of the software, including the various scripts, have not been tampered with.)
- The requester retrieves the instructions in the uninstallation script and follows them. If there is an error in this process (such as insufficient resources), then the transaction ends with an error.
- The repositories perform the common closing transaction steps.

Claims

1. A system for secure distribution and control of digital works between repositories comprising:

means for creating usage rights, each instance of a usage right representing a specific instance of how a digital work may be used or distributed;
means for attaching a created set of usage rights to a digital work;
a communications medium for coupling repositories to enable exchange of repository transaction messages;
a plurality of general repositories for storing and securely exchanging digital works with attached usage rights, each of said general repositories comprising:
a storage means for storing digital works and their attached usage rights;
an identification certificate for indicating that the associated general repository is secure;
an external interface for removably coupling to said communications medium;
a session initiation transaction processing means for establishing a secure and trusted session with another repository, said session initiation transaction processing means using said identification certificate;
a usage transaction processing means having a requester mode of operation for generating usage repository transaction messages to request access to digital works stored in another general repository, said usage repository transaction message specifying a usage right, said usage transaction processing means further having a server mode of operation for determining if a request for access to a digital work stored in said storage means may be granted, said request being granted only if the usage right specified in said request is attached to said digital work; and
an input means coupled to said usage transaction processing means for enabling user created signals to cause generation of a usage repository transaction message to request access to digital works.

2. The system as recited in Claim 1 further comprising a rendering system, said rendering system comprising:

a rendering repository for securely accessing digital works from a general repository, said rendering repository comprising;
a storage means for storing digital works and their attached usage rights;
an identification certificate, said identification certificate for indicating that the rendering repository is secure;
an external interface for removably coupling to said communications medium;
a session initiation transaction processing means for establishing a secure and trusted session with a general repository, said session initiation transaction processing means using said identification certificate;
a usage transaction processing means for generating usage repository transaction messages to request access to digital works stored in a general repository, said usage repository transaction message specifying a usage right;
an input means coupled to said usage transaction processing means for enabling user created signals to cause generation of usage repository transaction messages to request access to digital works;
a rendering device for rendering digital works.

3. The system as recited in Claim 1 wherein said means for creating usage rights is further for the specification of different sets of usage rights to be attached to digital works when a corresponding usage right is exercised.

4. The system as recited in Claim 1 wherein said usage rights grammar further defines means for specifying conditions which must be satisfied before a usage right may be exercised and said usage transaction processing means in said server mode is further comprised of means for determining if specified conditions for a usage right are satisfied before access is granted.

5. The system as recited in Claim 1 wherein a first usage right enables copying of a digital work and specification of a revenue owner who is paid a fee whenever a copy of said digital work is made.

6. A method for controlling distribution and use of digital works comprising the steps of:

a) attaching a set of usage rights to a digital work, each of said usage rights defining a specific instance of how a digital work may be used or distributed, said usage right specifying one or more conditions which must be satisfied in order for said usage right to be exercised and a next set of usage rights to be attached to a distributed digital work;
b) storing said digital work and its attached usage rights in a first repository;

c) a second repository initiating a request to access said digital work in said first repository, said request identifying a usage right representing how said second repository desires to use said digital work;
d) said first repository receiving said request from said second repository;
e) said first repository determining if the identified usage right is attached to said digital work;
f) said first repository denying access to said digital work if said identified usage right is not attached to said digital work;
g) if said identified usage right is attached to said digital work, said first repository determining if conditions specified by said usage right are satisfied;
h) if said conditions are not satisfied, said first repository denying access to said digital work;
i) if said conditions are satisfied, said first repository attaching a next set of usage rights to said digital work, said next set of usage rights specifying how said second repository may use and distribute said digital work; and
j) said first repository transmitting said digital work and said attached next set of usage rights to said second repository.

7. The method as recited in Claim 6 wherein said step of a second repository initiating a request to access said digital work in said first repository is further comprised of the steps of:

c1) said second repository initiating establishment of a trusted session with said first repository;
c2) said first repository performing a set of registration transaction steps with said second repository, successful completion of said set of registration transaction steps indicating that said first repository is a trusted repository;
c3) said second repository performing said set of registration transaction steps with said first repository, successful completion of said set of registration transaction steps indicating that said second repository is a trusted repository;
c4) if said first repository and said second repository each successfully complete said set of registration steps, said first and second repository exchanging session encryption and decryption keys for secure transmission of subsequent communications between said first and second repository; and
c5) if said first repository or said second repository cannot successfully complete said set of registration transaction steps, terminating said session.

8. A system for controlling distribution and use of digital works comprising:

means for attaching usage rights to said digital work, said usage rights indicating how a recipient may use and and subsequently distribute said digital work;
a communications medium for coupling repositories to enable distribution of digital works;
a plurality of repositories for managing exchange of digital works based on usage rights attached to said digital works, each of said plurality of repositories comprising:
a storage means for storing digital works and their attached usage rights;
a processor operating responsive to coded instructions;
a memory means coupled to said processor for storing coded instruction to enable said processor to operate in a first server mode for processing access requests to digital works and for attaching usage rights to digital works when transmitted to another of said plurality of repositories, a second requester mode for initiating requests to access digital works, and a session initiation mode for establishing a trusted session with another of said plurality of repositories over said communications medium;
a clock;
a repository interface for coupling to said communications medium.

9. The system as recited in Claim 8 further comprising a plurality of rendering systems for rendering of digital works, each of said rendering systems comprising:

a repository for secure receipt of a digital work; and
a rendering device having means for converting digital works to signals suitable for rendering of said digital works.

10. A method for secure access of digital works stored on a server repository, said digital works having associated therewith one or more usage rights for specifying how said digital work may be used or distributed, said method comprising the steps of:

a) a requesting repository performing a first registration transaction with a server repository, said first regis-

tration transaction for establishing to said server repository that said requesting repository is trustworthy;
b) concurrently with step a), said server repository responding with a second registration transaction, said second registration transaction for establishing to said requesting repository that said server repository is trustworthy;
c) if either said first registration transaction or said second registration transaction fails, said server repository denying access to said digital work;
d) if said first registration transaction and said second registration transaction are successful, said requesting repository initiating a usage transaction with respect to a digital work stored in said server repository, said usage transaction indicating a request to access a digital work and specifying a particular usage right;
e) determining if said usage transaction may be completed by comparing said particular usage right specified in said usage transaction and usage rights associated with said digital work;
f) if said particular usage right is not one of said usage rights associated with said digital work, denying access to said digital work; and
g) if said particular usage right is one of said usage rights associated with said digital work, granting access to said digital work and performing usage transaction steps associated with said particular usage right.

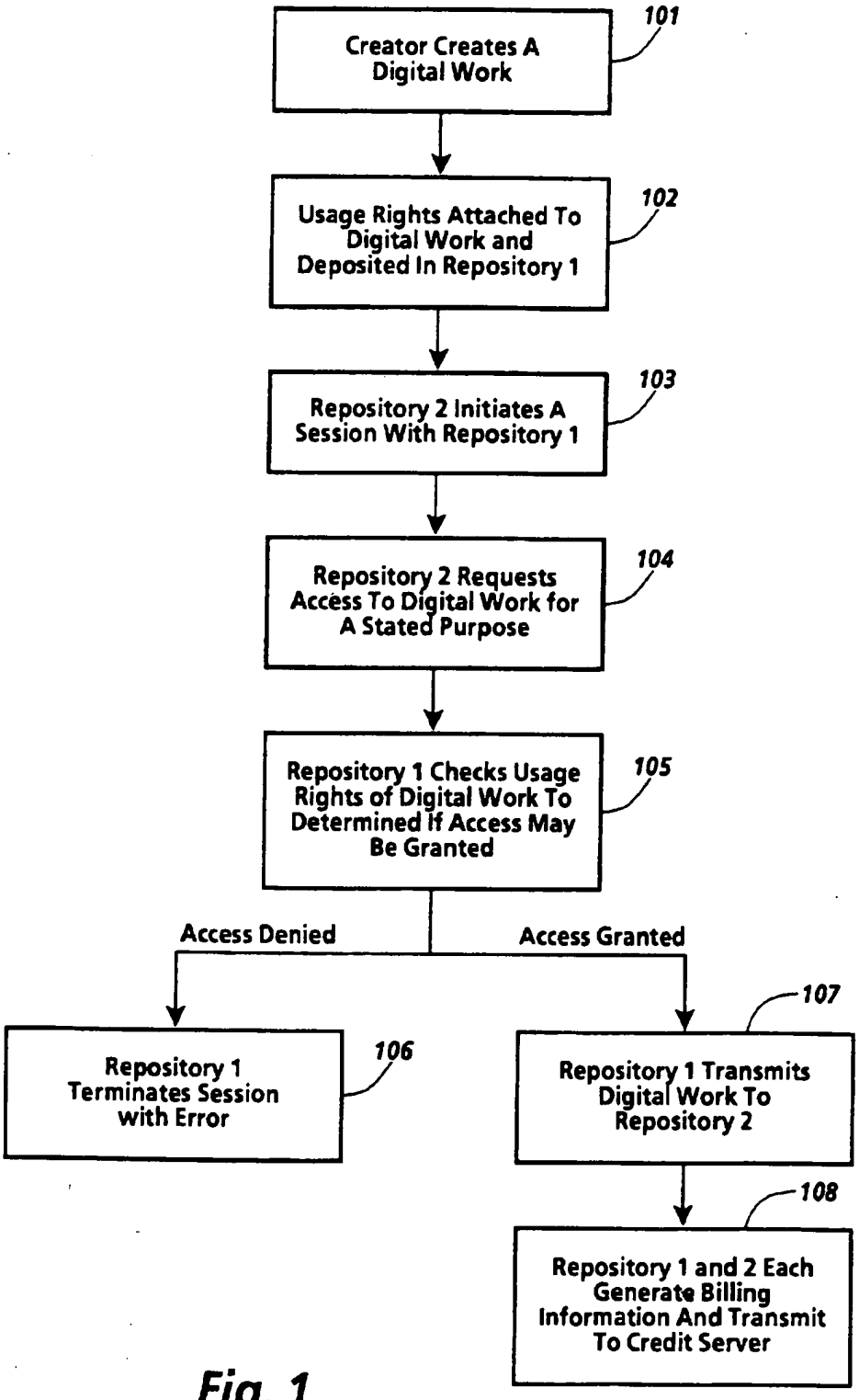


Fig. 1

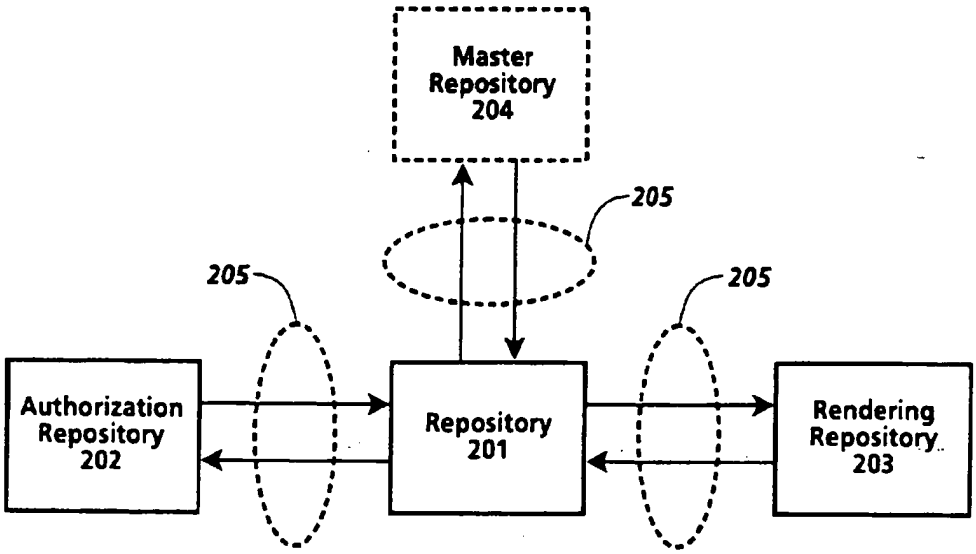


Fig. 2

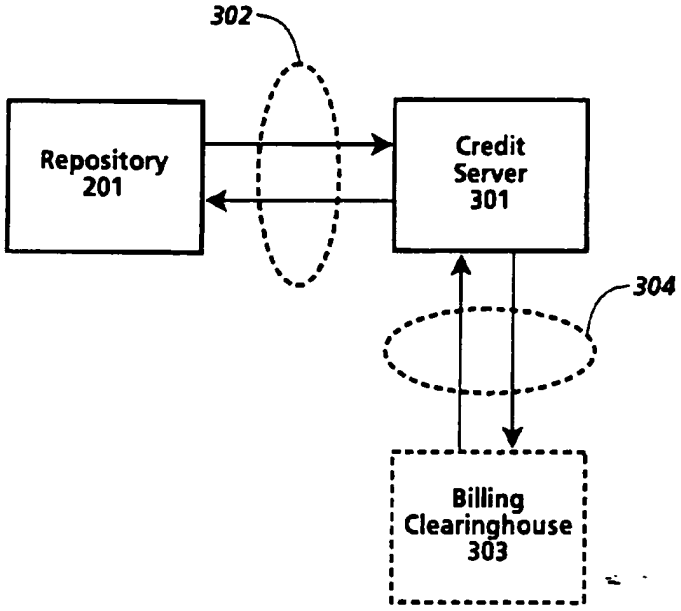


Fig. 3

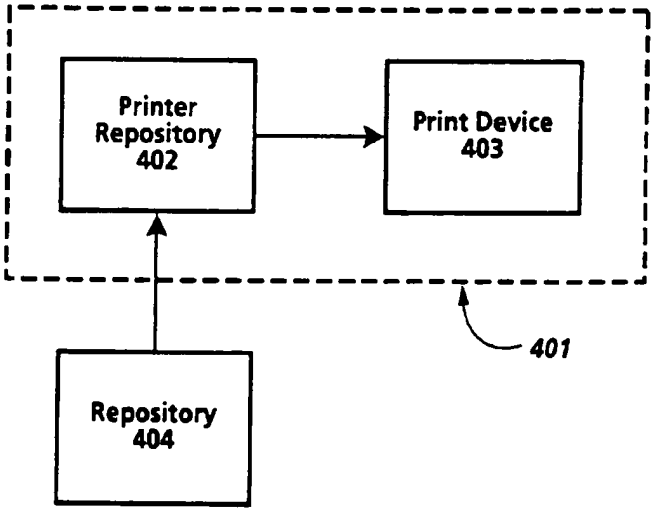


Fig. 4a

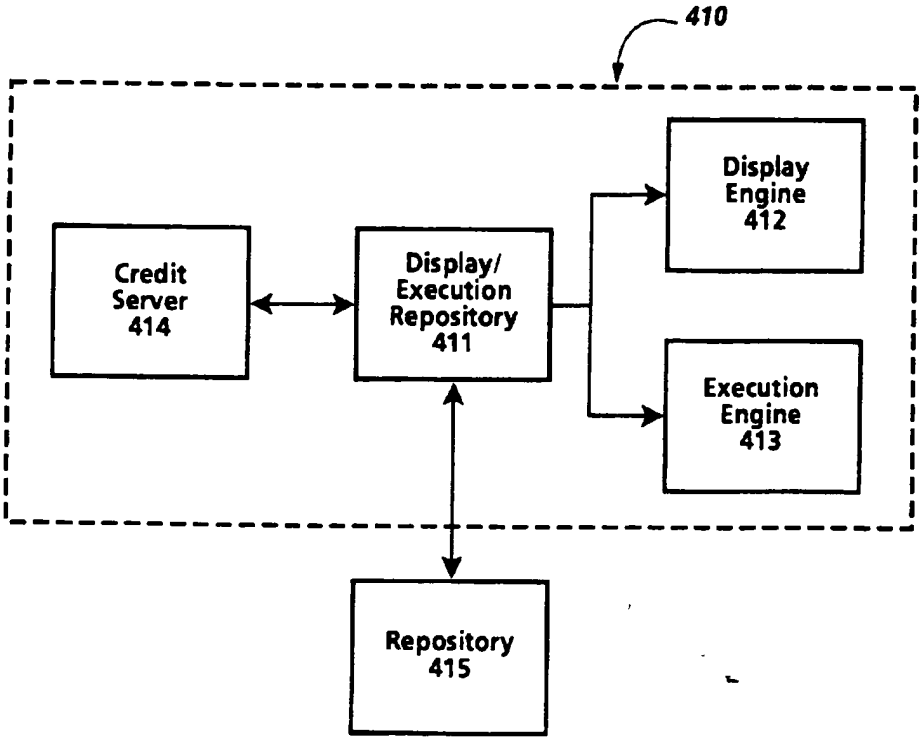


Fig. 4b

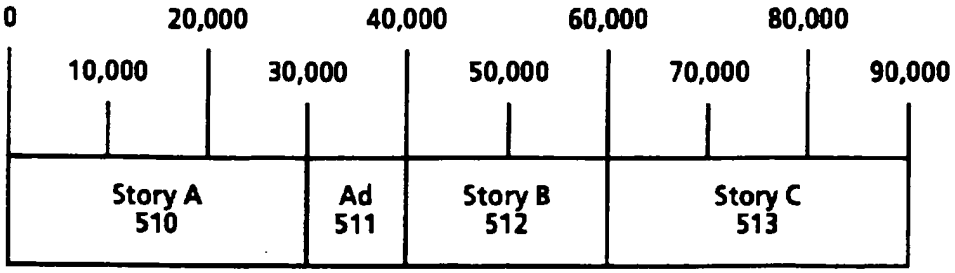


Fig. 5

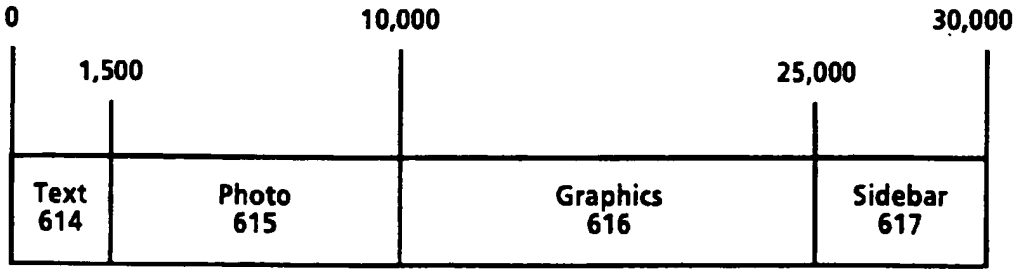


Fig. 6

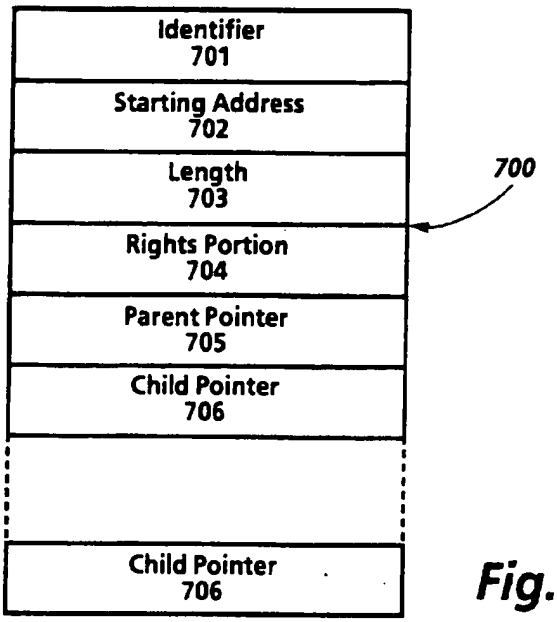


Fig. 7

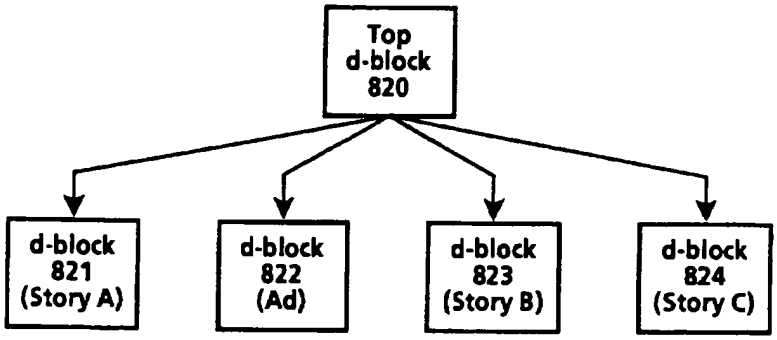


Fig. 8

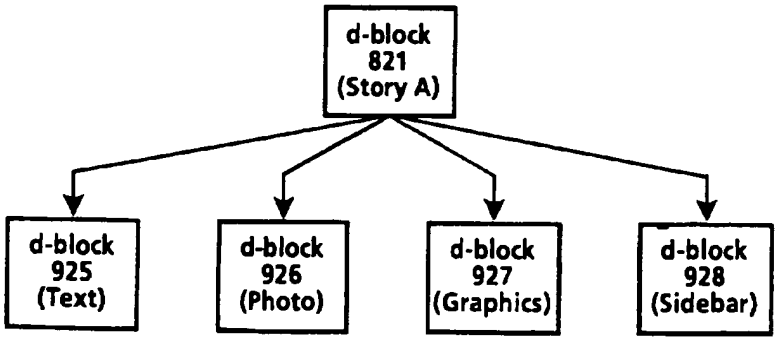


Fig. 9

Fig.10

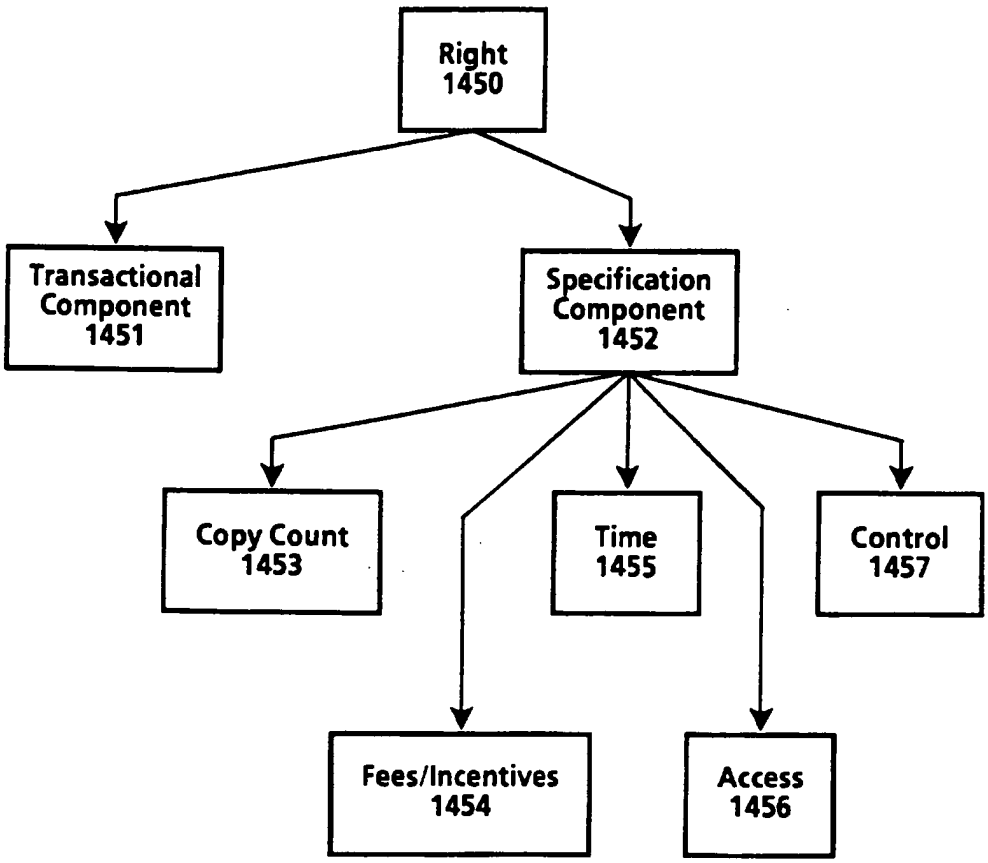


Fig.14

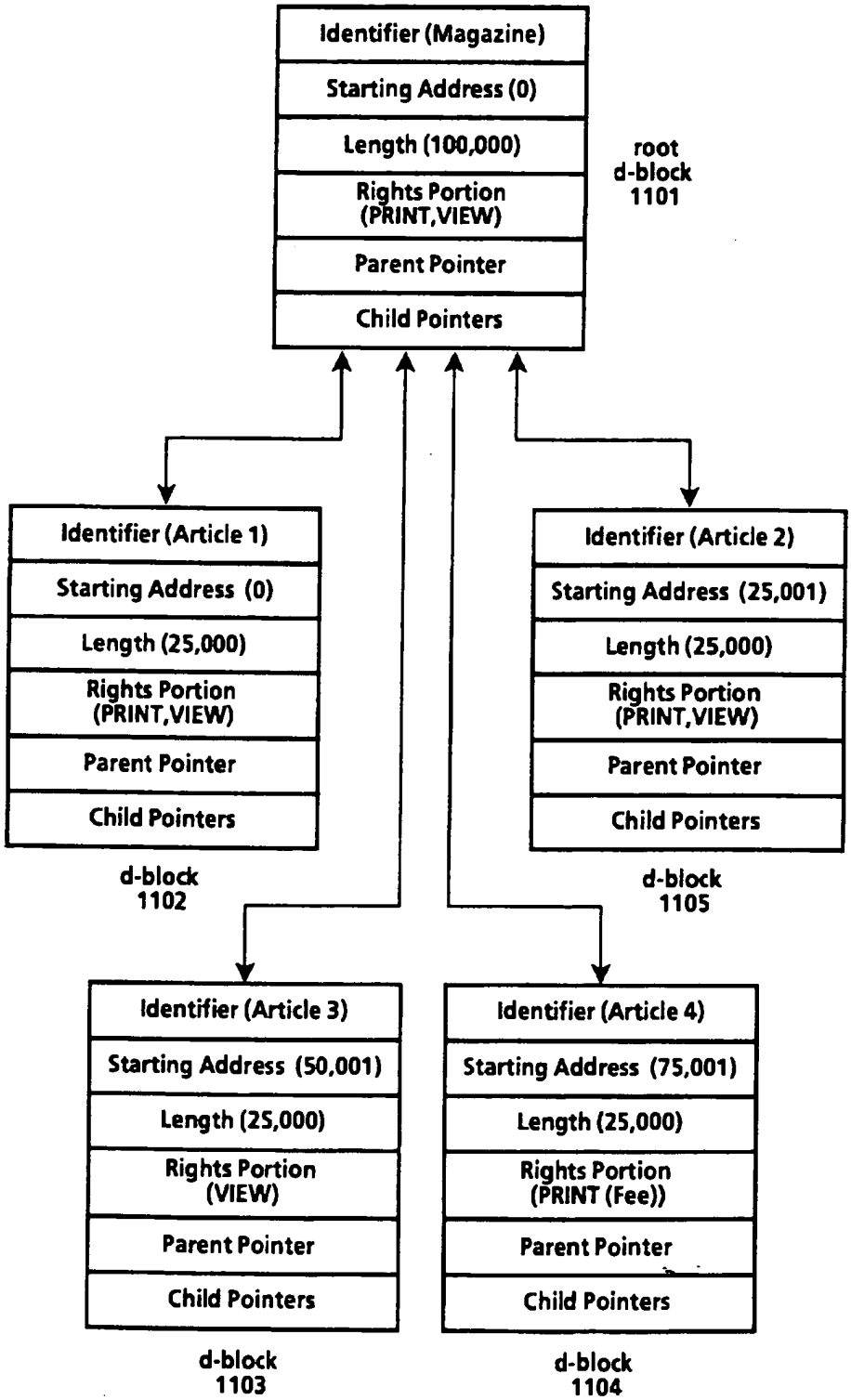


Fig.11

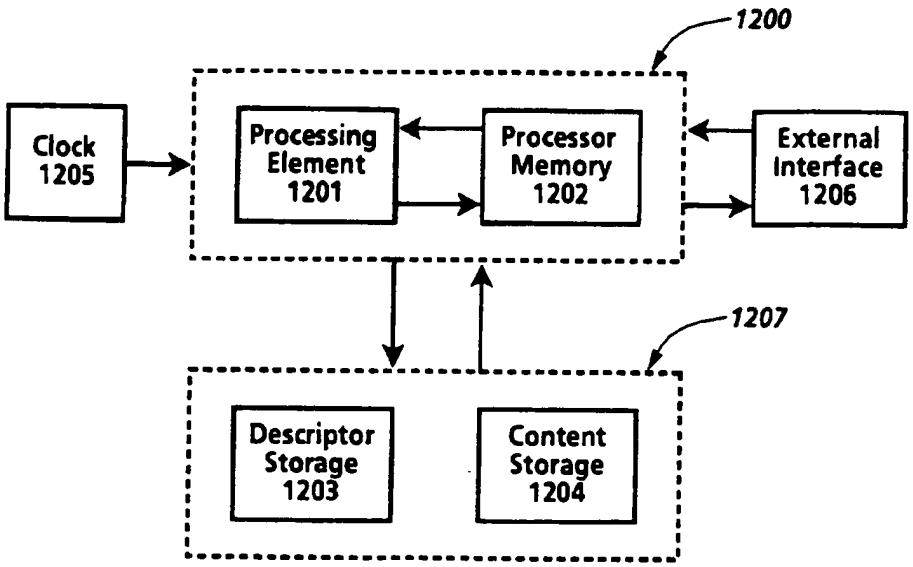


Fig.12

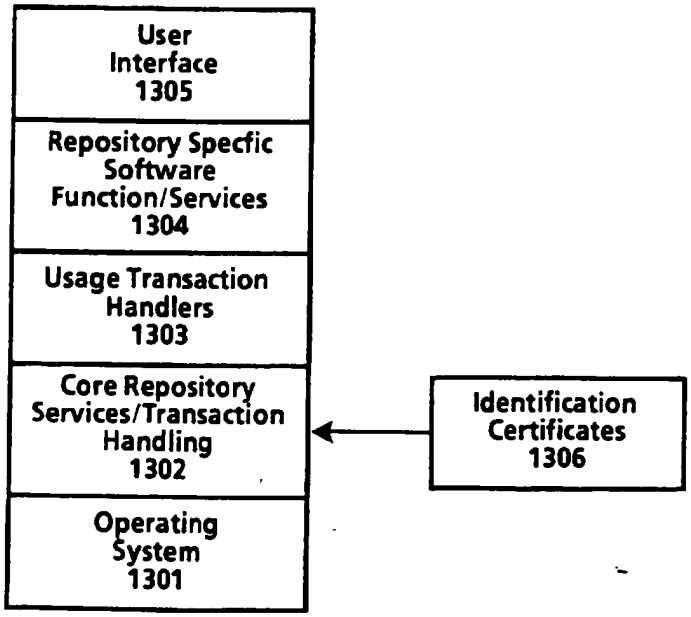


Fig.13

```
1501 ~Digital Work Rights: = (Rights*)
1502 ~Right : = (Right-Code {Copy-Count} {Control-Spec} {Time-Spec }
        {Access-Spec} {Fee-Spec})
  1503 ~Right-Code : = Render-Code | Transport-Code | File-Management-
        Code| Derivative-Works- Code | Configuration-Code
    1504 ~Render-Code : = [ Play : {Player: Player-ID} | Print: {Printer: Printer-ID}]
    1505 ~Transport-Code : = [Copy | Transfer | Loan {Remaining-Rights:
        Next-Set-of-Rights}]{(Next-Copy-Rights: Next-Set-of-Rights)}
    1506 ~File-Management-Code : = Backup {Back-Up-Copy-Rights:
                                  Next-Set-of-Rights} | Restore | Delete | Folder
                                  | Directory {Name: Hide-Local | Hide-Remote}
                                  {Parts: Hide-Local | Hide-Remote}
    1507 ~Derivative-Works-Code : = [Extract | Embed | Edit{Process:
                                  Process-ID)] {Next-Copy-Rights :
                                  Next-Set-of Rights}
    1508 ~Configuration-Code : = Install | Uninstall
    1509 ~Next-Set-of-Rights : = {(Add: Set-Of-Rights)} {(Delete:
        Set-Of-Rights)} {(Replace: Set-Of-Rights )}{(Keep: Set-Of-Rights )}
1510 ~Copy-Count : = (Copies:positive-integer | 0 | Unlimited)
1511 ~Control-Spec : = (Control: {Restrictable | Unrestrictable}
                    {Unchargeable | Chargeable})
1512 ~Time-Spec : = ({Fixed-Interval | Sliding-Interval | Meter-Time}
                  Until: Expiration-Date)
  1513 ~Fixed-Interval : = From: Start-Time
  1514 ~Sliding-Interval : = Interval: Use-Duration
  1515 ~Meter-Time: = Time-Remaining: Remaining-Use
1516 ~Access-Spec : = ({SC: Security-Class} {Authorization: Authorization-ID*}
      {Other-Authorization: Authorization-ID*} {Ticket: Ticket-ID})
1517 ~Fee-Spec: = {Scheduled-Discount} Regular-Fee-Spec | Scheduled-Fee-Spec |
                Markup-Spec
  1518 ~Scheduled-Discount: = Scheduled-Discount: (Scheduled-Discount:
                              (Time-Spec Percentage)*)
  1519 ~Regular-Fee-Spec : = ({Fee: | Incentive: } [Per-Use-Spec | Metered-Rate-
                            Spec | Best-Price-Spec | Call-For-Price-Spec]
                            {Min: Money-Unit Per: Time-Spec}{Max:
                            Money-Unit Per: Time-Spec} To: Account-ID)
    1520 ~Per-Use-Spec: = Per-Use: Money-unit
    1521 ~Metered-Rate-Spec : = Metered: Money-Unit Per: Time-Spec
    1522 ~Best-Price-Spec : = Best-Price: Money-unit Max: Money-unit
    1523 ~Call-For-Price-Spec : = Call-For -Price
  1524 ~Scheduled-Fee-Spec: = (Schedule: (Time-Spec Regular-Fee-Spec)* )
  1525 ~Markup-Spec: = Markup: percentage To: Account-ID
```

Fig.15

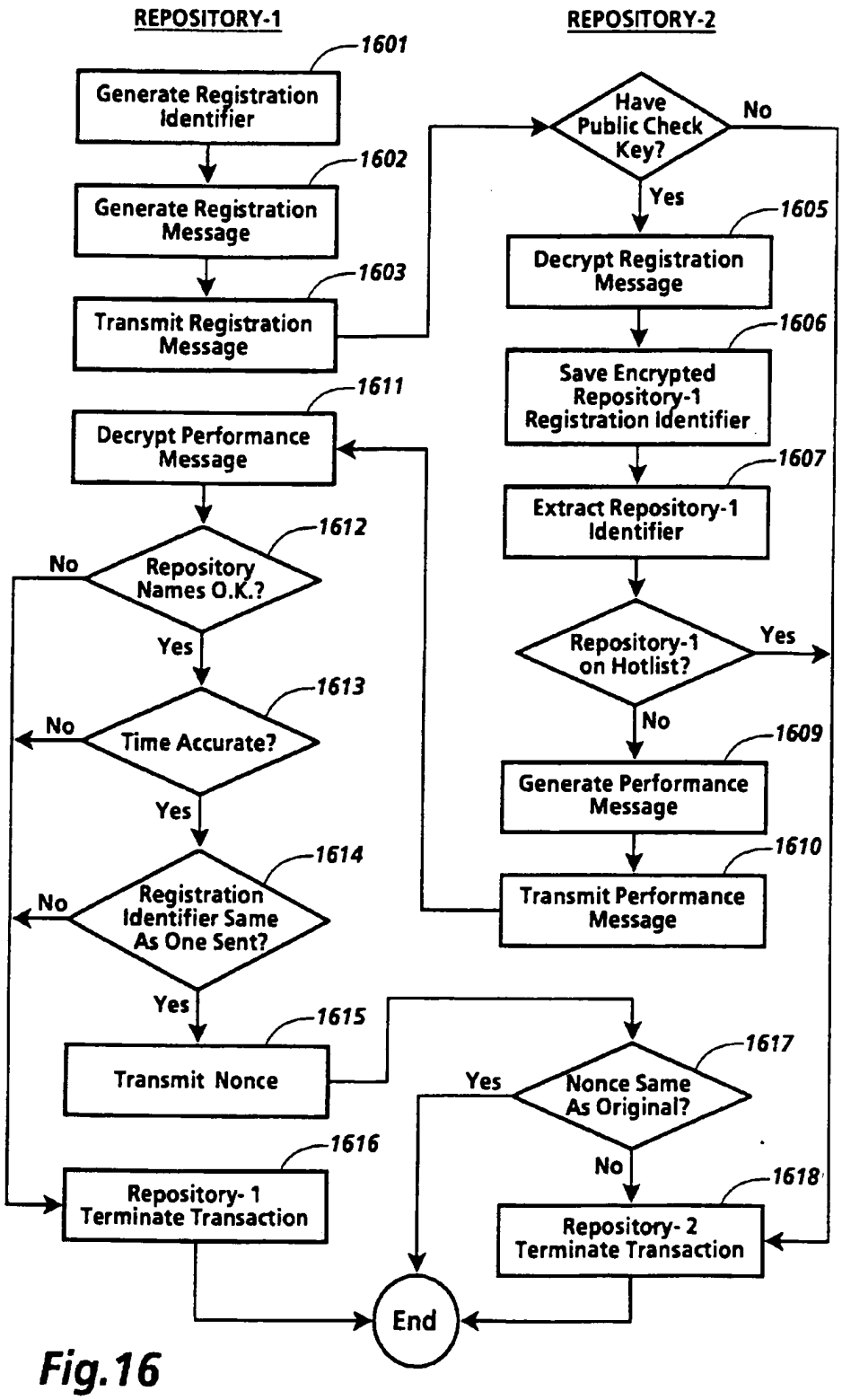


Fig.16

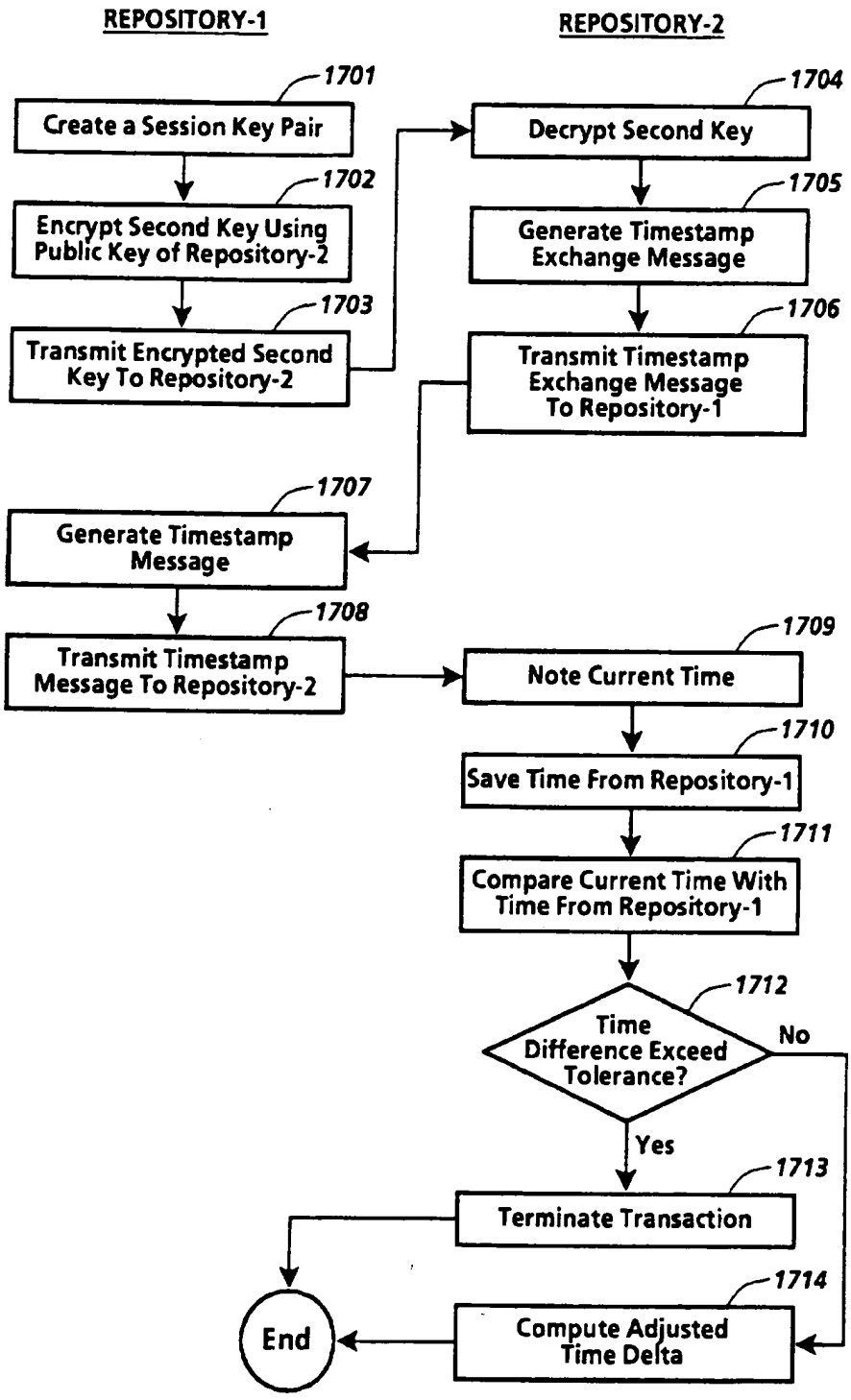


Fig.17

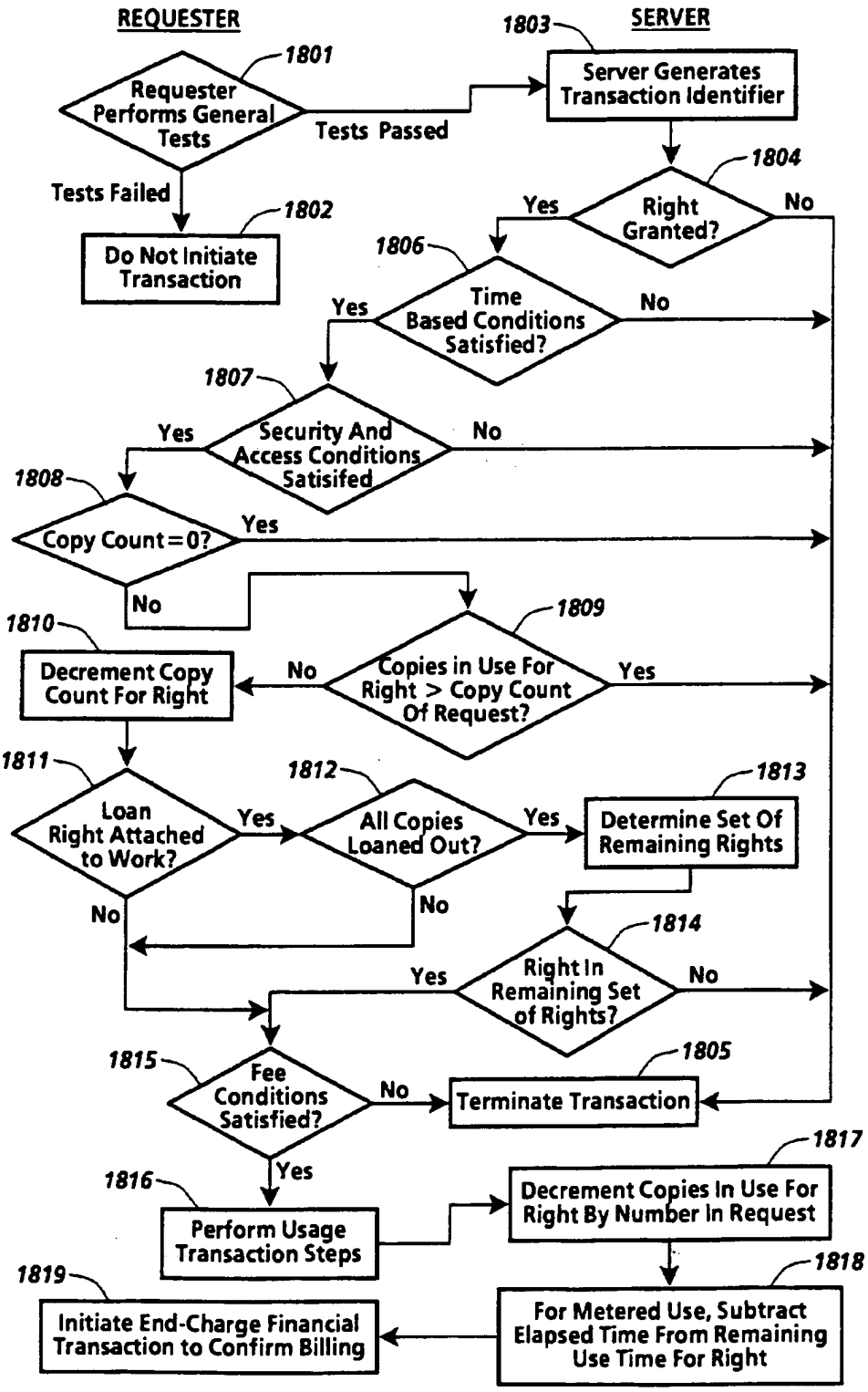


Fig.18

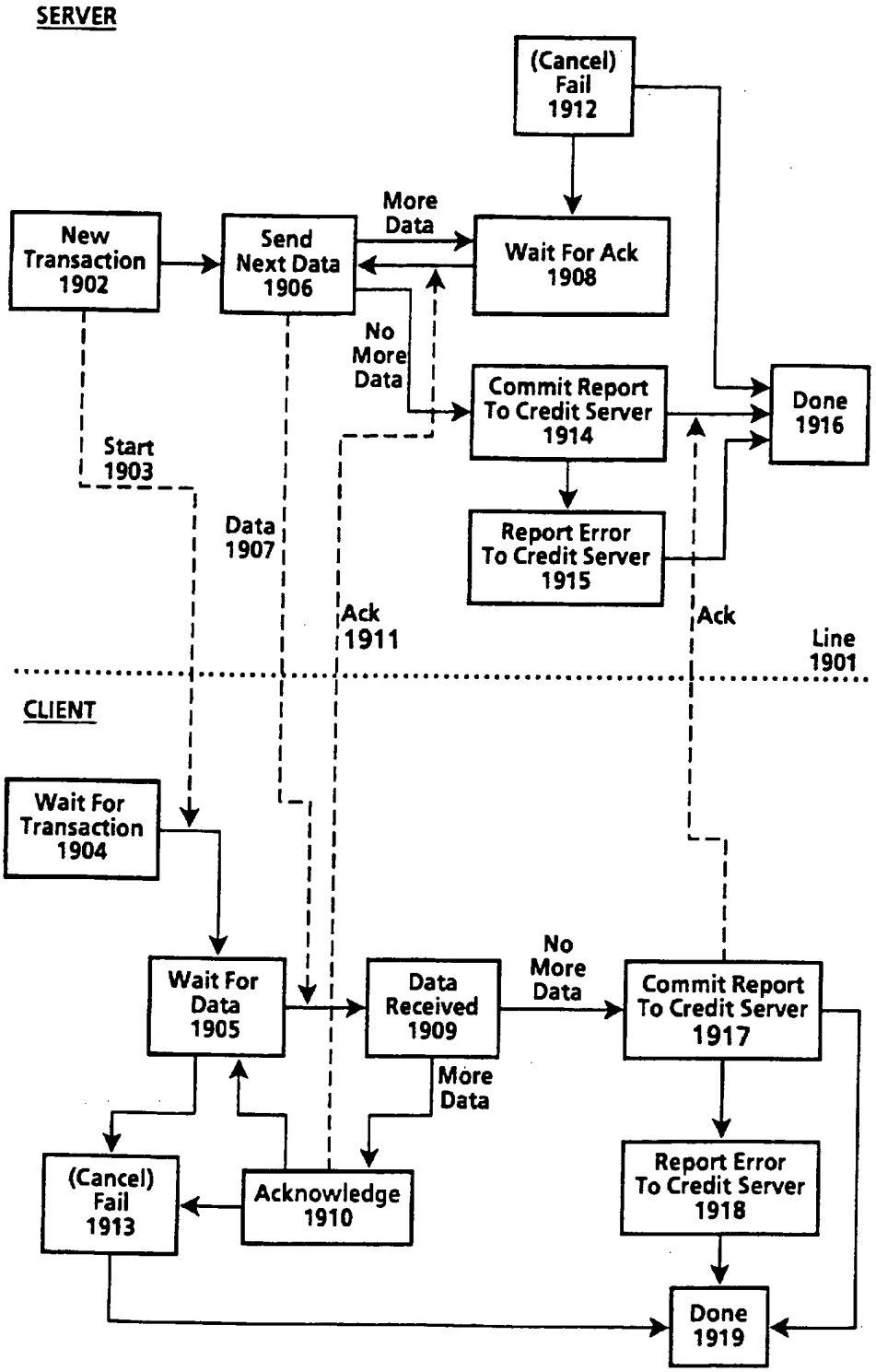


Fig.19

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 95 30 8420

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
A	WO-A-92 20022 (DIGITAL EQUIPMENT CORP.) * page 45, line 10 - page 64, line 17 * ---	1,6,8,10	G06F1/00
A	GB-A-2 236 604 (SUN MICROSYSTEMS INC) * page 9, line 11 - page 20, line 15 * ---	1,6,8,10	
A	US-A-5 291 596 (MITA) * the whole document * -----	1,6,8,10	

TECHNICAL FIELDS SEARCHED (Int.Cl.6)

G06F

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	1 April 1996	Moens, R

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 731 404 A1

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
11.09.1996 Bulletin 1996/37

(51) Int. Cl.[6]: **G06F 1/00**, G06F 19/00

(21) Application number: **96100832.3**

(22) Date of filing: **22.01.1996**

(84) Designated Contracting States:
DE FR GB

(30) Priority: **07.03.1995 US 401484**

(71) Applicant: **International Business Machines Corporation**
Armonk, N.Y. 10504 (US)

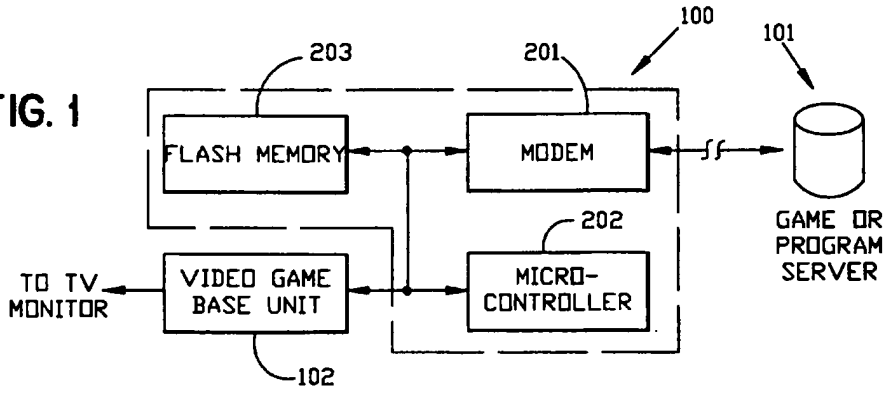
(72) Inventors:
- **Bakoglu, Halil Burhan**
  Ossining, New York 10562 (US)
- **Chen, Inching**
  Wappingers Falls, New York 12590 (US)
- **Lean, Andy Geng-Chyun**
  Merrick, New York 11566 (US)
- **Maruyama, Kiyoshi**
  Chappaqua, New York 10514 (US)
- **Yue, Chung-wai**
  Yorktown Heights, New York 10598 (US)

(74) Representative: **Rach, Werner, Dr.**
IBM Deutschland
Informationssysteme GmbH,
Patentwesen und Urheberrecht
70548 Stuttgart (DE)

(54) A universal electronic video game renting/distributing system

(57) A video game cartridge that can be plugged into a video game machine to enable a user to request and play a video game for a predetermined number of video frames. The cartridge has a receiver for receiving the video game program and the predetermined frame count in response to a request from the user. The program and frame count is then stored in a memory of the cartridge. Finally, the cartridge has a counter which changes its value when the user is actively playing the video game program. The counter ceases to change its value when the user is not playing the video game program. When the counter reaches a predetermined limit, the user is no longer authorized to play the video game program.

FIG. 1



EP 0 731 404 A1

Printed by Rank Xerox (UK) Business Services
2.13.4/3.4

Description

Technical Field

This invention relates to a video game cartridge for receiving video game programs from a remote server.

Description of the Prior Art

Today, there are many video games available for purchase or for rental at stores. Generally, there is no trial or test playing of the games in the stores, and there is no return on purchased games once the game package has been opened. Therefore, a person who is interested in any game has to buy it before playing it and thus may face the risk of not liking the game later. There is no return or refund of the game since the package has been opened. A person who rents a game from a store has to go through the usual VCR tape rental trouble of driving to the store, picking up the game and then later returning the game to the store.

To make video game rental easier for the consumer, Sega has created the Sega Channel. In this service, via cable and using a cable adapter unit which is plugged into the Sega Genesis game machine, people can play games that are downloaded to the cable adapter. It requires the on-line Sega Channel connection as well as the special adapter while the game is being played.

Down loading a software program to a personal computer over the modem connection exists today. Such software can come with a limited life where the life can be specified by expiration date, or time, or the number of times of the software usage. These schemes in limiting the software usage is not applicable to down loading video games to cartridges which are plugged into existing video game base units because these game base units do not have timer device built in. Thus a new scheme for controlling the usage of the game is needed.

The US Patent 4,905,280 to J.D. Wiedemer, et al describes a method for real time down loading of broadcast programs for pay-per-view or for subscription. Descrambling of broadcast programs is done by codes on a replaceable memory module, which is delivered to a subscriber by the service provider. This patent is applicable to the "purchase" of software content or real-time service, but it is not applicable to limiting the life of rented software.

US patent 5,251,909 to Reed et al describes software renting or distributing schemes in which access is granted to a subscriber prior to the actual programs being transmitted. This patent describes an off-line process and is not applicable to delivering software for rental purposes.

Summary of the Invention

It is an object of this invention to provide a portable video game cartridge which can be plugged into a video game machine base unit, such as Nintendos, Sega Genesis&tm. video game machine or Atari's Jaguar&tm. video game machine. The cartridge will allow a video game program to be used by receiving the video program over a telephone network or cable system.

The current invention describes a way of distributing and controlling the usage of a video game program (or any software program) by using a "watchdog mechanism" and by limiting the "life" of a game by limiting the total number of graphic frames that a video machine can generate. It offers a simple and effective way of software renting and distribution where game machines have no timer.

It is also an object of this invention to prevent piracy of video programs and programs in general by storing the frame count in a random location of the memory that is unknown to a potential pirate, especially if the count itself is encrypted. Since the count is part of the video game program or program execution path, the video game or program cannot be used without knowledge of the count.

This invention is generally an apparatus and method for enabling a user to request and use a program where the user receives the program and a frame count indicating the number of frames of the program that the user is authorized to execute or use. This program and the frame count is then stored in a memory. When the user is actively providing input to the program, the frame count changes. The frame count will cease to change when the user is not providing input to the program. When the count reaches a predetermined limit, the user is prevented from continuing use of the program.

This invention is a video game cartridge which can be plugged into a video game machine for enabling a user to receive and play a video game for a predetermined number of frames. The cartridge has a receiver for receiving the video program and for receiving a frame count indicating the number of video frames of the video game program that the user is authorized to play. The video program and frame count is then stored in a memory of the cartridge. The cartridge also has a counter which changes the frame count when the user is actively playing the video game program. When the user is not playing the video game program, the counter ceases to change its count. Finally when the counter reaches a predetermined limit, the user is prevented from further playing the video game program.

Brief Description of the Drawings

FIG. 1 schematically illustrates the major components of the video game cartridge along with a video game machine and a remote server.

FIG. 2 is a functional diagram showing the functions of each of the major components of the video game cartridge.

FIG. 3 schematically illustrates the flow chart for the watch "dog mechanism".

Description of the Preferred Embodiment

FIG. 1 illustrates a sample diagram of a electronic game or program renting system setup. The dotted line encloses the portable and programmable game cartridge unit 100 that can be plugged into a video game machine base unit 102, such as Sega Genesis&tm. video game machine, and remotely be connected to a video game server 101 via a modem connection. The connection to the remote video server can be through cable TV, or other telecommunication facilities.

When a video game base unit 102 is powered on, a user could either play a game (or games) stored in the programmable game cartridge 100 or place an order of a new game (either for rental or for purchase) to the game or program server 101. The cartridge 100 contains screen assistance (and voice assistance) to help place an order for a video game program to the server 101.

FIG. 2 illustrates the components of the video game cartridge unit 100. It consists of modem 201, microcontroller 202, flash memory 203 and an interface 204 to the video game base unit 102. The modem 201 performs the interface to the telephone or cable network. It can optionally perform decompression of received game or software if necessary. The received game is stored in flash memory 203. The game comes with its "life" which is indicated by the total number of graphic frames the video game machine 102 is authorized to generate when the game is actively played. For example, the game machine could render game graphics frame by frame at the rate of thirty framers per second.

After the number of graphic frames is exhausted, further playing of the game is prevented by the following mechanism. The flash memory 203 also stores a "watchdog mechanism" which keeps track of the remaining life of the game. An hourglass routine is embedded in the watchdog mechanism which is executed by microcontroller 202. This watchdog mechanism updates and tracks down a specified register in the flash memory 203 with its location randomly determined by the game server 101 in FIG. 1 during the down loading of the game.

The use of expiration date or time for voiding the game is an obvious approach if the video game base unit 102 comes with a timer. Since this patent application assumes a game base unit 102 which has no timer (which is the case of many existing game machines), the "life" of the rented game is determined by the total number of graphic frames that the base game unit can generate. This "life", or frame count, is what a renter gets when a game is down loaded. It is stored into a location in the flash memory 203. The location into which the frame count is stored in the flash memory is determined randomly by the video server at the time of the game down loading. The video game can resume at any time when it is being turned on, provided there is available frame count stored in the designated random location. The microcontroller 202 can pick up the frame count and allow the renting period, and thus the game or software, to be continued. As the rented game is being played, the frame count is decremented. When the user turns off the power, the hourglass routine in memory 203 will first store the remaining frame count to a random location in the non- volatile memory 203 and then shut down the game. The rental expires when there is no frame count remaining. The microcontroller 202 will not allow any portion of the game to be played by the game base unit 102 when the frame count reaches zero.

FIG. 3 illustrates the watchdog mechanism embedded with the video game program execution path that contains the hourglass routine which serves as part of the watchdog mechanism which can expire the game. When the user starts the game, the frame count is first fetched (305) and checked (306). If the frame count reaches zero, the game is over even though the game unit still has its power on (306N). If the frame count is still greater than zero (306Y), the scanner continues to monitor the game player's input in playing the video game (307). No active input (307) means the player is not playing the video game, and the scanner continues to monitor the player inputs from the key pad connected to the video game. When there is no active input, the video game will not render any game graphic frames. Therefore, the game program execution path will fall through decisions 308 and 309 and immediately return to continue scanning (307). When the game is not actively played and the player leaves the game machine's power on, the game will be sitting idle without rendering any new graphic frames. The frame count will not be consumed until the player becomes active again in playing the game as detected by the scanner (307 and 308).

If the player's input has been detected as active (307), a check is made to see if graphic rendering is required (309). Graphics rendering is required when the game program determines that the input signals from the key pad connected to the video game are valid signals. If rendering is required (309Y), the frame counter will be decremented (301). The hourglass routine (301 and 302) decrements the frame count and checks for any frame count left.

If the count is valid (302Y), then the program flows back to (310) which is the game program main collections, and then at the same time, 302 Y:sup.':esup. branches to check for power-off condition (303).

If the user decides to power-off the game, the watchdog mechanism will go through decision (303) and the shutdown routine (304) to store any remaining frame count in the flash memory. The shutdown routine stores the remaining frame count in the flash memory and exits the game. In summary flowchart components (301-306) and their associated flash memory form the "watchdog mechanism" that contains the hourglass rou-

tine (301 and 302) to keep track of the games "life" (remaining frame count). The watchdog mechanism also insures that the game can be resumed if there is still a valid frame count in the flash memory. Microcontroller (202) can also give advance warning when the rental is about to expire. Rental extension, if desired, can be downloaded again by the server (101) through a telephone or cable connection. Thus, server (101) in FIG. 1 has complete control over the game playing time, which should reflect the user's request for renting the game.

Although this embodiment was described in terms of a video game program in a cartridge, this invention can be extended to software programs in general. As long as the programs monitor user inputs, a scanner and watchdog mechanism can be implemented in similar fashion using a non-volatile memory.

The watchdog mechanism can even be made more secure by encrypting the frame count, which is stored at a random location in the memory. Even if the would-be pirate stumbles across the count in the memory, he/she wouldn't know what he/she found.

Claims

1. An apparatus for enabling a user to request and use a program, said apparatus comprising:

   a. a receiver for receiving the program and a frame count indicating a number of frames of the program that is authorized to be executed by the user;

   b. a memory for storing the program and the frame count received by the receiver; and

   c. a counter for changing the frame count when the user is actively providing input to the program, wherein the counter ceases to change its count when the user is not providing input to the program, and wherein the user is prevented from continuing use of the program when the counter reaches a predetermined limit.

2. An apparatus as recited in claim 1, further comprising:

   means for randomly determining an address in the memory in which the frame count is to be stored, and wherein the address is unknown to the user.

3. A method of enabling a user to request and use a program, said method comprising:

   a. receiving the game program and a frame count indicating a number of frames of the program that is authorized to be used by the user in response to a request;

   b. a memory for storing the program and the frame count; and

   c. changing the frame count when the user is actively using the program, wherein the frame count ceases to change when the user is not using the program and wherein the user is prevented from continuing use of the program when the counter reaches a predetermined limit.

4. A method as recited in claim 3, wherein the frame count is stored in a randomly determined location in the memory.

5. A video game cartridge which can be plugged into, for operation with, a video game machine to enable a user to request and play a video game program which is received from a remotely located server, said video game cartridge comprising:

   a. a receiver for receiving from the server the video game program and a frame count indicating a number of frames of the video game program that is authorized to be played by the user in response to a request;

   b. a memory for storing the video game program and the frame count received by the receiver; and

   c. a counter for changing the frame count when the user is actively playing the video game program, wherein the counter ceases to change its count when the user is not playing the video game program, and wherein the user is prevented from further playing the video game program when the counter reaches a predetermined limit, indicating that the user has played said video game for the number of frames.

6. A video game cartridge as recited in claim 5, further comprising:

   means for randomly determining an address in the memory in which the frame count is to be stored.

7. A video game cartridge as recited in claim 5, further comprising:

   a modem for transmitting to the server the request from the user to play a video game program.

8. A video game cartridge, as recited in claim 5, wherein said memory is a non-volatile memory.

9. A video game cartridge, as recited in claim 8, wherein the frame count indicated in the counter is stored in the memory when power for the video game machine is turned off.

10. A video game cartridge, as recited in claim 9, further comprising:

a means for fetching the frame count stored in the memory when power for said game machine is turned on.

11. A video game cartridge which can be plugged into, for operation with, a video game machine to enable a user to request and play a video game program which is received from a remotely located server, said video game cartridge comprising:

a. a modem for transmitting from the user over a telephone or cable network a request to receive the video game from the server, and for receiving the video game program and frame count from the server over the telephone or cable network, the frame count indicating a predetermined number of frames of the video game program that is authorized to be played by the user in response to the request;

b. a non-volatile memory for storing the video game program and the frame count;

c. a counter for changing the frame count when the player is actively playing the video game;

d. a means for storing the changed frame count of the counter in the memory when the power to the video game machine is turned off; and

e. a means for fetching the changed frame count stored in the memory in step (d) when the player resumes playing the video game, wherein the user is prevented from further playing of the video game program when the frame count of the counter reaches a predetermined limit, indicating that the user has played said video game for the predetermined number of frames.

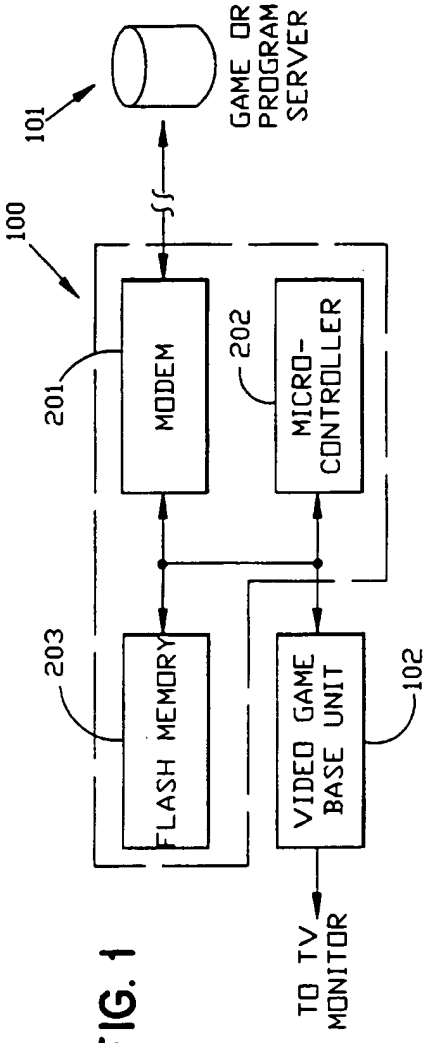


FIG. 1

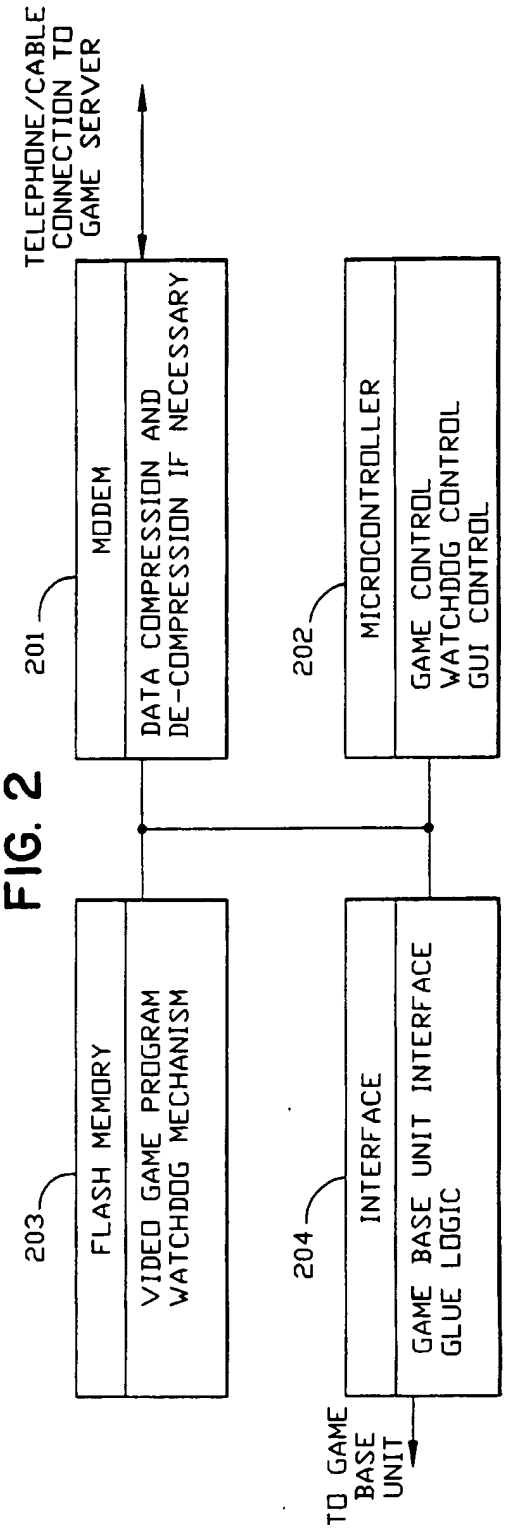
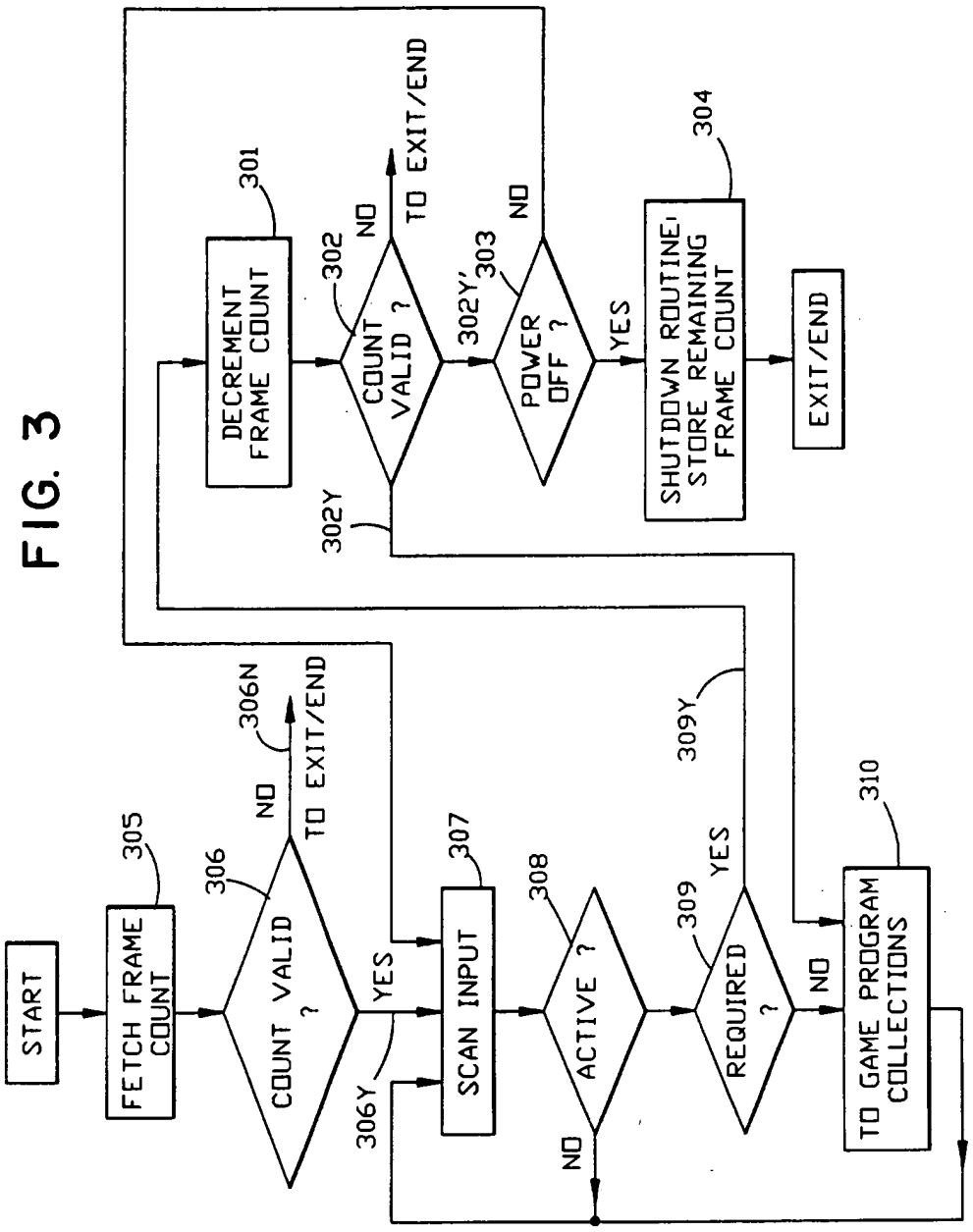


FIG. 2

FIG. 3



European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 96 10 0832

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
P,A	EP-A-0 671 711 (SEGA ENTERPRISES KK) 13 September 1995 * the whole document * ---	1-11	G06F1/00 G06F19/00
A	IBM TECHNICAL DISCLOSURE BULLETIN, vol. 37, no. 3, 1 March 1994, pages 413-417, XP000441522 "MULTIMEDIA MIXED OBJECT ENVELOPES SUPORTING A GRADUATED FEE SCHEME VIA ENCRYPTION" * page 413, line 1 - page 414, line 14 * ---	1-11	
A	WO-A-93 01550 (INFOLOGIC SOFTWARE INC) 21 January 1993 * page 1, line 1 - page 8, line 32 * * claims 1-3 * -----	1-11	

TECHNICAL FIELDS SEARCHED (Int.Cl.6)

G06F

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	14 June 1996	Powell, D

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 763 936 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
19.03.1997 Bulletin 1997/12

(51) Int Cl.6: H04N 5/913

(21) Application number: 96306507.3

(22) Date of filing: 06.09.1996

(84) Designated Contracting States:
DE FR GB

(30) Priority: 18.09.1995 KR 9530444

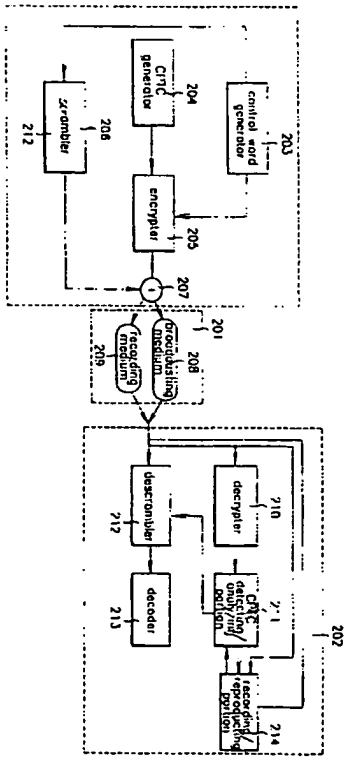
(71) Applicant: LG ELECTRONICS INC.
Seoul (KR)

(72) Inventors:
- Kim, Yung Gil, c/o LG Elec. Video-Media R&D
Seoul (KR)
- Park, Tae Joon
Seoul (KR)

(74) Representative:
Cross, Rupert Edward Blount et al
BOULT WADE TENNANT
27 Furnival Street
London EC4A 1PQ (GB)

(54) Illegal view/copy protection method and apparatus for digital broadcasting system

(57) An illegal view/copy protection method for a digital broadcasting system is disclosed including an audio/video signal transmission step (200,201) for multiplexing and transmitting audio/video bit stream scrambled in control words (206) and information where the control words and CPTC information for illegal view/copy protection are encrypted (208); and an audio/video reception step (202) for decrypting (210) the transmitted bit stream to analyze the CPTC information and control words (211), deciding whether recording is allowed or not to be recorded on cassette tape, and using the control words, performing descrambling (212) and decoding (213) to output audio/video signals to a monitor, thereby protecting copyright.



F I G.16

EP 0 763 936 A2

Printed by Jouve, 75001 PARIS (FR)

Description

Background of the Invention

The present invention relates to an illegal view/copy protection method and apparatus for a digital broadcasting system, in which digital broadcasting performed through broadcasting media such as cable, satellite and terrestrial broadcasting, or through prerecorded media such as video cassette tapes, is prevented from being illegally viewed or copied to thereby protect its copyright.

For conventional systems for copyright protection on digital media, there are Macrovision's intellectual property protection system (IPPS), which is disclosed in US Patent No. 5,315,448, and the integrated receiver/decoder (IRD), a conditional receiving system for digital broadcasting media, for receiving DirecTV's satellite broadcasting currently transmitted in the US.

The Macrovision's IPPS disclosed in US Patent No. 5,315,448 is a copy protection system for a hybrid digital VCR having digital recording functions for both a digital input signal and an analog input signal.

As shown in Figs. 1 and 2, in operating its copy protection function, Macrovision's IPPS detects, when a digital signal is input, copy protection control bits from an input signal, and when an analog signal is input, detects the analog copy protection waveform from the input signal.

More specifically, as shown in Fig. 2, a signal in which the analog copy protection waveform generated from an analog copy protection generator is added to the analog video output of the output signals of the digital VCR is output and displayed to be normal on an analog TV but distorted on an analog VCR, as shown in Fig. 1. In digital recording of the input signal, the copy protection control bits are changed to prevent digital copy or to permit one-time digital copy.

Referring to Fig. 3, the IPPS comprises an analog copy protection detector (ACP) 2 for detecting the analog copy protection waveform from an input analog NTSC video signal 1, an A/D converter 3 for A/D-converting analog NTSC video signal 1 input according to the signal output from the ACP detector, an AC bit detector 5 for detecting the AC bit from input digital video signal 4, an SCPS bit detector 6 for detecting the SCPS from input digital video signal 4, an AC bit adder 7 for adding the AC bit to input digital video signal 4 according to the SCPS bit output from SCPS bit detector 6, a switch 8 for outputting a signal output from AC bit adder 7 according to the AC bit output from AC bit detector 5, a switch 9 for selecting and outputting the signal output from A/D converter 3 and switch 8, a digital tape deck mechanism/circuit 10 for digitally recording the signal output from switch 9 and outputting a digital video signal, an AC bit detector 11 for detecting the AC bit from the signal output from digital tape deck mechanism/circuit 10, an ACP signal generator 12 for generating the ACP signal from the signal output from AC bit detector 11, and a D/A converter 13 for adding the ACP signal output from ACP signal generator 12 to the signal output from digital tape deck mechanism/circuit 10 and D/A converting the added result which is output as an analog NTSC video signal.

The operation of the IPPS will be explained below.

The copy protection control bits are made up of the AC and SCPS bits. The AC bit is added to recorded digital video data so that if the AC bit is set, digital copy is prohibited and if the SCPS bit is set, one-time digital copy is allowed.

In playback, when the AC bit is detected by AC bit detector 11, the analog copy protection waveform generated from ACP signal generator 12 is added to the analog video signal, which is output to D/A converter 13. Here, as the position of the copy protection control bits of the digital video data, an area of an MPEC-2 digital copy protection header where one-bit copyright flag and one-bit original-or-copy flag of a PES header are placed is used, or a transport-private-data field area of the transport header of the MPEC-2 is used.

The analog copy protection waveform is a signal which is severely distorted when inserted into the analog NTSC waveform and directly coupled to the analog TV. A method of generating such a signal is presented in US Patents Nos. 4,613,603 and 4,914,694. Using this method, the IPPS generates the analog copy protection waveform.

Referring to Fig. 4, the IRD, as a conditional receiving system for digital broadcasting media, for receiving the DirecTV's satellite broadcasting currently transmitted in US comprises an outdoor unit (ODU) 21 made up of a satellite antenna for receiving 12GHz-satellite broadcasting signals and a low noise block converter (LNB) for converting down the received satellite broadcasting signal into a 1GHz-signal, an IRD 20 for receiving satellite broadcasting from ODU 21 and offering audio and video services to a subscriber's TV or monitor, and an access card 22 required for conditional access (CA) for conditional reception.

Here, IRD 20 performs forward error correction (FEC), decoding, transport demultiplexing, MPEG decoding, NTSC encoding, and audio processing which is a D/A conversion.

Access card 22, whose size is similar to that of a general credit card, has a built-in IC. With this, the card receives CA-related information through a broadcast bit stream and telephone line, that is, a telco MODEM, in order to decide whether a user, subscriber, -selected channel can be viewed or not and to collect its subscription fee.

As shown in Fig. 4, IRD 20 comprises an IR receiver 25 for receiving and processing the subscriber's remote controller input, a telco MODEM 26 which is a general MODEM coupled to the telephone line, a microcomputer 27 made up of an NDC verifier code including software for the CA function and IRD software for IRD driving, a tuner/demodulator/FEC 28 for selecting one channel of

the signal received through ODU 21 and converting the selected channel into a digital bit stream for the purpose of error correction, a transport IC 29 for selecting one program of bit streams output from tuner/demodulator/FEC 28 and multiplexed with various programs, and converting the selected program into a bit stream decodable in the MPEG video decoder and MPEG audio decoder, a card reader interface 23 for data communication between transport IC 29 and access card 22, a system memory 24 coupled to transport IC 29 and for intermediate buffering of data, an MPEG video decoder 30 for expanding a video bit stream compressed in the MPEG format, a frame memory 31 for storing video data expanded in MPEG video decoder 30 in units of frame, an encode/sync/anti-tape/D/A 33 for converting the digital video data expanded in MPEG video decoder 30 into the analog NTSC format and inserting horizontal and vertical sync signals H-Sync and V-Sync and a Macrovision-mode analog copy protection signal in the conversion process, an RF modulator 34 for modulating an NTSC signal of the baseband output from encode/sync/anti-tape/D/A 33 into the RF band, an MPEG audio decoder 32 for expanding the audio bit stream compressed in the MPEG format, and a D/A 35 for converting the expanded digital audio data output from MPEG audio decoder 32 into analog.

Here, in the procedure of conversion into decodable bit stream in the MPEG video and audio decoders from transport IC 29, it is decided whether a program selected through communication with access card 22 can be viewed or not. If the bit stream is scrambled, its descrambling is performed with the access card's permission.

During the process of encode/sync/anti-tape/D/A 33 prior to NTSC video output, the analog copy protection waveform is added to prohibit copying to the analog VCR.

IRD 20 employs a CA system for conditional reception so that a subscriber views programs provided through a broadcasting medium such as satellite broadcasting.

In IRD 20, the NDC verifier code, which is software, and access card 22, which is a smart card for CA, are used to support CA function. A descrambler 36 is contained in transport IC 29.

The detailed block diagram of CA unit 37 and transport IC 29 for operating the CA function in a manner generally used in digital broadcasting is shown in Fig. 5.

More specifically, CA unit 37, included in smart card 22, is made up of smart card 38 for CA and microcomputer 39 operated with CA software.

The CA function is performed when the following two kinds of data are transmitted from a broadcasting station to the IRD. In other words, there are two types of data such as entitlement control message (ECM) or control word packet (CWP), and entitlement management message (EMM) or conditional access packet (CAP).

The EMM is accessed, through the telephone line or satellite broadcasting, to the smart card of the respective IRD at the data rate of 200kbps. The broadcasting station can access all of subscribers' smart cards in a manner that the EMM is transmitted along with ID or address. The EMM has information required to make a control word (CW) for descrambling from the ECM information. The ECM, information in which the control word is encrypted, is transmitted at a speed over 10 per second.

For satellite broadcasting, there are Europe's DVB, Korea's DBS, US' echoster, and the like, aside from DirecTV. Their CA function commonly uses the ECM and EMM information, though different means is provided for the respective broadcastings.

The conventional Macrovision's IPPS is a system having a good performance with respect to the copy protection of analog NTSC video signal. This is an appropriate copyright protection means when a program supplied through a digital medium is converted into analog audio/video signal and recorded or copied through an analog VCR.

However, the IPPS cannot guarantee a satisfactory protection if digital data is recorded or copied using a digital recording medium such as digital VCR. This is because the IPPS uses a method of operating the header's flag bits, without employing, to digital data, encoding methods such as scrambling and encryption. By doing so, hacking is easy to perform only by modulating the flag bits, resulting in very low security.

Summary of the Invention

It would therefore be desirable to provide an illegal view/copy protection method and apparatus for a digital broadcasting system in which intellectual properties supplied via digital media and protected by copyright are prohibited from being illegally recorded or copied using a digital recording medium such as digital VCR by a user.

It would also be desirable to provide an illegal view/copy protection method and apparatus for a digital broadcasting system in which data recorded on a cassette tape is always scrambled to make its hacking difficult and protect its copyright.

It would also be desirable to provide an illegal view/copy protection method and apparatus for a digital broadcasting system in which copyright is protected appropriately for respective media which are divided into broadcasting media and pre-recorded media.

It would also be desirable to provide an illegal view/copy protection method and apparatus for a digital broadcasting system in which intellectual properties supplied from a program provider are reproduced to be viewed on screen, copying of the intellectual properties copied and the number of copy are controlled arbitrarily, and fee for recording and copying is collected for the purpose of copyright protection.

According to a first aspect of the present invention, there is provided an illegal view/copy protection method for a digital broadcasting system comprising: an audio/video signal transmission step for multiplexing and transmitting audio/video bit stream scrambled in control words and information where the control words and CPTC information for illegal view/copy protection are encrypted; and an audio/video reception step for decrypting the transmitted bit stream to analyze the CPTC information and control words, deciding whether recording is allowed or not to be recorded on cassette tape, and using the control words, performing descrambling and decoding to output audio/video signals to a monitor.

According to a second aspect of the present invention, there is provided an illegal view/copy protection apparatus for a digital broadcasting system comprising: a program producing portion for multiplexing information encrypted both with the control word for scrambling and the CPTC information for prohibiting illegal view/copy, and the audio/video bit stream scrambled in control words, to thereby make a program; a distribution medium portion for distributing programs made in the program producing portion through a transmission medium; and a program receiving portion for detecting and analyzing the CPTC information from the bit stream transmitted from the distribution medium portion and the bit stream reproduced from cassette tape, and descrambling and decoding the bit stream transmitted from the distribution medium portion.

Brief Description of the Attached Drawings

Figs. 1 and 2 illustrate the operation state of a conventional IPPS;
Fig. 3 is a block diagram of a conventional IPPS;
Fig. 4 is a block diagram of an IRD system;
Fig. 5 shows a configuration of general hardware performing CA function;
Figs. 6A and 6B show formats of CPTC information of an embodiment of the present invention;
Fig. 7 shows a state of generation copy indicating the number of tape recopiable;
Figs. 8A-8D show the recording positions of the CPTC information of an embodiment of the present invention;
Fig. 9 is a flowchart of showing the transmission step of an illegal view/copy protection method embodying the present invention;
Fig. 10 is a flowchart of showing the reception step of an illegal view/copy protection method embodying the present invention;
Fig. 11 is a flowchart of the CPTC information analyzing step of Fig. 10;
Fig. 12 is a flowchart of showing the reproduction/rerecording step of an illegal view/copy protection method embodying the present invention;
Fig. 13 shows the format of an EMM lookup table;
Fig. 14 shows the format of a tape state signal;
Fig. 15 is a flowchart of showing the EMM processing step;
Fig. 16 is a block diagram of the whole configuration of an illegal view/copy protection apparatus embodying the present invention;
Fig. 17 is a block diagram of one embodiment of the program receiving portion of Fig. 16;
Fig. 18 is a block diagram of another embodiment of the program receiving portion of Fig. 16;
Fig. 19 is a block diagram of still another embodiment of the program receiving portion of Fig. 16;
Fig. 20 is a block diagram of yet another embodiment of the program receiving portion of Fig. 16;
Fig. 21 is a block diagram of the IRD shown in Figs. 17, 19 and 20;
Fig. 22 is a block diagram of the IRD and DVCR of Fig. 18;
Fig. 23 illustrates the flow of signals of Fig. 21;
Fig. 24 is a block diagram of one embodiment of the smart card of Fig. 17;
Fig. 25 is a block diagram of another embodiment of the smart card of Fig. 17; and
Fig. 26 is a block diagram of the DVCR of Fig. 17.

Detailed Description of the Invention

An illegal view/copy protection method for a digital broadcasting system embodying the present invention is performed by audio/video signal transmission and audio/video reception steps.

In the audio/video signal transmission step, audio/video bit stream scrambled in control words and information where the control words and CPTC information for illegal view/copy protection are encrypted are multiplexed and transmitted.

In the audio/video reception step, the bit stream transmitted in the audio/video signal transmission step is decrypted to analyze the CPTC information and control words. By doing so, it is decided whether recording is allowed or not. This result is recorded on cassette tape. Using the control words, descrambling and decoding are performed, and then audio/video signals are output to a monitor. Here, the CPTC information separately manages the ECM, EMM and control words, and contains CA information, to thereby control illegal view/copy protection. The CPTC information will be described with reference to Figs. 6A and 6B.

The CPTC information is formatted in a generational copy control field for limiting the number of copy available in order to control the depth of generational copy, and a reproducibility control field for limiting the reproduction of a copied program in order to control the number of copyiable tapes. As shown in Fig. 6A, formatting is performed containing a descrambling information field where part of the control words for descrambling are recorded, or containing a CA field where CA information for conditional access is recorded, as shown in Fig. 6B.

The CPTC information may be encrypted separately to be multiplexed with scrambled digital data, or contained in the ECM information for CA for encryption and multiplexing. Here, the generational copy control field is made up of a permissible generational field for limiting the number of copy permissible and a present generational field for indicating the present generation of a program copied. If the present generation stored in the present generational field is greater than or equal to the permissible generation stored in the permissible generational field, recording or copying is impossible.

A reproduction control field is made up of a reproducible number field for limiting the number of reproducing a copied program, and a maximum reproducible time field for limiting time to reproduce the copied program.

Here, the reproducible number stored in the reproducible number field implements a conditional-number reproducibility function according to the current reproduction number of cassette tape. The maximum reproducible time stored in the maximum reproducible time field implements the conditional-time reproducibility function of copied cassette tape according to the current time information of digital hardware.

The CPTC information may allow the copied cassette tape to be always reproducible, make it never reproducible, allow it to be reproducible as many as a limited number, or make the copied cassette tape reproducible for a limited time after recording or copying.

Using the permissible generational field and present generational field of the generational copy control field, the reproducible number field of the reproduction control field, and data of the maximum reproducible time field, the depth of generation copy, recopying of copied cassette tape, and reproduction time and number are controlled. This process controls the number of copiable cassette tape copied, and reproduction time and number.

In other words, as shown in Fig. 7, information stored in the permissible generational field and present generational field is used to allow first and second generation copy to be perform. Information stored in the reproducible number field and maximum reproducible time field is used to allow reproduction as many as a limited number or for a limited time.

In order to prohibit illegal recording or copy of a program protected by copyright law, collect fee for recording or copy, or arbitrarily control the number of reproducible copied tape to be made from a program supplied by a provider, the depth of generation copy and reproduction of copy tape are controlled to decide how long the first generation recording and copy and second generation copy are made possible.

For this purpose, the copy tape made to be always reproducible, it is made never to be reproducible, it is made to be reproducible as many as a limited number, or it is made to be reproducible for a limited time after recording or copy.

The data recorded on cassette tape contains scrambled audio/video bit stream and CPTC information. The CPTC information is recorded on a recording medium, that is, a rental tape, to prohibit illegal view/copy.

In other words, as shown in Fig. 8A, the CPTC information is overwritten on the scrambled audio/video bit stream for the error effect and recorded on cassette tape. Otherwise, as shown in Fig. 8B, the CPTC information is recorded on a portion of the audio track of cassette tape, on the control track of cassette tape as shown in Fig. 8C, or on the video track of cassette tape as shown in Fig. 8D.

In other words, as shown in Fig. 8A, the CPTC information is overwritten in a predetermined position in the form of error after parities for error correction, that is, inner and outer parities, are added to the scrambled digital data. This method reduces error correction capability but requires no additional tape area for recording the CPTC information. Further, during interleaving and decoding of ECC, the CPTC information is recognized as an error and removed, obtaining the scrambled digital data. Here, the CPTC information is detected separately.

In case that the CPTC information is recorded in part of audio track or control track, as shown in Figs. 8B and 8C, the audio head or control head must be additionally used as the means for detecting the CPTC so that audio track and control track are additionally accessed to detect the CPTC information.

The audio/video signal transmission step using the CPTC information will be explained with reference to Fig. 9.

One embodiment of the audio/video signal transmission step is to transmit an audio/video signal not containing the CA information for conditional access. This, having only the copy protection function, is used in case that a program which can be provided to all viewers is transmitted.

As shown in Fig. 9, the first embodiment of the audio/video signal transmission step comprises the steps of: encoding (100) the audio/video bit stream; generating (105) a control word for scrambling; scrambling (104) for the encoded audio/video bit stream using the generated control word; generating (102) CPTC information for illegal view/copy protection; encrypting (103) for encrypting the control word and CPTC information; and multiplexing and transmitting (106) the scrambled audio/video bit stream and encrypted CPTC information.

In other words, in step 100, the audio/video bit stream is encoded. In step 105, the control word for scrambling is generated. In step 104, the encoded audio/video bit stream is scrambled using the generated control word. In step 102, the CPTC information for illegal view/copy protection is generated. In step 103, the CPTC information and CA information are encrypted using the generated control word. The scrambled audio/video bit stream, encrypted CPTC information and CA

information are multiplexed and transmitted through a transmission medium in step 106. The audio/video signal transmitted through the first embodiment of the audio/video signal transmission step is received through one embodiment of an audio/video reception step.

Referring to Fig. 10, the first embodiment of the audio/video reception step comprises the steps of filtering (110) the transmitted bit stream and decrypting (111) the CPTC information; analyzing (113 and 114) the CPTC information to generate a control word and a signal for controlling the protection of copyright and to update the CPTC information; deciding (115) whether to allow recording according to the signal for controlling the protection of copyright to record the scrambled and transmitted bit stream on cassette tape; and descrambling and decoding (116 and 117) the transmitted bit stream in the control word and outputting an audio/video signal.

In other words, the bit stream transmitted in the first embodiment of the audio/video signal transmission step is filtered and the CPTC information is decrypted in steps 110 and 111. The CPTC information is analyzed to generate the control word and the signal for controlling the protection of copyright, and the CPTC information is updated in steps 113 and 114. Whether to allow recording is determined by the generated signal for controlling the protection of copyright so that the scrambled and transmitted bit stream is recorded on cassette tape in step 115. Then, the transmitted bit stream is descrambled and decoded in control words and output as an audio/video signal in steps 116 and 117. Here, all of the control word is contained in the CPTC information.

Referring to Fig. 11, the CPTC information analyzing step comprises the steps of detecting (130, 131, 132 and 133) the permissible generation of the permissible generational field for limiting the available number of copy of a program of the CPTC information and the present generation of the present generational field indicating the present generation of the program copied, to thereby perform copy-impossible and update the CPTC information; and detecting (134, 135, 136 and 137) the reproducible number of the reproducible number field for limiting the number of reproduction of copied programs of the CPTC information, the maximum reproducible time of the maximum reproducible time field for limiting time to reproduce the copied program, and the number and time of reproduction of tape, to thereby process reproduction-impossible.

The copying number limiting step comprises the steps of: comparing (130) the permissible generation of the permissible generational field and the present generation of the present generational field and deciding whether the permissible generation is below the present generation; if the permissible generation is below the present generation, generating (131) an output disable signal to make copying impossible and destroying the control word; and if the permissible generation is not below the present generation, increasing (132) the present invention by '1' and recording the result on cassette tape. If the permissible generation is not below the present generation, the CPTC information is updated in step 133, instead of increasing the present generation by '1.'

In order to control generation copy, the permissible generation of the permissible generational field and the present generation of the present generational field are compared in step 130. If the permissible generation is below the present generation, the output disable signal is generated to make copying impossible and the control word is destroyed in step 131. If the permissible generation is not below the present generation, the present generation is increased by '1' and thus recorded on cassette tape in step 132. This enables generation copy. Here, it can be possible that generation copy is limited by updating the CPTC information, instead of increasing the present generation by '1.'

The reproduction limiting step comprises the steps of: comparing the reproducible number of the reproducible number field and the reproduction number of tape and deciding (134) whether the reproducible number is below the reproduction number of tape; if the reproducible number is not below the reproduction number of tape, comparing the maximum reproducible time and reproduction time of tape, and deciding (135) whether the maximum reproducible time is below the reproduction time of tape; if the maximum reproducible time is not below reproduction time of tape, turning off (136) an enable erase signal to thereby enable the copied program to be reproduced; if the reproducible number is below the reproduction number of tape or the maximum reproducible time is below the reproduction time of tape, turning on (137) the enable erase signal to make the reproduction of the copied program impossible so that part of or the whole program recorded on cassette tape is erased.

In order to control reproduction, the reproducible number of the reproducible number field and the reproduction number of tape are compared in step 134. If the reproducible number is not below the reproduction number of tape, the maximum reproducible time of the maximum reproducible time field and the reproduction time of tape are compared and it is decided whether the maximum reproducible time is below the reproduction time of tape in step 135. In other words, though reproducible, whether it is limited by the reproducible time must be checked. If the maximum reproducible time is not below the reproduction time of tape, the enable erase signal is turned off in step 136 to thereby make the copied program reproducible. If the reproducible number is below the reproduction number of tape or the maximum reproducible time is below the reproduction time of tape, the enable erase signal is turned on to prohibit the reproduction of the copied program. By doing so, part of or the whole program recorded on cassette tape is erased to make copy and reproduction impossible in step 137.

Here, the current time is transmitted to the user by

a provider along with a program. In this case, the copyright protection system implements limited time reproduction using transmitted time information. In this method, the program provider manages the whole users' time so that time modulation by a user cannot occur. Therefore, this is very secure.

The bit stream transmitted in the first embodiment of the audio/video signal transmission step contains ECM and EMM. Part of the control word may be contained in the CPTC information. Its remainder may be contained in the ECM or EMM. The whole control word is contained in the ECM or EMM.

The audio/video signal containing the control word and transmitted according to the audio/video signal transmission step is received according to another embodiment of the audio/video reception step.

Referring to Fig. 10, the second embodiment of the audio/video reception step comprises the steps of filtering (110) the transmitted bit stream and decrypting (111) the CPTC information and control word; filtering (118) the control word; analyzing (113 and 114) the CPTC information to generate a control word and a signal for controlling the protection of copyright and to update the CPTC information; deciding (115) whether to allow recording according to the signal for controlling the protection of copyright to record the scrambled and transmitted bit stream on cassette tape; and descrambling and decoding (116 and 117) the transmitted bit stream in control words and outputting an audio/video signal.

In other words, the bit stream transmitted in the audio/video signal transmission step is filtered and the CPTC information and control word are decrypted in steps 110 and 111. The control word is filtered in step 118. The decrypted CPTC information is analyzed to generate the control word and the signal for controlling the protection of copyright, and the CPTC information is updated in steps 113 and 114. Whether to allow recording is determined by the generated signal for controlling the protection of copyright so that the scrambled and transmitted bit stream is recorded on cassette tape in step 115. Then, the transmitted bit stream is descrambled and decoded in control words and output as an audio/video signal in steps 116 and 117.

Referring to Fig. 11, in the same manner as the first embodiment of the audio/video reception step, the CPTC information analyzing step comprises the steps of: generating the control words; detecting (130, 131, 132 and 133) the permissible generation of the permissible generational field for limiting the available number of copy of a program of the CPTC information and the present generation of the present generational field indicating the present generation of the program copied, to thereby perform copy-impossible and update the CPTC information; and detecting (134, 135, 136 and 137) the reproducible number of the reproducible number field for limiting the number of reproduction of copied programs of the CPTC information, the maximum reproducible time of the maximum reproducible time field for limiting time to reproduce the copied program, and the number and time of reproduction of tape, to thereby process reproduction-impossible.

The copying number limiting step comprises the steps of: comparing (130) the permissible generation of the permissible generational field and the present generation of the present generational field and deciding whether the permissible generation is below the present generation; if the permissible generation is below the present generation, generating (131) an output disable signal to make copying impossible and destroying the control word; and if the permissible generation is not below the present generation, increasing (132) the present invention by '1' and recording the result on cassette tape. If the permissible generation is not below the present generation, the CPTC information is updated in step 133, instead of increasing the present generation by '1.'

The reproduction limiting step comprises the steps of: comparing the reproducible number of the reproducible number field and the reproduction number of tape and deciding (134) whether the reproducible number is below the reproduction number of tape; if the reproducible number is not below the reproduction number of tape, comparing the maximum reproducible time and reproduction time of tape, and deciding (135) whether the maximum reproducible time is below the reproduction time of tape; if the maximum reproducible time is not below reproduction time of tape, turning off (136) an enable erase signal to thereby enable the copied program to be reproduced; if the reproducible number is below the reproduction number of tape or the maximum reproducible time is below the reproduction time of tape, turning on (137) the enable erase signal to make the reproduction of the copied program impossible so that part of or the whole program recorded on cassette tape is erased.

Another embodiment of the audio/video signal transmission step is to transmit an audio/video signal containing the CA information for conditional access. This, having the illegal reception and copy protection functions, is used in case that a program which can be provided to limited viewers is transmitted.

As shown in Fig. 9, the second embodiment of the audio/video signal transmission step comprises the steps of: encoding (100) the audio/video bit stream; generating (105) a control word for scrambling; scrambling (104) for the encoded audio/video bit stream using the generated control word; generating (102) CPTC information for illegal view/copy protection; generating (101) CA information for conditional reception; encrypting (103) for encrypting the CPTC information and CA information; and multiplexing and transmitting (106) the scrambled audio/video bit stream and encrypted CPTC information and CA information.

In other words, in step 100, the audio/video bit stream is encoded. In step 105, the control word for scrambling is generated. In step 104, the encoded au-

dio/video bit stream is scrambled using the generated control word. In step 102, the CPTC information for illegal view/copy protection is generated. In step 101, CA information for conditional reception is generated. In step 103, the CPTC information and CA information are encrypted using the generated control word. The scrambled audio/video bit stream, encrypted CPTC information and CA information are multiplexed and transmitted through a transmission medium in step 106. The audio/video signal transmitted through the second embodiment of the audio/video signal transmission step is received through the second embodiment of the audio/video reception step.

Referring to Fig. 10, the second embodiment of the audio/video reception step comprises the steps of: filtering (110) the transmitted bit stream and decrypting (111) the CPTC information; analyzing (112, 113 and 114) the CPTC information and CA information to generate a control word and a signal for controlling the protection of copyright and to update the CPTC information; deciding (115) whether to allow recording according to the signal for controlling the protection of copyright to record the scrambled and transmitted bit stream on cassette tape; and descrambling and decoding (116 and 117) the transmitted bit stream and outputting an audio/video signal.

Referring to Fig. 11, in the same manner as the first embodiment of the audio/video reception step, the CPTC information analyzing step comprises the steps of: generating a control word; detecting (130, 131, 132 and 133) the permissible generation of the permissible generational field for limiting the available number of copy of a program of the CPTC information and the present generation of the present generational field indicating the present generation of the program copied, to thereby perform copy-impossible and update the CPTC information; and detecting (134, 135, 136 and 137) the reproducible number of the reproducible number field for limiting the number of reproduction of copied programs of the CPTC information, the maximum reproducible time of the maximum reproducible time field for limiting time to reproduce the copied program, and the number and time of reproduction of tape, to thereby process reproduction-impossible.

In the same manner as the first embodiment of the audio/video reception step, the copying number limiting step comprises the steps of: comparing (130) the permissible generation of the permissible generational field and the present generation of the present generational field and deciding whether the permissible generation is below the present generation; if the permissible generation is below the present generation, generating (131) an output disable signal to make copying impossible and destroying the control word; and if the permissible generation is not below the present generation, increasing (132) the present invention by '1' and recording the result on cassette tape. If the permissible generation is not below the present generation, the CPTC information is updated in step 133.

The reproduction limiting step comprises the steps of: comparing the reproducible number of the reproducible number field and the reproduction number of tape and deciding (134) whether the reproducible number is below the reproduction number of tape; if the reproducible number is not below the reproduction number of tape, comparing the maximum reproducible time and reproduction time of tape, and deciding (135) whether the maximum reproducible time is below the reproduction time of tape; if the maximum reproducible time is not below reproduction time of tape, turning off (136) an enable erase signal to thereby enable the copied program to be reproduced; if the reproducible number is below the reproduction number of tape or the maximum reproducible time is below the reproduction time of tape, turning on (137) the enable erase signal to make the reproduction of the copied program impossible so that part of or the whole program recorded on cassette tape is erased.

The bit stream transmitted in the second embodiment of the audio/video signal transmission step contains ECM and EMM. Part of the control word may be contained in the CPTC information. Its remainder may be contained in the ECM or EMM. The whole control word is contained in the ECM or EMM.

The audio/video signal containing the control word and transmitted according to the audio/video signal transmission step is received according to another embodiment of the audio/video reception step. The audio/video signal transmitted in the audio/video signal transmission step containing the control word is received according to still another embodiment of the audio/video reception step.

Referring to Fig. 10, the third embodiment of the audio/video reception step comprises the steps of: filtering (110) the transmitted bit stream and decrypting (111) the CPTC information and CA information; analyzing (112, 113, 114 and 118) the CPTC information and CA information and filtering the control word to generate a control word and a signal for controlling the protection of copyright and to update the CPTC information; deciding (115) whether to allow recording according to the signal for controlling the protection of copyright to record the scrambled and transmitted bit stream on cassette tape; and descrambling and decoding (116 and 117) the transmitted bit stream and outputting an audio/video signal.

Referring to Fig. 11, in the same manner as the first embodiment of the audio/video reception step, the CPTC information analyzing step comprises the steps of: generating the control words; detecting (130, 131, 132 and 133) the permissible generation of the permissible generational field for limiting the available number of copy of a program of the CPTC information and the present generation of the present generational field indicating the present generation of the program copied, to thereby perform copy-impossible and update the CPTC information; and detecting (134, 135, 136 and