

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.
(Also referred to as FORM PTO-1465)

REQUEST FOR *EX PARTE* REEXAMINATION TRANSMITTAL FORM

Address to:
**Mail Stop *Ex Parte* Reexam
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450**

Attorney Docket No.: _____

Date: 05-16-2018

1. This is a request for *ex parte* reexamination pursuant to 37 CFR 1.510 of patent number 9104842 issued 08-11-2015. The request is made by:
 patent owner. third party requester.
2. The name and address of the person requesting reexamination is:
Fisch Sigler LLP
5301 Wisconsin Avenue, NW, Fourth Floor
Washington, DC 20015
3. Requester asserts small entity status (37 CFR 1.27) or certifies micro entity status (37 CFR 1.29). Only a patent owner requester can certify micro entity status. Form PTO/SB/15A or B must be attached to certify micro entity status.
4. a. A check in the amount of \$_____ is enclosed to cover the reexamination fee, 37 CFR 1.20(c)(1);
 b. The Director is hereby authorized to charge the fee as set forth in 37 CFR 1.20(c)(1) to Deposit Account No. _____;
 c. Payment by credit card. Form PTO-2038 is attached; **or**
 d. Payment made via EFS-Web.
5. Any refund should be made by check or credit to Deposit Account No. _____.
37 CFR 1.26(c). If payment is made by credit card, refund must be to credit card account.
6. A copy of the patent to be reexamined having a double column format on one side of a separate paper is enclosed. 37 CFR 1.510(b)(4).
7. CD-ROM or CD-R in duplicate, Computer Program (Appendix) or large table
 Landscape Table on CD
8. Nucleotide and/or Amino Acid Sequence Submission
If applicable, items a. – c. are required.
 - a. Computer Readable Form (CRF)
 - b. Specification Sequence Listing on:
 - i. CD-ROM (2 copies) or CD-R (2 copies); **or**
 - ii. paper
 - c. Statements verifying identity of above copies
9. A copy of any disclaimer, certificate of correction or reexamination certificate issued in the patent is included.
10. Reexamination of claim(s) 11, 12, 13, and 14 is requested.
11. A copy of every patent or printed publication relied upon is submitted herewith including a listing thereof on Form PTO/SB/08, PTO-1449, or equivalent.
12. An English language translation of all necessary and pertinent non-English language patents and/or printed publications is included.

[Page 1 of 2]

This collection of information is required by 37 CFR 1.510. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) a request for reexamination. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.11 and 1.14. This collection is estimated to take 18 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. **SEND TO: Mail Stop *Ex Parte* Reexam, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.**

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

13. The attached detailed request includes at least the following items:
- a. A statement identifying each substantial new question of patentability based on prior patents and printed publications. 37 CFR 1.510(b)(1).
 - b. An identification of every claim for which reexamination is requested, and a detailed explanation of the pertinency and manner of applying the cited art to every claim for which reexamination is requested. 37 CFR 1.510(b)(2).
14. A proposed amendment is included (only where the patent owner is the requester). 37 CFR 1.510(e).
15. It is certified that the statutory estoppel provisions of 35 U.S.C. 315(e)(1) or 35 U.S.C. 325(e)(1) do not prohibit requester from filing this *ex parte* reexamination request. 37 CFR 1.510(b)(6).
16. a. It is certified that a copy of this request (if filed by other than the patent owner) has been served in its entirety on the patent owner as provided in 37 CFR 1.33(c).
 The name and address of the party served and the date of service are:
Wistaria Trading LTD
Clarendon House, 2 Church Street, Hamilton HM 11, Bermuda
 Date of Service: May 16, 2018; or
- b. A duplicate copy is enclosed since service on patent owner was not possible. An explanation of the efforts made to serve patent owner is **attached**. See MPEP 2220.

17. Correspondence Address: Direct all communication about the reexamination to:

The address associated with Customer Number:

OR

Firm or Individual Name Fisch Sigler, LLP

Address _____

City Washington	State DC	Zip 20015
Country United States		
Telephone (202) 362-3524	Email Joe.Edell@fischllp.com	

18. The patent is currently the subject of the following concurrent proceeding(s):
- a. Copending reissue Application No. _____
 - b. Copending reexamination Control No. _____
 - c. Copending Interference No. _____
 - d. Copending litigation styled:
Blue Spike, LLC v. Juniper Networks, Inc., 6:17-cv-00016-KNM (ED. Tex. 2017)

WARNING: Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.

/Joseph F. Edell/ 05-16-2018
 Authorized Signature Date

Joseph F. Edell 67,625 For Patent Owner Requester
 Typed/Printed Name Registration No. For Third Party Requester

Privacy Act Statement

The **Privacy Act of 1974 (P.L. 93-579)** requires that you be given certain information in connection with your submission of the attached form related to a patent application or patent. Accordingly, pursuant to the requirements of the Act, please be advised that: (1) the general authority for the collection of this information is 35 U.S.C. 2(b)(2); (2) furnishing of the information solicited is voluntary; and (3) the principal purpose for which the information is used by the U.S. Patent and Trademark Office is to process and/or examine your submission related to a patent application or patent. If you do not furnish the requested information, the U.S. Patent and Trademark Office may not be able to process and/or examine your submission, which may result in termination of proceedings or abandonment of the application or expiration of the patent.

The information provided by you in this form will be subject to the following routine uses:

1. The information on this form will be treated confidentially to the extent allowed under the Freedom of Information Act (5 U.S.C. 552) and the Privacy Act (5 U.S.C. 552a). Records from this system of records may be disclosed to the Department of Justice to determine whether disclosure of these records is required by the Freedom of Information Act.
2. A record from this system of records may be disclosed, as a routine use, in the course of presenting evidence to a court, magistrate, or administrative tribunal, including disclosures to opposing counsel in the course of settlement negotiations.
3. A record in this system of records may be disclosed, as a routine use, to a Member of Congress submitting a request involving an individual, to whom the record pertains, when the individual has requested assistance from the Member with respect to the subject matter of the record.
4. A record in this system of records may be disclosed, as a routine use, to a contractor of the Agency having need for the information in order to perform a contract. Recipients of information shall be required to comply with the requirements of the Privacy Act of 1974, as amended, pursuant to 5 U.S.C. 552a(m).
5. A record related to an International Application filed under the Patent Cooperation Treaty in this system of records may be disclosed, as a routine use, to the International Bureau of the World Intellectual Property Organization, pursuant to the Patent Cooperation Treaty.
6. A record in this system of records may be disclosed, as a routine use, to another federal agency for purposes of National Security review (35 U.S.C. 181) and for review pursuant to the Atomic Energy Act (42 U.S.C. 218(c)).
7. A record from this system of records may be disclosed, as a routine use, to the Administrator, General Services, or his/her designee, during an inspection of records conducted by GSA as part of that agency's responsibility to recommend improvements in records management practices and programs, under authority of 44 U.S.C. 2904 and 2906. Such disclosure shall be made in accordance with the GSA regulations governing inspection of records for this purpose, and any other relevant (*i.e.*, GSA or Commerce) directive. Such disclosure shall not be used to make determinations about individuals.
8. A record from this system of records may be disclosed, as a routine use, to the public after either publication of the application pursuant to 35 U.S.C. 122(b) or issuance of a patent pursuant to 35 U.S.C. 151. Further, a record may be disclosed, subject to the limitations of 37 CFR 1.14, as a routine use, to the public if the record was filed in an application which became abandoned or in which the proceedings were terminated and which application is referenced by either a published application, an application open to public inspection or an issued patent.
9. A record from this system of records may be disclosed, as a routine use, to a Federal, State, or local law enforcement agency, if the USPTO becomes aware of a violation or potential violation of law or regulation.

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

United States Patent No.: 9,104,842
Inventors: Scott A. Moskowitz
Formerly Application No.: 11/895,388
Issue Date: August 11, 2015
Filing Date: August 24, 2007
Former Examiner: Izzuna Okeke
Former Group Art Unit: 2497

For: DATA PROTECTION METHOD AND DEVICE

MAIL STOP *EX PARTE* REEXAM
Central Reexamination Unit
Office of Patent Legal Administration
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

CERTIFICATE OF SERVICE

It is certified that, pursuant to 37 C.F.R. § 1.510(b)(5), copies of the following documents have been served in their entireties on the patent owner at the correspondence address of record as provide for in 37 C.F.R. § 1.33(c):

1. Request for *Ex Parte* Reexamination of U.S. Patent No. 9,104,842 Transmittal Form, PTO/SB/57.
 2. Request for *Ex Parte* Reexamination of U.S. Patent No. 9,104,842 Pursuant to 35 U.S.C. § 302 and 37 C.F.R. § 1.510 and accompanying exhibits:
 - Exhibit 1 U.S. Patent No. 9,104,842 (“the ‘842 patent”)
 - Exhibit 2 File History of U.S. Patent No. 9,104,842 (“the ‘842 Prosecution History”) (other than the prior art of record) (consecutive page numbers added for ease of citation)
- Prior Art
- Exhibit 3 U.S. Patent No. 5,933,497 (“Beetcher”)
 - Exhibit 4 JP Patent No. 05-334702 (“Beetcher”)
 - Exhibit 5 JP Patent No. 05-334702 Translation (“Beetcher Translation”)
 - Exhibit 6 International Application No. WO9,726,732 (“Cooperman”)
 - Exhibit 7 U.S. Patent No. 5,935,243 (“Hasebe”)

Exhibit 8 [INTENTIONALLY LEFT BLANK]

Expert Materials

Exhibit 9 Declaration of Cláudio T. Silva In Support of Request for *Ex Parte* Reexamination of U.S. Patent No. 9,104,842

Exhibit 10 Curriculum Vitae of Cláudio T. Silva

Claim Charts

Exhibit 11 Blue Spike LLC's Proposed Terms for Construction for U.S. Patent 9,104,842 ("the '842 patent") *Blue Spike, LLC. v. Juniper Networks, Inc.*, Case No. Case No. 6:17-cv-00016-KNM (EDTX)

3. Information Disclosure Statement, PTO/SB/08, listing references cited in the Request *for Ex Parte* Reexamination of U.S. Patent No. 9,104,842 pursuant to 35 U.S.C. § 302 and 37 C.F.R. § 1.510
4. A copy of U.S. Patent No. 9,104,842.
5. A copy of this Certificate of Service.

The copies have been served on May 16, 2018 by causing the aforementioned documents to be deposited in the United States Postal Service as first class mail postage pre-paid in an envelope addressed to:

Wistaria Trading LTD,
Calrendon House, 2 Church Street
Hamilton HM 11
Bermuda

Blue Spike, LLC
Garteiser Honea
119 W Ferguson
Tyler, TX 75702

Neifeld IP Law, PC
5400 Shawnee Road
Suite 310
Alexandria, VA 22312-2300

/Joseph F. Edell/

Joseph F. Edell
Attorney for Requester
Reg. No. 67,625

May 16, 2018

Requester:
Fisch Sigler LLP
5301 Wisconsin Ave. NW
Fourth Floor
Washington, DC 200015

Electronic Patent Application Fee Transmittal

Application Number:				
Filing Date:				
Title of Invention:	DATA PROTECTION METHOD AND DEVICE			
First Named Inventor/Applicant Name:	Scott Moskowitz			
Filer:	Joseph Edell			
Attorney Docket Number:				
Filed as Large Entity				
Filing Fees for ex parte reexam				
Description	Fee Code	Quantity	Amount	Sub-Total in USD(\$)
Basic Filing:				
EX PARTE REEXAMINATION (1.510(A)) NON-STREAMLINED	1812	1	12000	12000
Pages:				
Claims:				
Miscellaneous-Filing:				
Petition:				
Patent-Appeals-and-Interference:				
Post-Allowance-and-Post-Issuance:				

Description	Fee Code	Quantity	Amount	Sub-Total in USD(\$)
Extension-of-Time:				
Miscellaneous:				
Total in USD (\$)				12000

Electronic Acknowledgement Receipt

EFS ID:	32645771
Application Number:	90014138
International Application Number:	
Confirmation Number:	7638
Title of Invention:	DATA PROTECTION METHOD AND DEVICE
First Named Inventor/Applicant Name:	Scott Moskowitz
Correspondence Address:	Joseph F. Edell - 5301 Wisconsin Avenue NW - Washington DC 20015 US - -
Filer:	Joseph Edell
Filer Authorized By:	
Attorney Docket Number:	
Receipt Date:	16-MAY-2018
Filing Date:	
Time Stamp:	18:42:59
Application Type:	Reexam (Third Party)

Payment information:

Submitted with Payment	yes
Payment Type	CARD
Payment was successfully received in RAM	\$ 12000

RAM confirmation Number	051718INTEFSW18464300				
Deposit Account					
Authorized User					
The Director of the USPTO is hereby authorized to charge indicated fees and credit any overpayment as follows:					
File Listing:					
Document Number	Document Description	File Name	File Size(Bytes)/ Message Digest	Multi Part /.zip	Pages (if appl.)
1	Application Data Sheet	WebADS.pdf	118072 69095c289f22e27891a2eba563449f2b63f2dbc4	no	7
Warnings:					
Information:					
2	Reexam - Affidavit/Decl/Exhibit Filed by 3rd Party	Request.pdf	9778204 00796b026a25ae1f7657a06a7f48615b5f808584	no	129
Warnings:					
Information:					
3	Reexam - Affidavit/Decl/Exhibit Filed by 3rd Party	Exhibit1.pdf	3314824 e88acacc5b481e9acbcf3786bcf3a0276ae82a75	no	21
Warnings:					
The page size in the PDF is too large. The pages should be 8.5 x 11 or A4. If this PDF is submitted, the pages will be resized upon entry into the Image File Wrapper and may affect subsequent processing					
Information:					
4	Reexam - Affidavit/Decl/Exhibit Filed by 3rd Party	Exhibit3.pdf	1821085 dfd01b4ea3a4794f3d85875e166edfeb9c0d1528	no	20
Warnings:					
The page size in the PDF is too large. The pages should be 8.5 x 11 or A4. If this PDF is submitted, the pages will be resized upon entry into the Image File Wrapper and may affect subsequent processing					
Information:					

5	Reexam - Affidavit/Decl/Exhibit Filed by 3rd Party	Exhibit4.pdf	1848899	no	21
			979e6c2347a999bb2d73a5c32982feb18ebb6c6b22		
Warnings:					
The page size in the PDF is too large. The pages should be 8.5 x 11 or A4. If this PDF is submitted, the pages will be resized upon entry into the Image File Wrapper and may affect subsequent processing					
Information:					
6	Reexam - Affidavit/Decl/Exhibit Filed by 3rd Party	Exhibit5.pdf	2545083	no	20
			a13b232d3f94af4fe5d924d29fabbd9ea56d2261		
Warnings:					
The page size in the PDF is too large. The pages should be 8.5 x 11 or A4. If this PDF is submitted, the pages will be resized upon entry into the Image File Wrapper and may affect subsequent processing					
Information:					
7	Reexam - Affidavit/Decl/Exhibit Filed by 3rd Party	Exhibit6.pdf	1599448	no	23
			3a9c1e141e15472429861731665b7d749118135f		
Warnings:					
The page size in the PDF is too large. The pages should be 8.5 x 11 or A4. If this PDF is submitted, the pages will be resized upon entry into the Image File Wrapper and may affect subsequent processing					
Information:					
8	Reexam - Affidavit/Decl/Exhibit Filed by 3rd Party	Exhibit7.pdf	1463313	no	15
			bdbb3dfd5b355e25f8aed1a4762673f1d8e46436		
Warnings:					
The page size in the PDF is too large. The pages should be 8.5 x 11 or A4. If this PDF is submitted, the pages will be resized upon entry into the Image File Wrapper and may affect subsequent processing					
Information:					
9	Reexam - Affidavit/Decl/Exhibit Filed by 3rd Party	Exhibit8.pdf	9008	no	2
			5f7ad6b3f6ea2ad4234b760c80117ee023cae683		
Warnings:					
The page size in the PDF is too large. The pages should be 8.5 x 11 or A4. If this PDF is submitted, the pages will be resized upon entry into the Image File Wrapper and may affect subsequent processing					
Information:					
10	Reexam - Affidavit/Decl/Exhibit Filed by 3rd Party	Exhibit9.pdf	4774536	no	128
			115478734b8b236254506c66735de6f521215fc0		
Warnings:					

Information:					
11	Reexam - Affidavit/Decl/Exhibit Filed by 3rd Party	Exhibit10.pdf	4060940 e537f7ab45da08718e43d29859017dff785772e9	no	39
Warnings:					
The page size in the PDF is too large. The pages should be 8.5 x 11 or A4. If this PDF is submitted, the pages will be resized upon entry into the Image File Wrapper and may affect subsequent processing					
Information:					
12	Reexam - Affidavit/Decl/Exhibit Filed by 3rd Party	Exhibit11.pdf	7568429 ce18b6d11d31d161a891597075b6f61455c616e3	no	62
Warnings:					
The page size in the PDF is too large. The pages should be 8.5 x 11 or A4. If this PDF is submitted, the pages will be resized upon entry into the Image File Wrapper and may affect subsequent processing					
Information:					
13	Information Disclosure Statement (IDS) Form (SB08)	IDS.pdf	330957 b7638d3415f4a62f12d54814e4acef8a563ffbed	no	4
Warnings:					
Information:					
This is not an USPTO supplied IDS fillable form					
The page size in the PDF is too large. The pages should be 8.5 x 11 or A4. If this PDF is submitted, the pages will be resized upon entry into the Image File Wrapper and may affect subsequent processing					
14	Transmittal of New Application	TF.pdf	318599 dd247fdeb9fc602f0cb715d79b04b6426e974ab7	no	3
Warnings:					
The page size in the PDF is too large. The pages should be 8.5 x 11 or A4. If this PDF is submitted, the pages will be resized upon entry into the Image File Wrapper and may affect subsequent processing					
Information:					
15	Reexam - Affidavit/Decl/Exhibit Filed by 3rd Party	COS.pdf	121696 c59f1ac39bcb8ce2d2db4f03c66650dab2bc0e38	no	3
Warnings:					
The page size in the PDF is too large. The pages should be 8.5 x 11 or A4. If this PDF is submitted, the pages will be resized upon entry into the Image File Wrapper and may affect subsequent processing					
Information:					

16	Reexam - Affidavit/Decl/Exhibit Filed by 3rd Party	Exhibit2P1.pdf	17144565 31d0b2094983e85c1a7b0974c3e07d21cad2a26a	no	478
Warnings:					
Information:					
17	Reexam - Affidavit/Decl/Exhibit Filed by 3rd Party	Exhibit2P2.pdf	12839391 808d7ebdb805a9e294897e85b205441ec6a81adf	no	481
Warnings:					
Information:					
18	Fee Worksheet (SB06)	fee-info.pdf	29589 a8ad2eac018546a3dfc8074a51e83e7baf8aa6c7	no	2
Warnings:					
Information:					
Total Files Size (in bytes):				69686638	
<p>This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.</p> <p><u>New Applications Under 35 U.S.C. 111</u> If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.</p> <p><u>National Stage of an International Application under 35 U.S.C. 371</u> If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.</p> <p><u>New International Application Filed with the USPTO as a Receiving Office</u> If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.</p>					

Application Data Sheet 37 CFR 1.76		Attorney Docket Number	
		Application Number	
Title of Invention	DATA PROTECTION METHOD AND DEVICE		
<p>The application data sheet is part of the provisional or nonprovisional application for which it is being submitted. The following form contains the bibliographic data arranged in a format specified by the United States Patent and Trademark Office as outlined in 37 CFR 1.76. This document may be completed electronically and submitted to the Office in electronic format using the Electronic Filing System (EFS) or the document may be printed and included in a paper filed application.</p>			

Secrecy Order 37 CFR 5.2:

<input type="checkbox"/>	Portions or all of the application associated with this Application Data Sheet may fall under a Secrecy Order pursuant to 37 CFR 5.2 (Paper filers only. Applications that fall under Secrecy Order may not be filed electronically.)
--------------------------	---

Inventor Information:

Inventor 1				
Legal Name				
Prefix	Given Name	Middle Name	Family Name	Suffix
	Scott		Moskowitz	
Residence Information (Select One) <input checked="" type="radio"/> US Residency <input type="radio"/> Non US Residency <input type="radio"/> Active US Military Service				
City	Sunny Isles Beach	State/Province	FL	Country of Residence ⁱ US
Mailing Address of Inventor:				
Address 1	16711 Collins Avenue #2505			
Address 2				
City	Sunny Isles Beach	State/Province	DC	
Postal Code	20015	Country ⁱ	US	
All Inventors Must Be Listed - Additional Inventor Information blocks may be generated within this form by selecting the Add button.				<input type="button" value="Add"/>

Correspondence Information:

Enter either Customer Number or complete the Correspondence Information section below. For further information see 37 CFR 1.33(a).				
<input checked="" type="checkbox"/> An Address is being provided for the correspondence information of this application.				
Name 1	Joseph F. Edell	Name 2		
Address 1	5301 Wisconsin Avenue NW			
Address 2				
City	Washington	State/Province	DC	
Country ⁱ	US	Postal Code	20015	
Phone Number			Fax Number	
Email Address			<input type="button" value="Add Email"/>	<input type="button" value="Remove Email"/>

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it contains a valid OMB control number.

Application Data Sheet 37 CFR 1.76		Attorney Docket Number	
		Application Number	
Title of Invention	DATA PROTECTION METHOD AND DEVICE		

Application Information:

Title of the Invention	DATA PROTECTION METHOD AND DEVICE		
Attorney Docket Number		Small Entity Status Claimed	<input type="checkbox"/>
Application Type	Nonprovisional		
Subject Matter	Utility		
Total Number of Drawing Sheets (if any)		Suggested Figure for Publication (if any)	

Filing By Reference:

Only complete this section when filing an application by reference under 35 U.S.C. 111(c) and 37 CFR 1.57(a). Do not complete this section if application papers including a specification and any drawings are being filed. Any domestic benefit or foreign priority information must be provided in the appropriate section(s) below (i.e., "Domestic Benefit/National Stage Information" and "Foreign Priority Information").

For the purposes of a filing date under 37 CFR 1.53(b), the description and any drawings of the present application are replaced by this reference to the previously filed application, subject to conditions and requirements of 37 CFR 1.57(a).

Application number of the previously filed application	Filing date (YYYY-MM-DD)	Intellectual Property Authority or Country

Publication Information:

Request Early Publication (Fee required at time of Request 37 CFR 1.219)

Request Not to Publish. I hereby request that the attached application not be published under 35 U.S.C. 122(b) and certify that the invention disclosed in the attached application **has not and will not** be the subject of an application filed in another country, or under a multilateral international agreement, that requires publication at eighteen months after filing.

Representative Information:

Representative information should be provided for all practitioners having a power of attorney in the application. Providing this information in the Application Data Sheet does not constitute a power of attorney in the application (see 37 CFR 1.32). Either enter Customer Number or complete the Representative Name section below. If both sections are completed the customer Number will be used for the Representative Information during processing.

Please Select One:	<input checked="" type="radio"/> Customer Number	<input type="radio"/> US Patent Practitioner	<input type="radio"/> Limited Recognition (37 CFR 11.9)
Customer Number			
Prefix	Given Name	Middle Name	Family Name
Registration Number			

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it contains a valid OMB control number.

Application Data Sheet 37 CFR 1.76		Attorney Docket Number	
		Application Number	
Title of Invention	DATA PROTECTION METHOD AND DEVICE		

Prefix	Given Name	Middle Name	Family Name	Suffix	<input type="button" value="Remove"/>
Registration Number					
Additional Representative Information blocks may be generated within this form by selecting the Add button.					

Domestic Benefit/National Stage Information:

This section allows for the applicant to either claim benefit under 35 U.S.C. 119(e), 120, 121, 365(c), or 386(c) or indicate National Stage entry from a PCT application. Providing benefit claim information in the Application Data Sheet constitutes the specific reference required by 35 U.S.C. 119(e) or 120, and 37 CFR 1.78. When referring to the current application, please leave the "Application Number" field blank.

Prior Application Status			<input type="button" value="Remove"/>
Application Number	Continuity Type	Prior Application Number	Filing or 371(c) Date (YYYY-MM-DD)
Additional Domestic Benefit/National Stage Data may be generated within this form by selecting the Add button.			

Foreign Priority Information:

This section allows for the applicant to claim priority to a foreign application. Providing this information in the application data sheet constitutes the claim for priority as required by 35 U.S.C. 119(b) and 37 CFR 1.55. When priority is claimed to a foreign application that is eligible for retrieval under the priority document exchange program (PDX) the information will be used by the Office to automatically attempt retrieval pursuant to 37 CFR 1.55(i)(1) and (2). Under the PDX program, applicant bears the ultimate responsibility for ensuring that a copy of the foreign application is received by the Office from the participating foreign intellectual property office, or a certified copy of the foreign priority application is filed, within the time period specified in 37 CFR 1.55(g)(1).

Application Number	Country ¹	Filing Date (YYYY-MM-DD)	Access Code ¹ (if applicable)
Additional Foreign Priority Data may be generated within this form by selecting the Add button.			

Statement under 37 CFR 1.55 or 1.78 for AIA (First Inventor to File) Transition Applications

<p>This application (1) claims priority to or the benefit of an application filed before March 16, 2013 and (2) also contains, or contained at any time, a claim to a claimed invention that has an effective filing date on or after March 16, 2013.</p> <p><input type="checkbox"/> NOTE: By providing this statement under 37 CFR 1.55 or 1.78, this application, with a filing date on or after March 16, 2013, will be examined under the first inventor to file provisions of the AIA.</p>
--

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it contains a valid OMB control number.

Application Data Sheet 37 CFR 1.76	Attorney Docket Number	
	Application Number	
Title of Invention	DATA PROTECTION METHOD AND DEVICE	

Authorization or Opt-Out of Authorization to Permit Access:

When this Application Data Sheet is properly signed and filed with the application, applicant has provided written authority to permit a participating foreign intellectual property (IP) office access to the instant application-as-filed (see paragraph A in subsection 1 below) and the European Patent Office (EPO) access to any search results from the instant application (see paragraph B in subsection 1 below).

Should applicant choose not to provide an authorization identified in subsection 1 below, applicant **must opt-out** of the authorization by checking the corresponding box A or B or both in subsection 2 below.

NOTE: This section of the Application Data Sheet is **ONLY** reviewed and processed with the **INITIAL** filing of an application. After the initial filing of an application, an Application Data Sheet cannot be used to provide or rescind authorization for access by a foreign IP office(s). Instead, Form PTO/SB/39 or PTO/SB/69 must be used as appropriate.

1. Authorization to Permit Access by a Foreign Intellectual Property Office(s)

A. Priority Document Exchange (PDX) - Unless box A in subsection 2 (opt-out of authorization) is checked, the undersigned hereby **grants the USPTO authority** to provide the European Patent Office (EPO), the Japan Patent Office (JPO), the Korean Intellectual Property Office (KIPO), the State Intellectual Property Office of the People's Republic of China (SIPO), the World Intellectual Property Organization (WIPO), and any other foreign intellectual property office participating with the USPTO in a bilateral or multilateral priority document exchange agreement in which a foreign application claiming priority to the instant patent application is filed, access to: (1) the instant patent application-as-filed and its related bibliographic data, (2) any foreign or domestic application to which priority or benefit is claimed by the instant application and its related bibliographic data, and (3) the date of filing of this Authorization. See 37 CFR 1.14(h)(1).

B. Search Results from U.S. Application to EPO - Unless box B in subsection 2 (opt-out of authorization) is checked, the undersigned hereby **grants the USPTO authority** to provide the EPO access to the bibliographic data and search results from the instant patent application when a European patent application claiming priority to the instant patent application is filed. See 37 CFR 1.14(h)(2).

The applicant is reminded that the EPO's Rule 141(1) EPC (European Patent Convention) requires applicants to submit a copy of search results from the instant application without delay in a European patent application that claims priority to the instant application.

2. Opt-Out of Authorizations to Permit Access by a Foreign Intellectual Property Office(s)

A. Applicant **DOES NOT** authorize the USPTO to permit a participating foreign IP office access to the instant application-as-filed. If this box is checked, the USPTO will not be providing a participating foreign IP office with any documents and information identified in subsection 1A above.

B. Applicant **DOES NOT** authorize the USPTO to transmit to the EPO any search results from the instant patent application. If this box is checked, the USPTO will not be providing the EPO with search results from the instant application.

NOTE: Once the application has published or is otherwise publicly available, the USPTO may provide access to the application in accordance with 37 CFR 1.14.

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it contains a valid OMB control number.

Application Data Sheet 37 CFR 1.76		Attorney Docket Number	
		Application Number	
Title of Invention	DATA PROTECTION METHOD AND DEVICE		

Applicant Information:

Providing assignment information in this section does not substitute for compliance with any requirement of part 3 of Title 37 of CFR to have an assignment recorded by the Office.

Applicant 1

If the applicant is the inventor (or the remaining joint inventor or inventors under 37 CFR 1.45), this section should not be completed. The information to be provided in this section is the name and address of the legal representative who is the applicant under 37 CFR 1.43; or the name and address of the assignee, person to whom the inventor is under an obligation to assign the invention, or person who otherwise shows sufficient proprietary interest in the matter who is the applicant under 37 CFR 1.46. If the applicant is an applicant under 37 CFR 1.46 (assignee, person to whom the inventor is obligated to assign, or person who otherwise shows sufficient proprietary interest) together with one or more joint inventors, then the joint inventor or inventors who are also the applicant should be identified in this section.

Assignee
 Legal Representative under 35 U.S.C. 117
 Joint Inventor

Person to whom the inventor is obligated to assign.
 Person who shows sufficient proprietary interest

If applicant is the legal representative, indicate the authority to file the patent application, the inventor is:

Name of the Deceased or Legally Incapacitated Inventor:

If the Applicant is an Organization check here.

Prefix	Given Name	Middle Name	Family Name	Suffix

Mailing Address Information For Applicant:

Address 1			
Address 2			
City		State/Province	
Country ¹		Postal Code	
Phone Number		Fax Number	
Email Address			

Additional Applicant Data may be generated within this form by selecting the Add button.

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it contains a valid OMB control number.

Application Data Sheet 37 CFR 1.76		Attorney Docket Number	
		Application Number	
Title of Invention	DATA PROTECTION METHOD AND DEVICE		

Assignee Information including Non-Applicant Assignee Information:

Providing assignment information in this section does not substitute for compliance with any requirement of part 3 of Title 37 of CFR to have an assignment recorded by the Office.

Assignee 1				
Complete this section if assignee information, including non-applicant assignee information, is desired to be included on the patent application publication. An assignee-applicant identified in the "Applicant Information" section will appear on the patent application publication as an applicant. For an assignee-applicant, complete this section only if identification as an assignee is also desired on the patent application publication.				
If the Assignee or Non-Applicant Assignee is an Organization check here. <input type="checkbox"/>				
Prefix	Given Name	Middle Name	Family Name	Suffix
Mailing Address Information For Assignee including Non-Applicant Assignee:				
Address 1				
Address 2				
City		State/Province		
Country i		Postal Code		
Phone Number		Fax Number		
Email Address				
Additional Assignee or Non-Applicant Assignee Data may be generated within this form by selecting the Add button.				

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it contains a valid OMB control number.

Application Data Sheet 37 CFR 1.76		Attorney Docket Number	
		Application Number	
Title of Invention	DATA PROTECTION METHOD AND DEVICE		

Signature:

NOTE: This Application Data Sheet must be signed in accordance with 37 CFR 1.33(b). **However, if this Application Data Sheet is submitted with the INITIAL filing of the application and either box A or B is not checked in subsection 2 of the "Authorization or Opt-Out of Authorization to Permit Access" section, then this form must also be signed in accordance with 37 CFR 1.14(c).**

This Application Data Sheet **must** be signed by a patent practitioner if one or more of the applicants is a **juristic entity** (e.g., corporation or association). If the applicant is two or more joint inventors, this form must be signed by a patent practitioner, **all** joint inventors who are the applicant, or one or more joint inventor-applicants who have been given power of attorney (e.g., see USPTO Form PTO/AIA/81) on behalf of **all** joint inventor-applicants.

See 37 CFR 1.4(d) for the manner of making signatures and certifications.

Signature	/Joseph F. Edell/		Date (YYYY-MM-DD)		
First Name	Joseph	Last Name	Edell	Registration Number	67625
Additional Signature may be generated within this form by selecting the Add button.					

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent of: Scott A. Moskowitz
U.S. Patent No.: 9,104,842
Issue Date: August 11, 2015
Appl. No.: 11/895,388
Filing Date: August 24, 2007
Title: DATA PROTECTION METHOD AND DEVICE
Control No.: To be assigned

Mail Stop Ex Parte Reexam
ATTN: Central Reexamination Unit
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

**REQUEST FOR *EX PARTE* REEXAMINATION OF
U.S. PATENT NO. 9,104,842**

Dear Sir or Madam,

Pursuant to 35 U.S.C. § 302 and 37 C.F.R. § 1.510, *ex parte* reexamination is requested for claims 11, 12, 13, and 14 of United States Patent No. 9,104,842 (“the ’842 Patent,” Exhibit 1), issued on August 11, 2015. The ’842 Patent is currently assigned to Wistaria Trading Ltd. and remains in force.

TABLE OF CONTENTS

I.	INTRODUCTION	1
II.	CLAIMS FOR WHICH REEXAMINATION IS REQUESTED.....	1
III.	IDENTIFICATION OF PATENTS AND PRINTED PUBLICATIONS PRESENTED TO SHOW SUBSTANTIAL NEW QUESTIONS OF PATENTABILITY	2
IV.	CO-PENDING LITIGATION	2
V.	ESTOPPEL	3
VI.	OVERVIEW OF THE ORIGINAL PROSECUTION HISTORY	3
VII.	THE PRIORITY DATE OF THE '842 PATENT	10
VIII.	CLAIM CONSTRUCTION.....	10
IX.	THE PRIOR ART PROVIDES NEW, NON-CUMULATIVE TECHNICAL TEACHINGS.....	12
X.	DETAILED EXPLANATION UNDER 37 C.F.R. 1.510(b)(2).....	19
A.	SNQ-1: Claims 11, 12, 13, and 14 are Anticipated by Beetcher Under 35 U.S.C. §§ 102(a), (e).....	19
	1. Beetcher Anticipates Independent Claim 11.....	19
	2. Beetcher Anticipates Independent Claim 12.....	28
	3. Beetcher Anticipates Independent Claim 13.....	40
	4. Beetcher Anticipates Independent Claim 14.....	46
B.	SNQ-2: Claims 11, 12, 13, and 14 are Anticipated by Beetcher '072 Under 35 U.S.C. §§ 102(a), (b).....	50
	1. Beetcher '072 Anticipates Independent Claim 11.	50
	2. Beetcher '072 Anticipates Independent Claim 12.	59
	3. Beetcher '072 Anticipates Independent Claim 13.	71
	4. Beetcher '072 Anticipates Independent Claim 14.	76

C.	SNQ-3: Claims 11, 12, 13, and 14 are Anticipated by Cooperman Under 35 U.S.C. § 102(a).....	81
	1. Cooperman Anticipates Independent Claim 11.....	82
	2. Cooperman Anticipates Independent Claim 12.....	86
	3. Cooperman Anticipates Independent Claim 13.....	92
	4. Cooperman Anticipates Independent Claim 14.....	96
D.	SNQ-4: Claims 11, 12, 13, and 14 are Anticipated by Hasebe Under 35 U.S.C. §§ 102(a), (e).....	102
	1. Hasebe Anticipates Independent Claim 11.....	103
	2. Hasebe Anticipates Independent Claim 12.....	109
	3. Hasebe Anticipates Independent Claim 13.....	116
	4. Hasebe Anticipates Independent Claim 14.....	121
XI.	CONCLUSION.....	125

TABLE OF EXHIBITS

Exhibit No.	Description
Exhibit 1	U.S. Patent No. 9,104,842 to Moskowitz (“the ’842 Patent”)
Exhibit 2	Prosecution History of the ’842 Patent
Exhibit 3	U.S. Patent No. 5,933,497 (“Beetcher”)
Exhibit 4	Japanese Patent Application Publication No. H05334072 (“Beetcher ’072”)
Exhibit 5	English Translation of Beetcher ’072
Exhibit 6	PCT Application Publication No. WO 97/26732 (“Cooperman”)
Exhibit 7	U.S. Patent No. 5,935,243 (“Hasebe”)
Exhibit 8	[INTENTIONALLY LEFT BLANK]
Exhibit 9	Declaration of Dr. Claudio Silva (“Silva Declaration”)
Exhibit 10	Curriculum Vitae of Dr. Silva
Exhibit 11	Plaintiff Blue Spike LLC’s Proposed Terms for Construction, Pursuant to Patent Rule (P.R.) 4-2 in <i>Blue Spike, LLC v. Juniper Networks, Inc.</i> , Case No. 6:17-cv-16-KNM (E.D. Tex.)

I. INTRODUCTION

The '842 Patent claims methods of adding a license key to computer software. As the patent explains, the function of the key is to discourage consumers from making unauthorized copies of the software. During its original prosecution, the '842 Patent was subject to four rejections and an appeal resulting in an affirmation-in-part of the Examiner's rejections. The Examiner only allowed claims 11-14 to issue after the Board found the Holmes and Houser references did not teach or suggest the claimed license key. The Board found that the prior art did not include three elements: (1) software underlying functionality relating to code resource interrelationships, (2) a license key enabling software functionality, and (3) decoding an encoded code resource.¹ When rendering this conclusion, however, the Examiner and the Board were not aware of the prior art references that indeed disclose these three elements, as well as the remaining elements of claims 11-14. These prior art references—Beetcher, Beetcher '072, Cooperman, and Hasebe—establish that each of independent claims 11-14 are invalid as anticipated. In light of the substantial new questions of patentability that these references raise, as explained in further detail below, Requester respectfully seeks *ex parte* reexamination.

II. CLAIMS FOR WHICH REEXAMINATION IS REQUESTED

In accordance with 35 U.S.C. § 302 and 37 C.F.R. § 1.510, Requester seeks reexamination of claims 11, 12, 13, and 14 of the '842 Patent in view of the prior art patents and publications discussed herein.

¹ Ex. 2, Prosecution History at 1944-47 (*Patent Board Decision* (filed Mar. 12, 2015)); *id.* at 797-801 (*Notice of Allowability* (filed May 31, 2015)).

**III. IDENTIFICATION OF PATENTS AND PRINTED PUBLICATIONS
PRESENTED TO SHOW SUBSTANTIAL NEW QUESTIONS OF
PATENTABILITY**

The following four prior art patents and printed publications establish substantial new questions of patentability of claims 11, 12, 13, and 14 of the '842 Patent:

1. U.S. Patent No. 5,933,497 ("Beetcher" (Ex. 3));
2. Japanese Patent Application Publication No. H05334072 ("Beetcher '072" (Ex. 4));
3. PCT Application Publication No. WO 97/26732 ("Cooperman" (Ex. 6)).
4. U.S. Patent No. 5,935,243 ("Hasebe" (Ex. 7)).

Beetcher, Beetcher '072, and Hasebe were not cited in the '842 Patent itself, nor were they identified as being considered by the Examiner during prosecution. The '842 Patent lists Cooperman in its References Cited section,² but Cooperman was not subject to any rejection or prior art discussion during the original prosecution. And as detailed in Section IX., this request presents Cooperman in a new light and a different way that escaped review during earlier examination.

IV. CO-PENDING LITIGATION

Requester is currently engaged in pending litigation concerning the '842 Patent in *Blue Spike, LLC v. Juniper Networks, Inc.*, Case No. 6:17-cv-16-KNM (E.D. Tex.).

U.S. Patent No. 9,021,602 claims to be a continuation of the application that issued as the '842 Patent. Requester has filed an *ex parte* reexamination request for the '602 Patent in Control No. 90/014137.

Requester is unaware of any pending prosecution concerning the '842 Patent.

² '842 Patent at page 2.

V. ESTOPPEL

The statutory estoppel provisions of 35 U.S.C. § 315(e)(1) and 35 U.S.C. § 325(e)(1) do not prohibit Requester from filing this *ex parte* reexamination request.

VI. OVERVIEW OF THE ORIGINAL PROSECUTION HISTORY

The '842 Patent's claims 11-14 recite methods for adding license information to computer software and using that information to decode the software.³ The '842 Patent was subject to four rejections, an appeal resulting in an affirmation-in-part of the Examiner's rejection, and an amendment after allowance. This extended prosecution raises multiple issues relating to patentability.

Preliminary Amendments and Restriction Requirement

The application for the '842 Patent was filed on August 24, 2007 with 31 claims.⁴ With this initial filing, Patent Owner added 30 paragraphs to the specification.⁵ Patent Owner asserted that these new paragraphs were disclosed in U.S. Patent No. 5,745,569 ("the '569 Patent").⁶ Patent Owner stated that the parent application, for which the instant application claims to be a continuation, incorporated by reference the application that issues as the '569 Patent.⁷ Based on

³ *Id.* at claims 11-14.

⁴ Ex. 2, Prosecution History at 26-28 (*Claims* (filed Aug. 24, 2007)).

⁵ *Id.* at 30-28 (*Specification* (filed Aug. 24, 2007)).

⁶ *Id.* at 82 (*Applicant Arguments/Remarks* (filed Aug. 24, 2007)).

⁷ *Id.* Requester is not aware of any rule or precedent that permits a Patent Owner to amend an application to include substantially all of an issued patent into that application's specification based on a prior incorporation-by-reference in entirety statement. On the contrary, a general incorporation of a patent in its entirety is insufficient. *See, e.g., Callaway Golf Co. v. Acushnet Co.*, 576 F.3d 1331, 1346 (Fed. Cir. 2009) ("To incorporate matter by reference, a host document must contain language 'clearly identifying the subject matter which is incorporated and where it is to be found'; a 'mere reference to another application, or patent, or publication is not an incorporation of anything therein...'") (quoting *In re De Seversky*, 474 F.2d 671, 674 (C.C.P.A. 1973)).

this incorporation-by-reference, Patent Owner asserted that it was permissible to add substantially all of the '569 Patent's disclosure into the specification of the application for the '842 Patent.

Along with the preliminary amendment to the specification, Patent Owner cancelled certain claims from the parent application and added new claims.⁸ Shortly thereafter, Patent Owner requested another preliminary claim amendment.⁹ Patent Owner then requested yet another preliminary amendment to further amend claims and to include new claims.¹⁰ And nearly two years later, Patent Owner requested yet another preliminary amendment.¹¹

Based on a restriction requirement, Patent Owner elected to prosecute claims 32-45 and 52-59.¹²

Non-Final and Final Rejections

The Examiner's first office action rejected all claims on several grounds.¹³ Specifically, the Examiner provisionally rejected all claims, 32-45 and 52-59 on the ground of non-statutory obviousness-type double patenting as being unpatentable over claims 1-20 of co-pending Application No. 08/587943. The Examiner further rejected claims 32-39 under 35 U.S.C. § 101 as not falling within one of the four statutory categories of invention. The Examiner found that the "process" claimed in the application was "neither positively tied to a particular machine that accomplishes the claimed method steps nor transform underlying subject matter."¹⁴ The

⁸ Ex. 2, Prosecution History at 63-82 (*Preliminary Amendment* (filed Aug. 24, 2007)).

⁹ *Id.* at 120-24 (*Preliminary Amendment* (filed Oct. 19, 2007)).

¹⁰ *Id.* at 175-80 (*Preliminary Amendment* (filed Sept. 8, 2009)).

¹¹ *Id.* at 208-09 (*Preliminary Amendment* (filed Oct. 14, 2009)).

¹² *Id.* at 218-20 (*Response to Election/Restriction* (filed Dec. 10, 2009)).

¹³ *Id.* at 222-31 (*Non-Final Rejection* (filed Apr. 5, 2010)).

¹⁴ *Id.* at 225 (*Non-Final Rejection* (filed Apr. 5, 2010)).

Examiner further rejected claims 32 and 52 under 35 U.S.C. § 112 as failing to comply with the written description requirement and indefiniteness respectively.¹⁵ Finally, the Examiner held that all claims 32-45 and 52-59 were anticipated by Moore (U.S. Patent No. 6,067,622).¹⁶

In response to these rejections, Patent Owner amended the claims and provided arguments.¹⁷ Patent Owner argued that the provisional rejection on the ground of non-statutory obviousness-type double patenting may be incorrect given that Application No. 08/587,943 had issued.¹⁸ Patent Owner amended the claims to include generic computer components to overcome the Examiner's § 101 rejection.¹⁹ Patent Owner also amended claims 32 and 52 to remove recitation of elements lacking written description support and definiteness.²⁰ Patent Owner also argued that disclosures in Moore pertained to code modules and a copyright module, and not to watermarking.²¹ Patent Owner then asserted that Moore "does not disclose encoding a license key in software, using license information to identify a watermark in software, or decoding software using license information."²²

Subsequently, the Examiner issued a final rejection of all claims.²³ In his final rejection, the Examiner reiterated his rejection of all claims 32-45 and 52-64 on the ground of non-statutory obviousness-type double patenting as being unpatentable over claims 1-20 of co-

¹⁵ *Id.* at 226 (*Non-Final Rejection* (filed Apr. 5, 2010)).

¹⁶ *Id.* at 226-29 (*Non-Final Rejection* explaining how each claim was anticipated by Moore (filed Apr. 5, 2010)).

¹⁷ *Id.* at 292-302 (*Patent Owner Arguments/Remarks Made in Amendment* (filed Sept. 3, 2010)).

¹⁸ *Id.* at 292.

¹⁹ *Id.*

²⁰ *Id.*

²¹ *Id.*

²² *Id.*

²³ *Id.* at 389-489 (*Final Rejection* (filed Nov. 26, 2010)).

pending Application No. 08/587,943.²⁴ The Examiner also rejected claims 32 and 41 due to informalities under § 112.²⁵ The Examiner found that claim 32 was inconsistent with the specification's disclosure and that claim 41 was too broad.²⁶ The Examiner additionally rejected claims 62-64 under § 112 as failing to comply with the written description requirement. The Examiner found that the specification provided no support for the element "software code interrelationships" in claim 62, as it fails to teach or mention that term. The Examiner similarly found that the specification provided no support for the elements "encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a second license key encoded software code; wherein said first license key encoded software code is not identical to said second license key encoded software code," "second license key," and "second license key encoded software" in claims 63 and 64.²⁷ Finally, the Examiner rejected claims 32-45 and 52-61 as anticipated by Houser et al (U.S. Patent No. 5,606,609).²⁸

In response to the final rejection, Patent Owner amended the claims and responded to the final rejection.²⁹ Patent Owner rescinded its assertion that the application was a continuation-in-part to U.S. Patent Application No. 08/587,943, filed January 17, 1996.³⁰ And Patent Owner argued that the rejection on the ground of non-statutory obviousness-type double patenting was incorrect because U.S. Patent 5,745,569 "do[es] not define 'a license watermarked into the

²⁴ *Id.* at 391-92.

²⁵ *Id.* at 392-93.

²⁶ *Id.*

²⁷ *Id.* at 393-94.

²⁸ *Id.* at 394-97.

²⁹ *Id.* at 414-28 (*Patent Owner Arguments/Remarks Made in an Amendment* (filed Feb. 28, 2011)).

³⁰ *Id.* at 419.

software.”³¹ Further, in response to the rejections of claim 32, Patent Owner removed elements not supported by the specification.³² Patent Owner also amended claim 41 to include the additional element of “[information] defining an executable code providing a functionality of said software.”³³ Patent Owner argued that claims 62-64 had sufficient written description support, citing to various passages in the proposed specification.³⁴ Patent Owner further argued that Houser does not anticipate claims 32-45 and 52-61 because it does not disclose the claimed embedding of a watermark into software nor encoded software “designed to decide a first license code encoded in said software.”³⁵

Upon consideration of Patent Owner’s after-final rejection responses, the Examiner issued a non-final rejection of all claims.³⁶ The Examiner rejected claims 32-45 and 52-64 as obvious in view of Houser and Holmes (U.S. Patent No. 5,287,407).³⁷ The Examiner maintained his §112 rejection of claims 62-64, finding that the application fails to explicitly disclose or define “software code interrelationships,” “encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a second license key encoded software code; wherein said first license key encoded software code is not identical to said second license key encoded software code,” “second license key,” and “second license key

³¹ *Id.* at 415 (emphasis in original).

³² *Id.*

³³ *Id.*

³⁴ *Id.* at 415-17.

³⁵ *Id.* at 417-19.

³⁶ *Id.* at 436-44 (*Non-Final Rejection* (filed April 1, 2011)).

³⁷ *Id.* at 437, 440-43.

encoded software.”³⁸ The Examiner further determined that claim 37 provides insufficient antecedent basis for the limitation “determine said key.”³⁹

In response to this non-final rejection, Patent Owner amended the claims and submitted arguments.⁴⁰ Patent Owner argued that claims 62-64 were improperly rejected under §112 because one skilled in the art would understand the elements at issue.⁴¹ Patent Owner also argued that the obviousness rejection of claims 32-45 and 52-64 by Holmes in view of Houser was incorrect.⁴² Patent Owner argued that Holmes was not relevant because Holmes disclosed changing of data rather than code and does not mention licenses or activation of software.⁴³ Patent Owner further argued that Houser did not disclose modifying the underlying functionality of the software as set forth in claim 32’s preamble, but instead teaches modifying a file containing software by changing nonfunctional identifying data contained in the file.⁴⁴ Patent Owner further argued that there was no motivation to combine Holmes in view of Houser.⁴⁵

On September 20, 2011, the Examiner issued a final rejection.⁴⁶ The Examiner restated its obviousness rejection of claims 32-45 and 52-64 based on Holmes and Houser⁴⁷ and withdrew its § 112 rejection of claims 62-64.⁴⁸

³⁸ *Id.* at 437-40.

³⁹ *Id.* at 440.

⁴⁰ *Id.* at 516-31 (*Patent Owner Arguments/Remarks Made in an Amendment* (filed Feb. 28, 2011)).

⁴¹ *Id.* at 517-21.

⁴² *Id.* at 521-23.

⁴³ *Id.* at 521-22.

⁴⁴ *Id.* at 523.

⁴⁵ *Id.*

⁴⁶ *Id.* at 537-44 (*Final Rejection* (filed Sept. 20, 2011)).

⁴⁷ *Id.* at 537-42.

⁴⁸ *Id.* at 538, 542-43.

Appeal

In response to this final rejection, Patent Owner submitted a notice of appeal⁴⁹ and later filed an appeal brief.⁵⁰ In response, the Examiner filed the Examiner's answer to the appeal brief, which, in part, withdrew the § 103 rejection of claim 58.⁵¹ And Patent Owner filed its reply on August 13, 2012.⁵²

Thirty-one months after submission of the appeal reply, the Board issued its decision affirming-in-part the Examiner's final rejection.⁵³ In its decision, the Board affirmed the Examiner's obviousness rejection of claims 32, 33, 35, 37, 39, 52, 53, 55-57, 59, and 63-64 and reversed the Examiner's obviousness rejection of claims 36, 38, 40-44, and 60-62. The Board found, in pertinent part, that neither Holmes nor Houser teaches or suggests enabling software functionality based on a license key, and thus did not sustain the rejection of claims 36 and 60.⁵⁴ With respect to claim 61, the Board found that neither Holmes or Houser teaches or suggests "a modified software code comprising an encoded first code resource and a decode resource for decoding the encoded first code resource, wherein the decode resource is configured to decode the encoded first code resource upon receipt of a first license key."⁵⁵ The Board further found that the Examiner failed to show how Houser or Holmes teaches or suggests all the limitations of

⁴⁹ *Id.* at 564 (*Notice of Appeal* (filed Mar. 12, 2012)).

⁵⁰ *Id.* at 569-682 (*Appeal Brief* (filed May 14, 2012)).

⁵¹ *Id.* at 687-91 (*Examiner's Answer to Appeal Brief* (filed Aug. 8, 2012)) (original claim 58 corresponds to issued claim 11).

⁵² *Id.* at 692-700 (*Reply Brief* (filed Aug. 13, 2012)).

⁵³ *Id.* at 705-16 (*Patent Board Decision* (filed Mar. 12, 2015)).

⁵⁴ *Id.* (original claim 60 corresponds to issued claim 12).

⁵⁵ *Id.* at 715 (original claim 61 corresponds to issued claim 130).

claim 62, such as “software code interrelationships between code resources that result in a specified underlying functionality.”⁵⁶

On June 4, 2015, the Examiner issued a notice of allowance based on the Board’s March 12, 2015 decision.⁵⁷ After the notice of allowance, Patent Owner requested claim amendments, adding, in pertinent part, the term “product” to claim 58.⁵⁸ The patent issued on August 11, 2015.

VII. THE PRIORITY DATE OF THE ’842 PATENT

The ’842 Patent lists on its face that it is a continuation of the application that issued as U.S. Patent No. 7,664,263 (“the ’263 Patent”), which was filed on June 25, 2003.⁵⁹ And the ’842 Patent lists on its face that the ’263 Patent is a continuation of the application that issued as 6,598,162 (“the ’162 Patent”), which was filed on March 24, 1998.⁶⁰

Requester does not concede that the ’842 Patent is entitled to claim priority to the filing date of either the ’263 Patent or the ’162 Patent but assumes, for purposes of this proceeding only, that the earliest possible priority date for the ’842 Patent is March 24, 1998.

VIII. CLAIM CONSTRUCTION

During reexamination of an unexpired patent, claims are given their “broadest reasonable interpretation” consistent with the specification.⁶¹ This standard, however, differs from the claim construction standard used in district court litigation.⁶² Accordingly, the discussion below is

⁵⁶ *Id.* at 715-16 (original claim 62 corresponds to issued claim 14).

⁵⁷ *Id.* at 793-801 (*Notice of Allowance* (filed June 4, 2015)).

⁵⁸ *Id.* at 874-81.

⁵⁹ ’842 Patent at [Related U.S. Application Data].

⁶⁰ *Id.*

⁶¹ MPEP 2258(G) (citing *In re Yamamoto*, 740 F.2d 1569 (Fed. Cir. 1984)).

⁶² *Phillips v. AWH Corp.*, 415 F.3d 1303, 1316 (Fed. Cir. 2005) (words of a claim “are generally given their ordinary and customary meaning” as understood by a person of ordinary skill in the art (“POSITA”) at the time of the invention).

directed to the broadest reasonable interpretation of the claims and is without prejudice to any claim interpretation that Requester may urge in litigation involving the '842 Patent.

“encoding algorithm” (claims 12-14): Requester proposes that the term “encoding algorithm” means “a process or set of instructions for encoding data.” This construction is the broadest reasonable interpretation consistent with the specification. Indeed, the specification refers to various processes related to encoding data for the generation of a license key. For example, the specification states that “any authenticating function can be combined, such as Digital Signature Standard (DSS) or Secure Hash Algorithm (SHA)” to generate an encoded key.⁶³ The patent also provides other example algorithms, including “[a] block cipher, such as a Data Encryption Standard (DES) algorithm, in combination with a sufficiently random seed value, such as one created using a Message Digest 5 (MD5) algorithm” to emulate a cryptographically secure random bit generator.⁶⁴ A POSITA would have recognized these examples as processes or sets of instructions for encoding data.⁶⁵

“code resource” (claims 12-14): This term is unclear, and the intrinsic evidence fails to provide any boundaries for it, thus rendering it indefinite. But, because an *ex parte* reexamination request may not challenge a claim based on indefiniteness,⁶⁶ Requester uses Patent Owner’s construction for this term proposed in the litigation, namely, that this term is subject to its plain and ordinary meaning.⁶⁷ The '842 Patent refers to sub-objects and a memory scheduler as examples of code resources.⁶⁸

⁶³ '842 Patent at 8:5-9, 21-23.

⁶⁴ *Id.* at 8:12-16.

⁶⁵ Silva Declaration at ¶ 22.

⁶⁶ MPEP 2258.

⁶⁷ Ex. 10, Blue Spike Proposed Constructions at 57-58.

⁶⁸ '842 Patent at 11:55-65, 15:36-42.

“software code interrelationships” (claims 14): This term is unclear, and the intrinsic evidence fails to provide any boundaries for it, thus rendering it indefinite. But, because an *ex parte* reexamination request may not challenge a claim based on indefiniteness,⁶⁹ Requester uses Patent Owner’s construction for this term proposed in the litigation, namely, that this term is subject to its plain and ordinary meaning.⁷⁰ Notably, during the original prosecution, Patent Owner stated “interrelationship” is defined as “the way in which two or more things affect each other because they are related in some way.”⁷¹

IX. THE PRIOR ART PROVIDES NEW, NON-CUMULATIVE TECHNICAL TEACHINGS.

The Patent Office did not consider Beetcher, Beetcher ’072, and Hasebe individually or in combination during the original prosecution of the ’842 Patent. And the Patent Office did not consider Cooperman in the new light presented herein. As such, these four references provide new, non-cumulative teachings that warrant a reexamination of the ’842 Patent.

Beetcher was issued on August 3, 1999 based on a U.S. application filed January 29, 1993, which in turn was a continuation application to a U.S. application filed December 14, 1990.⁷² Beetcher is a patent granted on a U.S. application by another before the earliest possible priority date for the ’842 Patent and is thus prior art under at least pre-AIA 35 U.S.C. § 102(a) and § 102(e). As explained in more detail below, Beetcher discloses an apparatus and method of key-protected software distributed separately from an encrypted entitlement key that enables execution of the software.⁷³ Beetcher further discloses (a) enabling software functionality based

⁶⁹ MPEP 2258.

⁷⁰ Ex. 10, Blue Spike Proposed Constructions at 58-59.

⁷¹ Ex. 2, Prosecution History at 518.

⁷² Beetcher at Date of Patent [45], Filed [22], Related U.S. Application Data [63].

⁷³ *Id.* at Abstract, 4:3-46.

on a license key, (b) decoding an encoded a code resource upon receipt of a license key, and (c) interrelationships between code resources that result in a specified underlying functionality, which the Board found was missing from the prior art of record during the original prosecution.⁷⁴ Beetcher's disclosures raise substantial questions as to the anticipation of claims 11-14 of the '842 Patent.

Beetcher '072 is a Japanese Patent Application Publication published on December 17, 1993.⁷⁵ Beetcher is a printed publication published more than one year prior to the earliest possible priority date for the '842 Patent and is thus prior art under at least pre-AIA 35 U.S.C. § 102(a) and § 102(b). Beetcher '072 claims priority to the U.S. application No. 07/629,295,⁷⁶ which is the parent application to the Beetcher reference discussed above. This Request refers to Beetcher '072's Japanese disclosures as well as to the corresponding translation of those Japanese disclosures, Ex. 5.⁷⁷ As explained in more detail below, Beetcher '072 discloses an apparatus and method of key-protected software distributed separately from an encrypted entitlement key that enables execution of the software.⁷⁸ Beetcher '072 further discloses (a) enabling software functionality based on a license key, (b) decoding an encoded a code resource upon receipt of a license key, and (c) interrelationships between code resources that result in a

⁷⁴ Ex. 2, Prosecution History at 1944-47 (*Patent Board Decision* (filed Mar. 12, 2015)).

⁷⁵ Beetcher '072 at Publication Date (43).

⁷⁶ *Id.* at Related Application Data (31), (32), (33).

⁷⁷ Ex. 5 is a machine translation of Beetcher '072 available at https://www19.j-platpat.inpit.go.jp/PA1/cgi-bin/PA1DETAIL?MaxCount=1000&PageCount=1000&SearchType=0&TempName=w--adaa&MaxPage=1&DispPage=1+1000&HitCount=31&ResultId=I00333004701&CookieId=2&DetailPage=9&Language=ENG&Reserve1=DetailPaging&Reserve2=j60EUdc54_KVb6a06Ieg&Reserve3=/ (last visited Apr. 18, 2018).

⁷⁸ *E.g.*, Beetcher '072 at Abstract.

specified underlying functionality, which the Board found was missing from the prior art of record during the original prosecution.⁷⁹

Beetcher '072's disclosures raise substantial questions as to the anticipation of claims 11-14 of the '842 Patent. These questions are non-cumulative of Beetcher because Beetcher '072 was published more than one year before the earliest potential priority date of the '842 Patent. Thus, it will not be possible for Patent Owner to attempt to ante-date Beetcher '072 by arguing the named inventor conceived and diligently reduced to practice the invention claimed in the '842 Patent prior to the publication date of Beetcher '072.

Hasebe was issued on August 10, 1999 based on a U.S. application filed July 1, 1993 and claims priority to a Japanese patent application filed August 31, 1995.⁸⁰ Hasebe is a patent granted on a U.S. application by another before the earliest possible priority date for the '842 Patent and is thus prior art under at least pre-AIA 35 U.S.C. § 102(a) and § 102(e). Hasebe discloses a license notification system for converting license-protected software to an executable form using license information, as explained in more detail below.⁸¹ Hasebe further discloses (a) enabling software functionality based on a license key, (b) decoding an encoded code resource upon receipt of a license key, and (c) interrelationships between code resources that result in a specified underlying functionality, which the Board found was missing from the prior art of record during the original prosecution.⁸² Thus, Hasebe's disclosures raise substantial questions as to the anticipation of claims 11-14 of the '842 Patent.

⁷⁹ Ex. 2, Prosecution History at 1944-47 (*Patent Board Decision* (filed Mar. 12, 2015)).

⁸⁰ Hasebe at Date of Patent [45], Filed [22], Related U.S. Application Data [30].

⁸¹ *Id.* at Abstract, 2:42-3:15.

⁸² Ex. 2, Prosecution History at 1944-47 (*Patent Board Decision* (filed Mar. 12, 2015)).

Cooperman was published on July 24, 1997⁸³ and is prior art under at least pre-AIA 35 U.S.C. § 102(a). Cooperman lists on its face inventors Marc Cooperman and Scott Moskowitz. As such, the Cooperman reference is a printed publication “by others,” as set forth in pre-AIA § 102(a). This is because the entities identified as the inventors of this reference differ from those of the ’842 Patent by at least one person, namely Mr. Cooperman.⁸⁴

While Patent Owner listed Cooperman among the 665 documents provided to the Examiner during the original prosecution,⁸⁵ Cooperman presents a substantial new question of patentability because this Request presents it in a new light. As set forth in MPEP 2216, a substantial new question of patentability exists when the pertinent publication raises:

[Q]uestions of patentability [that] are substantially different from those raised in the previous examination of the patent... The substantial new question of patentability may be based on art previously considered by the Office if the reference is presented in a new light or a different way that escaped review during earlier examination.⁸⁶

During the original prosecution of the ’842 Patent, none of the rejections or prior art discussions refer to Cooperman. The Board has routinely affirmed that a prior art reference cited on the face of a patent but neither relied upon to reject any claims during the prosecution nor discussed in the statement of reason for allowance of that patent should not preclude the existence of a

⁸³ Cooperman at 1.

⁸⁴ MPEP 2132, 2136.

⁸⁵ ’842 Patent at page 5.

⁸⁶ *See also* 35 U.S.C. § 303(a) (“The existence of a substantial new question of patentability is not precluded by the fact that a patent or printed publication was previously cited by or to the Office or considered by the Office.”); *In re Swanson*, 540 F.3d 1368, 1380 (Fed. Cir. 2008) (“The appropriate test to determine whether a ‘substantial new question of patentability’ exists should not merely look at the number of references or whether they were previously considered or cited but their combination in the appropriate context of a new light as it bears on the question of the validity of the patent” (quoting H.R. Rep. No. 107-120, at 3)).

substantial new question of patentability.⁸⁷ Here, Cooperman is presented in a new light because the question of whether Cooperman anticipates claims 11-14 was not addressed or resolved during the original prosecution, thus raising a substantial new question regarding patentability. Accordingly, SNQ-3 in Section X.C. presents a limitation-by-limitation discussion of Cooperman's teachings that is new and non-cumulative to the original prosecution's record.

Large portions of Cooperman's disclosure are identical to portions of the '842 specification.⁸⁸ During the original prosecution, Patent Owner admitted that these portions common to the '842 Patent and Cooperman teach limitations recited in independent claims 11-14.⁸⁹

More specifically, Cooperman discloses a method that ensures licensing information is preserved in copies of an original works, including application software, as explained in more detail below.⁹⁰ Cooperman further discloses (a) enabling software functionality based on a license key, (b) decoding an encoded code resource upon receipt of a license key, and (c) interrelationships between code resources that result in a specified underlying functionality, which the Board found was missing from the prior art of record during the original prosecution.⁹¹

⁸⁷ See, e.g., *Ex parte Civix DDI LLC*, 2011 WL 4007697, at *12 (B.P.A.I. Sept. 7, 2011) (“[T]he record reveals that Examiner did engage in a fact-specific inquiry and correctly determined that the “old art” of Tornetta raises an SNQ. Among other things, the Examiner stated that ‘a review of the prosecution history of application 08/920,044 Reveals that ... ‘Tornetta’ even though considered by the Examiner [was] not relied upon to reject any claims during the prosecution of the '307 patent, nor was it discussed by the examiner of record in the statement of reason for allowance of that patent.’”); *Ex parte Allied Mach. & Eng'g Corp.*, 2015 WL 5719730, at *6 (P.T.A.B. Sept. 25, 2015) (similar).

⁸⁸ E.g., compare Cooperman at 11:9-12:2 with '842 Patent at 13:44-14:6.

⁸⁹ E.g., Ex. 2, Prosecution History at 577-81 (original claim 58, 59, 61, and 62 issued as claim 11, 12, 13, and 14, respectively).

⁹⁰ Cooperman at Abstract, 5:25-6:9.

⁹¹ Ex. 2, Prosecution History at 1944-47 (*Patent Board Decision* (filed Mar. 12, 2015)).

Moreover, as discussed with respect to the prosecution history overview in Section VI, Patent Owner initially claimed priority to Application No. 08/587,943 filed January 17, 1996. Later, Patent Owner rescinded its priority claim to Application No. 08/587,943,⁹² and relied on Application No. 09/046,627 to establish the earliest possible priority of March 24, 1998.⁹³ Yet the prosecution history indicates that the Examiner limited his search to prior art dated after January 17, 1996 (filing date of Application No. 08/587,943), even after Patent Owner rescinded its claim to that priority date.⁹⁴ As annotated and shown below, the Examiner's search histories show limiting consideration of prior art dated after January 17, 1996 (dashed boxes):

Search History					
Search History (Prior Art)					
Hits	Search Query	Dbs	Default Operator	Plurals	Time Stamp
3060	[watermark near3 (software or program or application)]	US-PG-PUB; USPAT; USOCR; FPPS; EPO; JPO; DEPVENT; IBM_TDS	AND	ON	2011/03/16 13:21
193	[watermark near3 (software or program or application)] and (@ad~"19960117" or @rad~"19960117")	US-PG-PUB; USPAT; USOCR; FPPS; EPO; JPO; DEPVENT; IBM_TDS	AND	ON	2011/03/16 13:23
21	[watermark near3 (software or program or application)] and (@ad~"19960117")	US-PG-PUB; USPAT; USOCR; FPPS; EPO; JPO; DEPVENT; IBM_TDS	AND	ON	2011/03/16 22:01
94	[watermark same (software or program or application)] and (@ad~"19960117")	US-PG-PUB; USPAT; USOCR; FPPS; EPO; JPO; DEPVENT; IBM_TDS	AND	ON	2011/03/16 22:26

⁹² *Id.* at 1650 (*Patent Owner Arguments/Remarks Made in an Amendment* (filed Feb. 28, 2011)).

⁹³ '842 Patent at [Related U.S. Application Data].

⁹⁴ Ex. 2, Prosecution History at 448-57, 547-57, 863-69 (*Examiner Search Strategies and Results* (filed Apr. 1, 2011, Sept. 20, 2011, June 4, 2015)).

Search History					
Search History (Prior Art)					
Hits	Search Query	DBs	Default Operator	Plurals	Time Stamp
18223	713,183,176,187,187,184,380/201,228,229.cdc.	US-PQPLUS, USPAT, USDCPL, PPRS, EPC, JPO, DEPAMENT, ISM, TDS	AND	CN	2015/05/31 22:11
39333	F041,03/0425;F041,2203/008;F041,2203/00.cpc.	US-PQPLUS, USPAT, USDCPL, PPRS, EPC, JPO, DEPAMENT, ISM, TDS	AND	CN	2015/05/31 22:12
113	(watermark same ((software or program or application))) and (@act="19980117")	US-PQPLUS, USPAT, USDCPL, PPRS, EPC, JPO, DEPAMENT, ISM, TDS	AND	CN	2015/05/31 22:13
2	S73 and S74	US-PQPLUS, USPAT, USDCPL, PPRS, EPC, JPO, DEPAMENT, ISM, TDS	AND	CN	2015/05/31 22:13
23	(watermark near3 ((software or program or application))) and (@act="19980117")	US-PQPLUS, USPAT, USDCPL, PPRS, EPC, JPO, DEPAMENT, ISM, TDS	AND	CN	2015/05/31 22:13

As such, the Examiner would not have considered Cooperman (which was respectively filed and published on January 16 and July 24, 1997) to be prior art as it was published after January 17, 1996. Because Cooperman is prior art under at least § 102(a), Requester has presented Cooperman in a new light not considered during the original prosecution.

As explained, the Examiner did not consider the Beetcher, Beetcher '072, and Hasebe references. And this Request presents Cooperman in a new light and in a different way that escaped earlier review. As such, no consideration has been given whether any of these references anticipates claims 11-14, including limitations toward underlying functionality relating to code resource interrelationships, a license key enabling software functionality, and decoding an encoded code resource that the Board found missing from the prior art during the original prosecution.

The substantial new questions of patentability under 37 C.F.R. § 1.510(b)(1) presented in this Request are listed below and based on the four prior art references Beetcher, Beetcher '072, Hasebe, and Cooperman that were not the subject of any final decision by the Patent Office or court:

No.	Substantial New Questions of Patentability of the '842 Patent
1	Claims 11, 12, 13, and 14 are anticipated by Beetcher under pre-AIA 35 U.S.C. §§ 102(a), (e).
2	Claims 11, 12, 13, and 14 are anticipated by Beetcher '072 under pre-AIA 35 U.S.C. §§ 102(a), (b).
3	Claims 11, 12, 13, and 14 are anticipated by Cooperman under pre-AIA 35 U.S.C. § 102(a).
4	Claims 11, 12, 13, and 14 are anticipated by Hasebe under pre-AIA 35 U.S.C. §§ 102(a), (e).

X. DETAILED EXPLANATION UNDER 37 C.F.R. 1.510(b)(2)

A. SNQ-1: Claims 11, 12, 13, and 14 are Anticipated by Beetcher Under 35 U.S.C. §§ 102(a), (e).

Beetcher anticipates claims 11, 12, 13, and 14 under 35 U.S.C. §§ 102(a), (e).

1. Beetcher Anticipates Independent Claim 11.

a) Preamble: “A method for licensed software use, the method comprising”

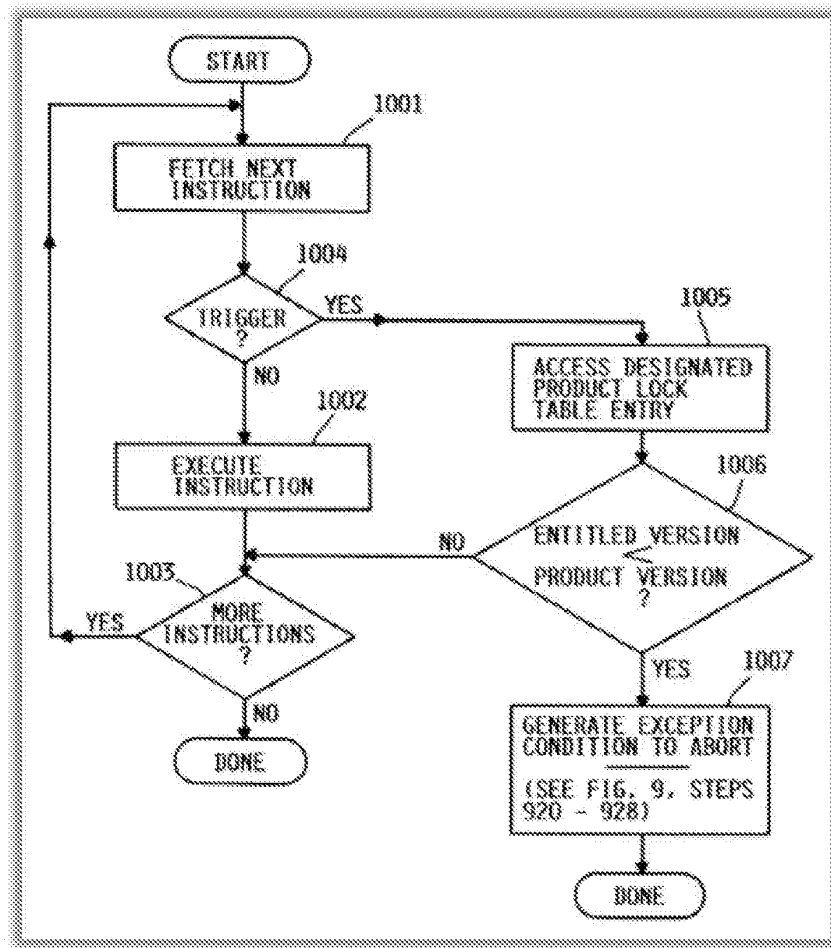
Under the broadest reasonable construction, the preamble is non-limiting. Nevertheless, Beetcher discloses claim 11’s preamble. Specifically, Beetcher describes a method of controlling access to licensed software using an encrypted entitlement key.⁹⁵ Beetcher, for instance, summarizes its invention as:

Software is distributed according to the present invention without entitlement to run. A separately distributed encrypted entitlement key enables execution of the Software. The key includes the serial number of the machine for which the Software

⁹⁵ Beetcher at Abstract, 4:3-13, 4:39-44, 10:48-11:3; *see also id.* at 1:7-11, 1:54-57, 3:54-62.

is licensed, together with a plurality of entitlement bits indicating which Software modules are entitled to run on the machine.⁹⁶

Beetcher's Figure 10, as provided below, illustrates the use of an entitled version of software based on the customer's license:



As such, Beetcher teaches this preamble⁹⁷.

⁹⁶ Beetcher at 4:3-9.

⁹⁷ Silva Declaration at ¶¶ 35-36.

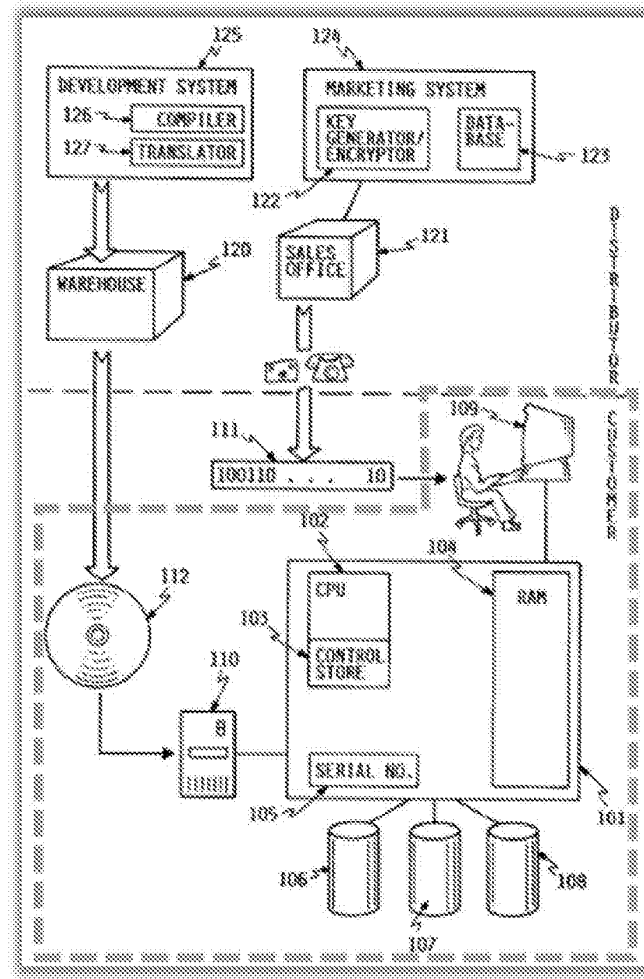
- b) ***Element 11.1: “loading a software product on a computer, said computer comprising a processor, memory, an input, and an output, so that said computer is programmed to execute said software product”***

Beetcher discloses element 11.1. Specifically, Beetcher’s system includes a customer computer 101 including a CPU 102, memory 104, and storage devices 106-108.⁹⁸ This customer computer 101 also includes a media reader 110 (i.e., an input) and an operator console 109 (i.e., an output).⁹⁹ As shown below in annotated Figure 1, Beetcher discloses a computer having software product 112 loaded for execution (dashed perimeter)¹⁰⁰:

⁹⁸ Beetcher at 5:14-21, Fig. 1.

⁹⁹ *Id.* at 5:25-32, 6:7-15, Fig. 1.

¹⁰⁰ Silva Declaration at ¶¶ 38-40.



Beecher details that the customer loads the media, such as an optical disk, containing a software product onto the computer to execute the software product:

[S]oftware media 112 comprise one or more optical read/only disks, and unit 110 is an optical disk reader, it being understood that electronic distribution or other distribution media could be used. Upon receipt of software media 112, the customer will typically load the desired software modules from unit 110 into system 101, and store the software modules on storage devices 106-108.¹⁰¹

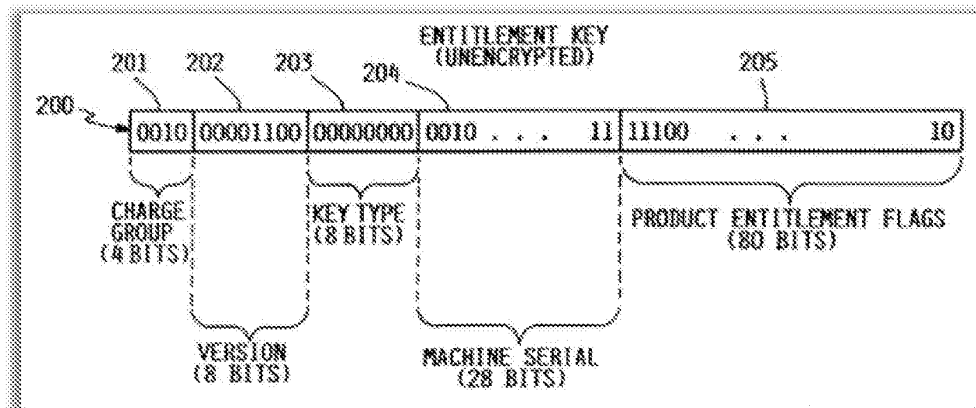
¹⁰¹ Beecher at 6:7-15; *see also id.* at Abstract, 3:48-50, 9:51-55, Fig. 1, claim 6.

c) **Element 11.2: “said software product outputting a prompt for input of license information”**

Beecher discloses element 11.2. Specifically, Beecher explains that its software product includes a user interface routine for the customer to input a license key into the computer before the product can be used.¹⁰² For instance, Beecher explains that the software product prompts the user to input license information:

This operation system support at **virtual machine level 404 contains two user interface routines needed to support input of the entitlement key**. General input routine 441 is used to handle input during normal operations. In addition, **special install input routine 440 is required to input the key during initial installation of the operating system**. This is required because that part of the operating system above machine interface level 405 is treated for purposes of this invention as any other program product; it will have a product number and its object code will be infected with entitlement verification triggers.¹⁰³

Beecher’s Figure 2 illustrates this license information in unencrypted form:



Beecher further explains that the software’s “install input routine 440 interacts with the operator to receive the input” of the customer’s license information during the software’s initial installation.¹⁰⁴ And as discussed with respect to element 11.1, the customer’s computer includes

¹⁰² *Id.* at 7:66-8:8; *see also id.* at 3:25-28.

¹⁰³ *Id.* at 7:66-8:8.

¹⁰⁴ *Id.* at 9:51-55; *see also id.* at Fig. 4 (reference number 440), claim 6.

an operator console 109 shown with a monitor and keyboard that “can receive input from an operator.”¹⁰⁵

- d) ***Element 11.3: “said software product using license information entered via said input in response to said prompt in a routine designed to decode a first license code encoded in said software product”***

Beetcher discloses element 11.3. Upon inserting the software’s disk 112, Beetcher explains that the operator console prompts the customer to enter a license key.¹⁰⁶ Beetcher details that the customer enters entitlement key 111, i.e., license information, in response to the prompt initiated by install input routine 440.¹⁰⁷ After entering that key, Beetcher teaches that the customer’s computer uses a decode key to initiate unlock routine 430 to decode the license code encoded in the software product.¹⁰⁸ Beetcher’s Figures 4 and 9a, which are provided below, show the software using the key (i.e., license information) entered by the customer to decode a first license code encoded in the software product. For instance, annotated Figure 4 illustrates that the install input routine 440 starts unlock routine 430 once the customer inputs key 111 into the computer.¹⁰⁹ And “[u]nlock routine 430 uses the unique machine key to decode[] entitlement key 111” (dashed perimeter):¹¹⁰

¹⁰⁵ *Id.* at 3:25-28, Fig. 1; Silva Declaration at ¶¶ 42-44.

¹⁰⁶ *E.g.*, Beetcher at 6:11-19, 7:66-8:8, Figs. 1, 9a.

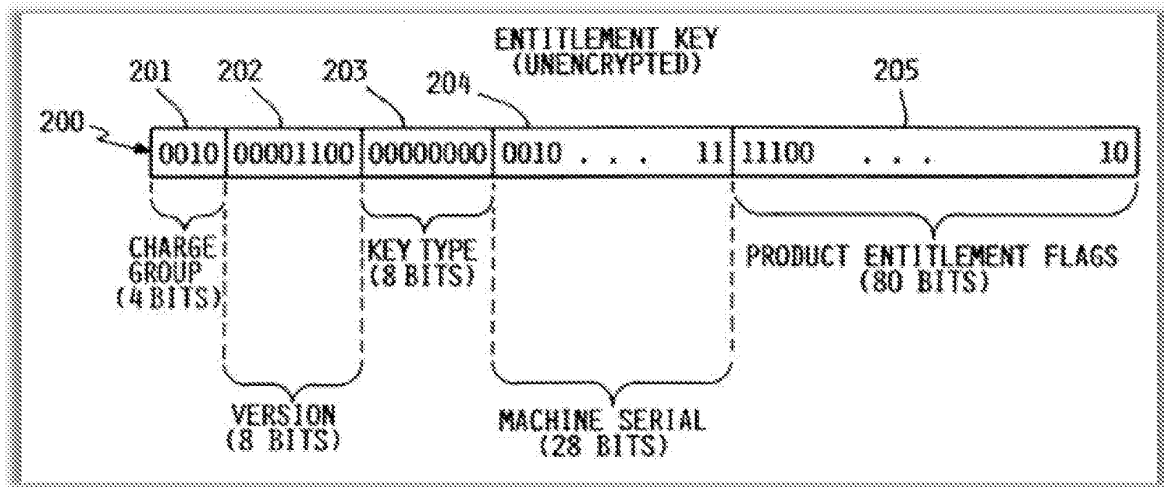
¹⁰⁷ *Id.* at 7:66-8:8; *see also id.* at 9:51-55, Figs. 1, 4, claim 6.

¹⁰⁸ *Id.* at 7:39-42, 9:49-60; *see also id.* at 6:66-7:5, 8:60-62 Figs. 4, 9a.

¹⁰⁹ *Id.* at 8:3-13, 9:52-60.

¹¹⁰ *Id.* at 7:39-42; *see also id.* at 8:62-62; 10:27-36.

specifying customer's accessible product numbers.¹¹² Beetcher's Figure 2 shows this license information with fields 201 to 205:



Beetcher's unlock routine 430 will complete the decoding process by building an encoded product key table (step 904), populating the key table for the relevant software product specified in the entitlement key (steps 905-908), and saving the key table (step 909).¹¹³ Beetcher also specifies that the customer's RAM includes table 460 populated with products having entitlement keys.¹¹⁴ Beetcher's software product uses the key's version and product number fields to decode a license code.

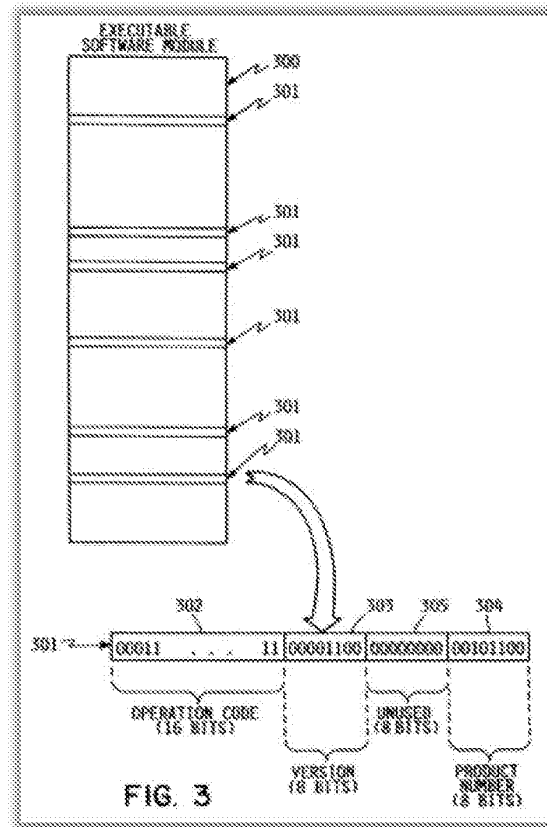
When compiling and translating the software code, Beetcher explains that the code includes entitlement verification triggering instructions encoded into the software.¹¹⁵ Beetcher's triggering instructions are encoded into the software when the software code is compiled and translated, as shown in Figure 3 provided below:

¹¹² *Id.* at 6:22-40.

¹¹³ *Id.* at 9:60-10:19, Figs. 5, 9a.

¹¹⁴ *Id.* at 7:42-44, 8:43-52, 10:20-47, Fig. 6, Fig. 9a.

¹¹⁵ *Id.* at 6:41-58, 11:4-39; *see also id.* at 4:14-23, 8:5-22, 8:56-9:20.



Beetcher explains that its software code verifies the customer is entitled to use the software when the code encounters a triggering instruction. When it encounters one of these instructions, Beetcher's code accesses the license key information stored in the key table 460.¹¹⁶ As such, a POSITA would have understood that Beetcher uses its license information in a routine, such as check lock function 422, designed to decode a first license code encoded in a software product via the triggering instructions:

If any instruction is an entitlement verification triggering instruction 301 (step 1004) check lock function 422 is invoked. Check lock function 422 accesses the product lock table entry 601 corresponding to the product number contained in the triggering instruction at step 1005. If the version number in product lock table 460

¹¹⁶ *Id.* at 10:48-11:39; *see also id.* at Abstract, 8:14-22, 8:53-9:20, Fig. 10.

is equal to or greater than the version number 303 contained in triggering instruction 301, the software is entitled to execute (step 1006).¹¹⁷

Moreover, Beetcher teaches that the triggering instructions will be encoded into the code resources controlling software functionality:

[An] additional barrier would be to define the entitlement triggering instruction to simultaneously perform some other function.... The alternative function must be so selected that any compiled software module will be reasonably certain of containing a number of instructions performing the function. If these criteria are met, the compiler can automatically generate the object code to perform the alternative function (and simultaneously, the entitlement verification trigger) as part of its normal compilation procedure. This definition would provide a significant barrier to patching of the object code to nullify the entitlement triggering instructions.¹¹⁸

And Beetcher details that “the triggering instruction is also a direct instruction to perform some other useful work [E]xecution of the triggering instruction causes system 101 to perform some other operation simultaneous with the entitlement verification.”¹¹⁹

Accordingly, Beetcher discloses claim 11.

2. Beetcher Anticipates Independent Claim 12.

a) **Preamble: “A method for encoding software code using a computer having a processor and memory, the method comprising”**

Under the broadest reasonable construction, the preamble is non-limiting.¹²⁰

Nevertheless, Beetcher discloses claim 12’s preamble. Specifically, Beetcher describes a method

¹¹⁷ *Id.* at 10:52-62, Fig. 10; Silva Declaration at ¶¶ 46-51.

¹¹⁸ Beetcher at 11:14-28; *see also id.* at 4:25-33, 6:58-65.

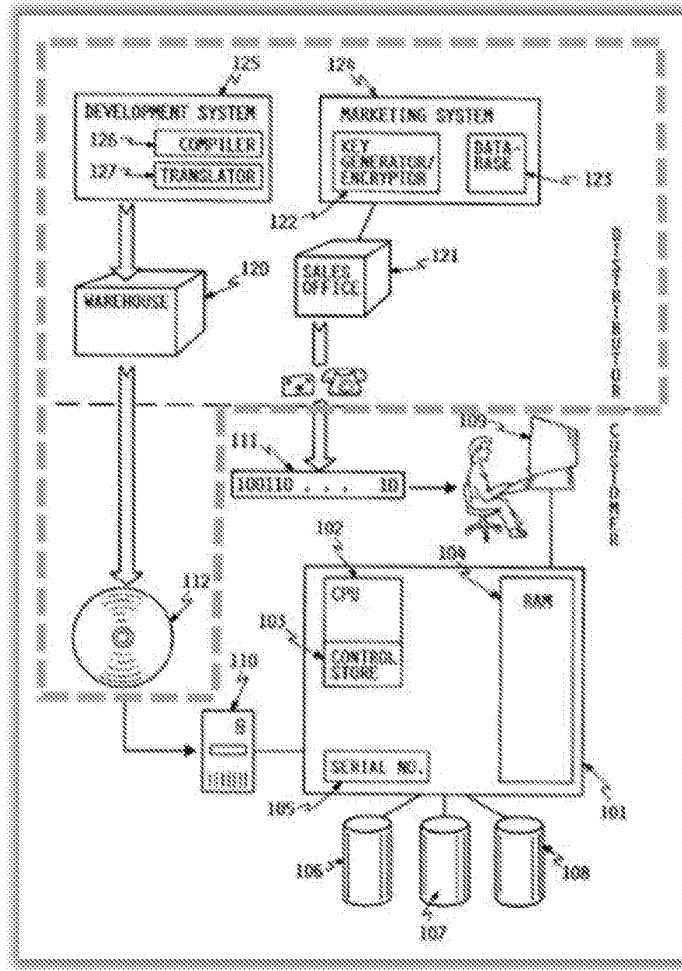
¹¹⁹ *Id.* at 6:58-65 (Beetcher specifies that these functions are those “which do not require that an operand for the action be specified in the instruction.”); Silva Declaration at ¶¶ 52-53.

¹²⁰ Claim 12’s preamble recites “a computer” and claim 12’s body recites “a computer system.” It is unclear whether those elements refer to the same or separate computing devices. For purposes of this Request and using the broadest reasonable interpretation consistent with the specification, it is assumed that the “computer” recited in the preamble is a device separate from the “computer system.”

for encoding software code using a computer with a processor and memory. Beetcher details that the software distributor has “development computer system 125, which contains compiler 126 and translator 127” where “[t]he software modules are recorded on software recording media 112” and “entitlement key generator/encrypter 122 and a database 123 containing customer information.”¹²¹ Beetcher specifies these compiling and key generating functions may be performed by a single computer.¹²² Below annotated Figure 1 illustrates the distributor’s computer system distributing memory media 112 and compiling encoded software code:

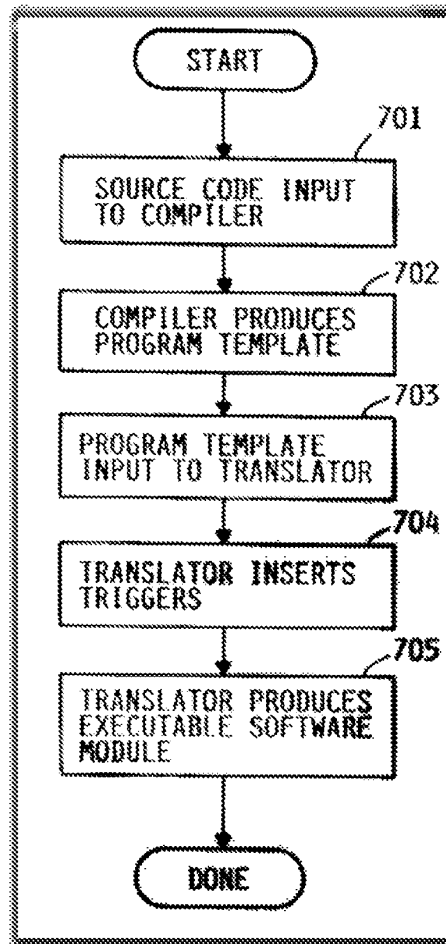
¹²¹ Beetcher at 5:38-48; *see also id.* at 9:1-20.

¹²² *Id.* at 5:51-58.



Beetcher's Figure 7 illustrates the software code being encoded to include watermarking triggers decoded by the customer's licensing information:¹²³

¹²³ *Id.* at 9:1-20, Fig. 7.



As such, a POSITA would have understood that Beetcher's distributor compiles and stores the encoded software code using a processor and memory akin to the console's CPU 102 and memory devices 106-108. As expert Dr. Silva explains in his declaration (Ex. 9), Beetcher's computer would necessarily include a processor and memory in order to function.¹²⁴

As such, Beetcher teaches this preamble.

¹²⁴ Silva Declaration at ¶¶ 56-59.

b) Element 12.1: “storing a software code in said memory”

Beetcher discloses element 12.1. Specifically, Beetcher discloses a development system 125 for compiling and translating for the software code.¹²⁵ Beetcher details that the software code is stored as disks 112 in warehouse 120. A POSITA would have understood that developer system 125 stores the compiled and translated code in memory and records that code onto disks 112 for distribution to customers. As expert Dr. Silva explains in his declaration (Ex. 9), Beetcher’s computer would necessarily include store software code in memory in order to function.¹²⁶

c) Element 12.2: “wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system”

Beetcher discloses element 12.2. Specifically, Beetcher explains that its software code includes multiple code resources that include a first code resource.¹²⁷ Beetcher’s code resources include software modules 300 (dashed box) including sub-objects within the code, as shown below in annotated Figure 4 and Figure 3.¹²⁸ These sub-objects control multiple functions of the software installed on the customer’s computer system 101.¹²⁹ And Beetcher’s software prevents unwanted “patching” of these sub-objects by including entitlement verification triggering instructions 301.¹³⁰

¹²⁵ Beetcher at 5:38-48, 9:1-20.

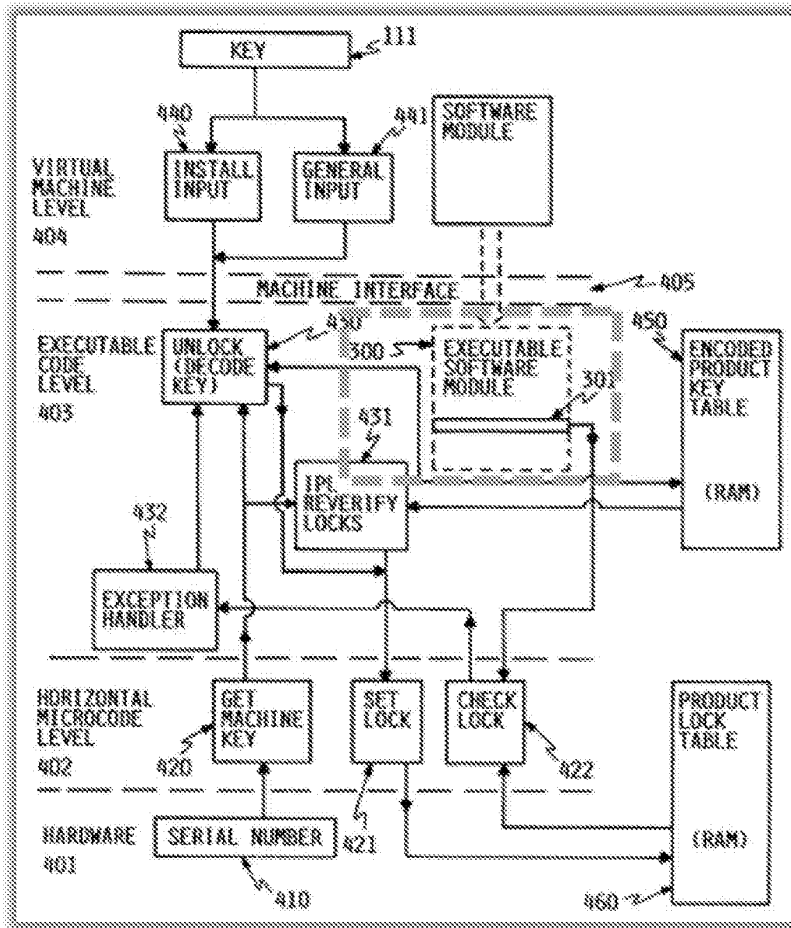
¹²⁶ Silva Declaration at ¶ 62.

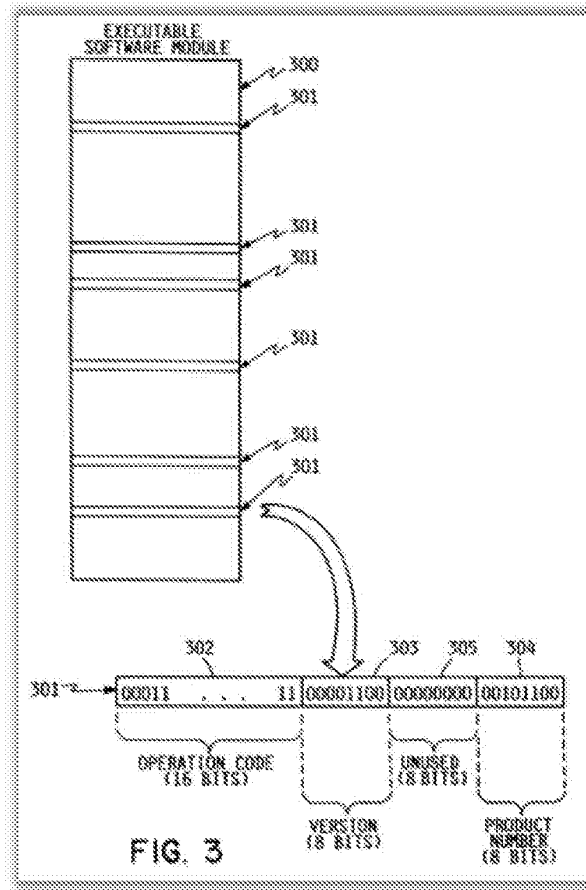
¹²⁷ Beetcher at 5:40-43, 6:1-15.

¹²⁸ *Id.* at 6:41-45, 8:14-17, Fig. 4; *see also id.* at 7:45-48, Fig. 3.

¹²⁹ *Id.* at 6:58-65, 11:4-39; *see also id.* at Abstract, 4:28-33, 6:65-7:5, claim 3.

¹³⁰ *Id.* at 4:25-33, 11:11-39; *see also id.* at Abstract, 3:14-18.





The '842 Patent refers to sub-objects and a memory scheduler as examples of code resources.¹³¹ A POSITA would have understood that Beetcher's module sub-objects are sub-objects.¹³²

Based on Beetcher's description, a POSITA would have understood that one sub-object in module 300 is a first code resource providing a specified underlying functionality when installed on the customer's computer system 101 and unlocked using the license information (key).¹³³

¹³¹ '842 Patent at 11:55-65, 15:36-42.

¹³² Silva Declaration at ¶¶ 65-66.

¹³³ *Id.* at ¶ 67.

d) *Element 12.3: “encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key encoded software code”*

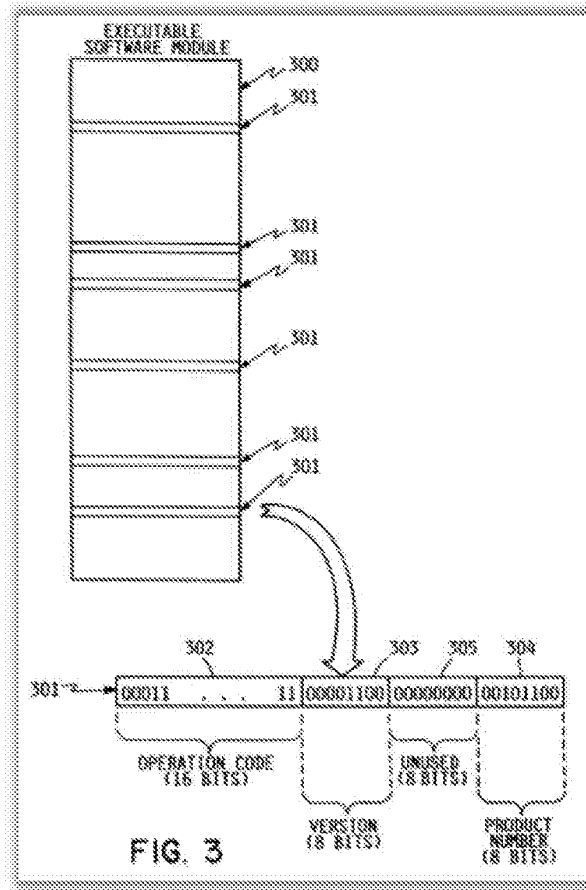
Beetcher discloses element 12.3. Beetcher describes encoding its software code by the distributor system that includes development system 125 and marketing system 124, which may be “a single computer system performing both functions.”¹³⁴ Specifically, Beetcher describes encoding a first license key into the software code where that key is used to authorize access to the software product:

Software module 300 is part of a program product in compiled object code form which executes on system 101.... [T]he actual executable code operates at executable code level 403, as shown by the box in broken lines. The executable code contains entitlement verification triggering instructions 301 (only one shown), which are executed by horizontal microcode check lock function 422.¹³⁵

This encoding is illustrated in Figure 3:

¹³⁴ Beetcher at 5:37-58, 6:41-65, 11:4-39.

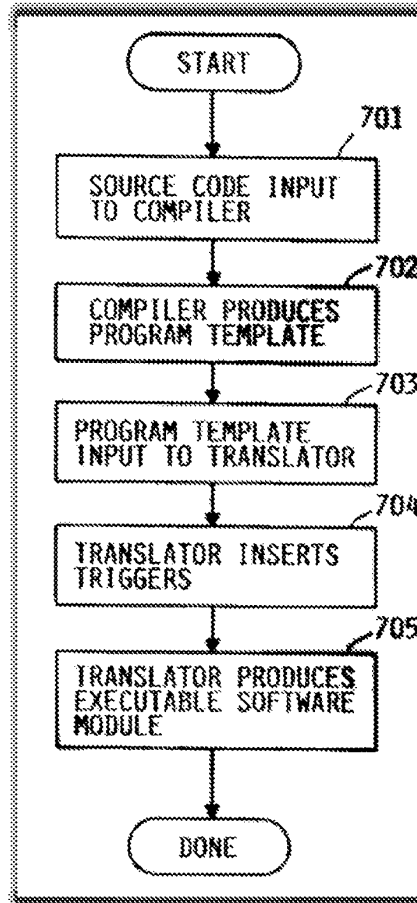
¹³⁵ *Id.* at 8:13-23; *see also id.* at 4:3-21, 6:20-55, 7:39-44, 8:58-67, 9:51-56, 10:22-38.



The computer in Betcher's development system 125 performs the encoding, as shown in Figure 7 at step 704, detailed as: "The program template serves as input to translator 127 at step 704, along with its product number and version number identification. Translator 127 automatically generates a substantial number of entitlement verification triggers, inserts them in random locations in the object code"¹³⁶

¹³⁶ *Id.* at 9:10-16; *see also id.* at 5:38-47, 9:1-10, 9:16-20, Fig. 7; Silva Declaration at ¶¶ 70-72.

Moreover, the computer in Beetcher's development system 125 uses an encoding algorithm to encode the first license key. Beetcher's system uses a set of instructions, as shown in Figure 7, to encode triggers into the software code to form the first license key.¹³⁷



The compiler begins the process by producing a template (step 702), next the template is input into the translator (step 703), then the translator encodes the triggers/license keys into the code (step 704), and finally the translator resolves references after key insertion to produce the executable module.¹³⁸ As such, a POSITA would have understood Beetcher's Figure 7 illustrates

¹³⁷ Beetcher at 9:10-16; *see also id.* at 5:38-47, 9:1-10, 9:16-20, Fig. 7; Silva Declaration at ¶ 73.

¹³⁸ Beetcher at 9:6-20, Fig. 7.

an encoding algorithm.¹³⁹ Beetcher's encoding process is further described with respect to element 11.3.

Moreover, during the original prosecution, Patent Owner specified that "[e]ncoding using a key and an algorithm is known."¹⁴⁰ As such, a POSITA would have understood that Beetcher's encoding technique necessarily includes a first license key and an encoding algorithm to form a first license key encoded software code¹⁴¹.

- e) ***Element 12.4: "wherein, when installed on a computer system, said first license key encoded software code will provide said specified underlying functionality only after receipt of said first license key"***

Beetcher discloses element 12.4. Specifically, Beetcher explains that its first license key encoded software code provides the specified underlying functionality only after receipt of the first license key.¹⁴² For instance, Beetcher states:

For support of such a traditional compilation path where the object code format is known by customers, additional barriers to patching of the object code to nullify or alter the entitlement triggering instructions may be appropriate. One such additional barrier would be to define the entitlement triggering instruction to simultaneously perform some other function. In this case, it is critical that the alternative function performed by the triggering instruction can not be performed by any other simple instruction. The alternative function must be so selected that any compiled software module will be reasonably certain of containing a number of instructions performing the function. If these criteria are met, the compiler can automatically generate the object code to perform the alternative function (and simultaneously, the entitlement verification trigger) as part of its normal compilation procedure. This definition would provide a significant barrier to patching of the object code to nullify the entitlement triggering instructions.¹⁴³

¹³⁹ Silva Declaration at ¶ 74.

¹⁴⁰ Ex. 2, Prosecution History at 519.

¹⁴¹ Silva Declaration at ¶¶ 70-75.

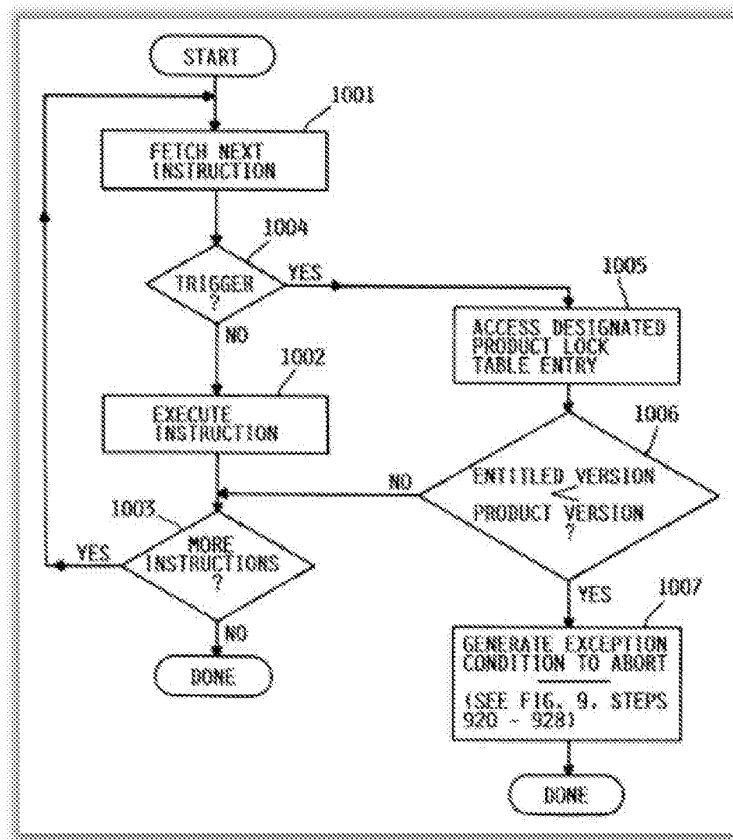
¹⁴² Beetcher at 6:58-65, 11:4-39; *see also id.* at Abstract, 3:14-18, 4:25-33, 6:65-7:5, claim 3.

¹⁴³ *Id.* at 11:10-28.

And as described with respect to element 12.3, Beecher teaches encoding the triggering instructions into the software code that is decoded via the first license key.

Beecher's Figure 10, as provided below, illustrates providing the software's underlying functionality based on the first license key (trigger information). For instance, Beecher explains:

System 101 executes the module by fetching (step 1001) and executing (step 1002) object code instructions until done (step 1003). If any instruction is an entitlement verification triggering instruction 301 (step 1004) check lock function 422 is invoked. Check lock function 422 accesses the product lock table entry 601 corresponding to the product number contained in the triggering instruction at step 1005. If the version number in product lock table 460 is equal to or greater than the version number 303 contained in triggering instruction 301, the software is entitled to execute (step 1006).¹⁴⁴



¹⁴⁴ *Id.* at 10:49-60; *see also id.* at 10:48-49, 10:60-11:3; Silva Declaration at ¶¶ 78-82.

Accordingly, Beetcher discloses claim 12.

3. Beetcher Anticipates Independent Claim 13.

a) *Preamble: “A method for encoding software code using a computer having a processor and memory, comprising”*

Under the broadest reasonable construction, the preamble is non-limiting. Nevertheless, Beetcher discloses claim 13’s preamble. Claim 13’s preamble is the same as claim 12’s preamble. As explained above, Beetcher discloses a method for encoding software using a computer with a processor and memory. As such, Beetcher teaches this preamble.¹⁴⁵

b) *Element 13.1: “storing a software code in said memory”*

Element 13.1 is identical to element 12.1. As explained above, Beetcher discloses each limitation of element 12.1. For the same reasons, Beetcher teaches element 13.1.¹⁴⁶

c) *Element 13.2: “wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system”*

Element 13.2 is identical to element 12.2. As explained above, Beetcher discloses each limitation of element 12.2. For the same reasons, Beetcher teaches element 13.2.¹⁴⁷

d) *Element 13.3: “modifying, by said computer, using a first license key and an encoding algorithm, said software code, to form a modified software code; and wherein said modifying comprises encoding said first code resource to form an encoded first code resource”*

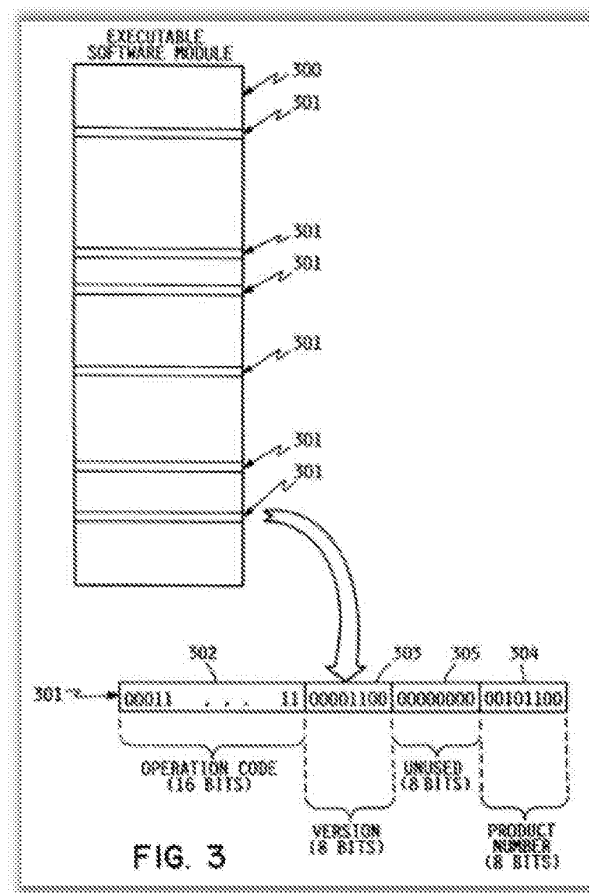
Beetcher discloses element 13.3. As described with respect to element 12.3, Beetcher’s distributor system includes a computer that encodes software code using a first license key (e.g., triggering information) and an encoding algorithm (e.g., Figure 7). And Beetcher’s encoding

¹⁴⁵ Silva Declaration at ¶ 85.

¹⁴⁶ *Id.* at ¶ 87.

¹⁴⁷ *Id.* at ¶ 89.

process modifies the software code by inserting triggering information into the code.¹⁴⁸ For instance, Beetcher details that system inputs compiled software code into a translator which modifies the code by “automatically generat[ing] a substantial number of entitlement verification triggers” and “insert[ing] them in random locations in the object code,” as shown in Figure 7’s steps 703 and 704.¹⁴⁹ Figure 3 illustrates this modifying by inserting triggering information 301 to form a modified software code:



¹⁴⁸ Beetcher at 8:13-23, 9:1-20; *see also id.* at 5:38-47, 9:1-10, 9:16-20, Fig. 7; Silva Declaration at ¶ 91.

¹⁴⁹ Beetcher at 9:11-15.

As described with respect to element 12.2, Beetcher's software code includes a series of code resources corresponding to sub-objects. And Beetcher teaches a code resource is modified to encode the first code resource via the triggering information.¹⁵⁰ For instance, Beetcher teaches:

For support of such a traditional compilation path where the object code format is known by customers, additional barriers to patching of the object code to nullify or alter the entitlement triggering instructions may be appropriate. One such additional barrier would be to define the entitlement triggering instruction to simultaneously perform some other function. In this case, it is critical that the alternative function performed by the triggering instruction can not be performed by any other simple instruction. The alternative function must be so selected that any compiled software module will be reasonably certain of containing a number of instructions performing the function. If these criteria are met, the compiler can automatically generate the object code to perform the alternative function (and simultaneously, the entitlement verification trigger) as part of its normal compilation procedure. This definition would provide a significant barrier to patching of the object code to nullify the entitlement triggering instructions.¹⁵¹

A POSITA would have understood that such modification results in an encoded first code resource.¹⁵²

Moreover, during the original prosecution, Patent Owner specified that "[e]ncoding using a key and an algorithm is known."¹⁵³ As such, a POSITA would have understood that Beetcher's encoding technique necessarily includes a first license key and an encoding algorithm to form a modified encoded first code resource.¹⁵⁴

¹⁵⁰ *Id.* at 4:25-33, 11:11-39; *see also id.* at Abstract, 3:14-18.

¹⁵¹ *Id.* at 11:10-28.

¹⁵² Silva Declaration at ¶ 92.

¹⁵³ Ex. 2, Prosecution History at 519.

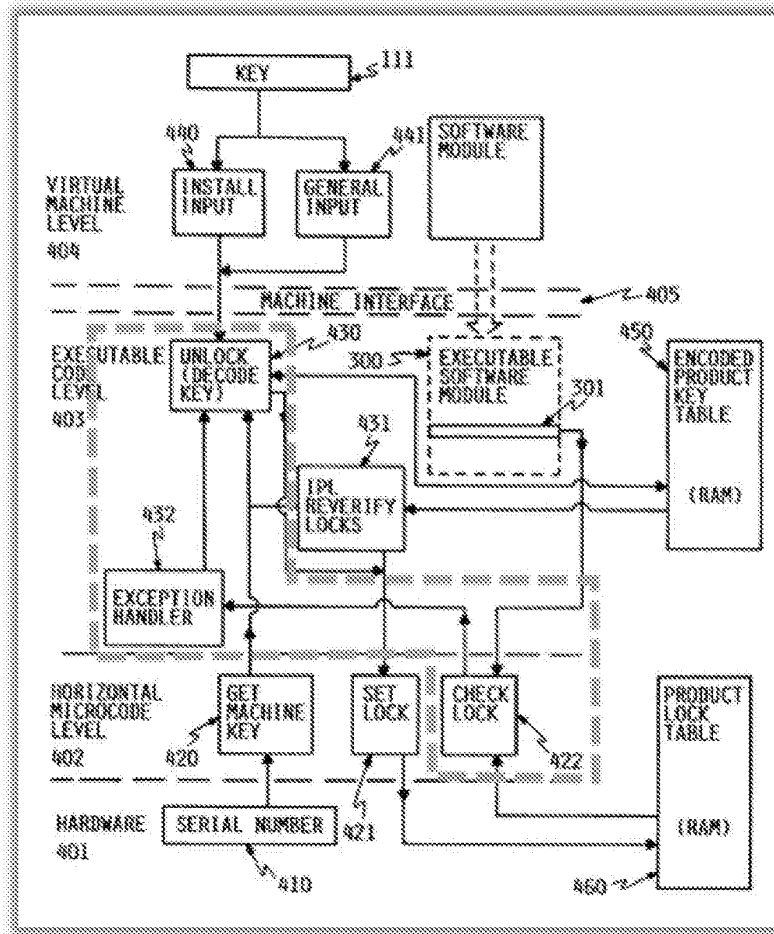
¹⁵⁴ Silva Declaration at ¶ 93.

e) ***Element 13.4: “wherein said modified software code comprises said encoded first code resource, and a decode resource for decoding said encoded first code resource”***

Beetcher discloses element 13.4. Beetcher explains that its modified software code includes a decode resource for decoding the encoded first code resource. Specifically, Beetcher teaches that executing a trigger 301 invokes check lock function 422, which results in accessing “unlock (decode key)” function 430 upon confirmation that the customer possesses the software’s license key.¹⁵⁵ Beetcher’s Figure 4, as annotated below, illustrates the decode resource (dashed perimeter) of the modified software code:¹⁵⁶

¹⁵⁵ Beetcher at 10:22-39, 10:52-65, Figs. 9b, 10; *see also id.* at 7:16-38, 8:18-22, 9:49-10:7.

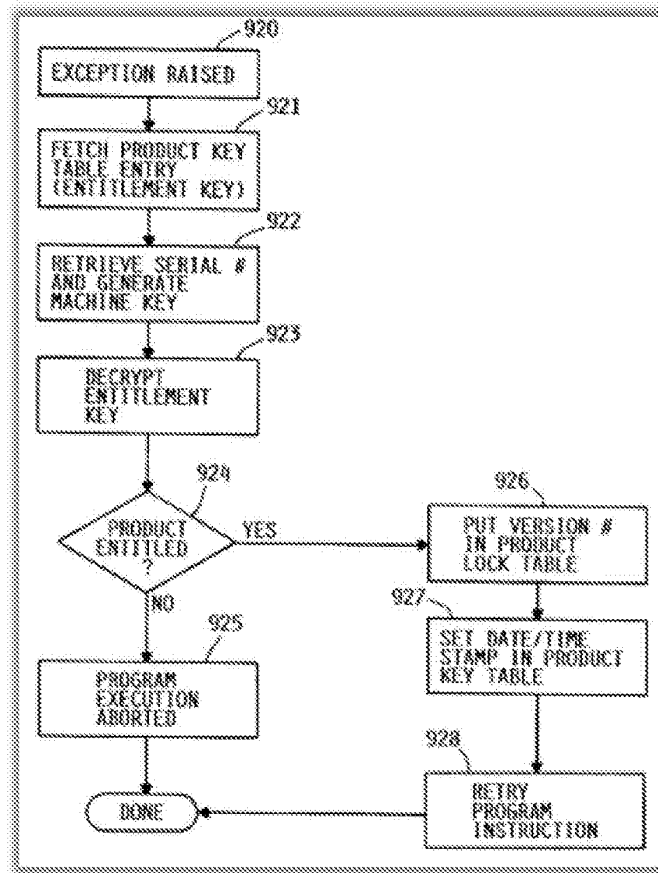
¹⁵⁶ Silva Declaration at ¶ 96.



f) **Element 13.5:** “wherein said decode resource is configured to decode said encoded first code resource upon receipt of said first license key”

Beecher discloses element 13.5. Beecher specifies that its decode resource decodes the encoded first code resource upon receipt of the license key. Beecher, for example, states that unlock routine 430 “fetches the encrypted entitlement key from ... table 450 ... and decodes the entitlement key The triggering instruction is then retried and program execution continues at

step 928.¹⁵⁷ And Beetcher's Figure 9b illustrates accessing the decode resource to decode the encoded first code resources based on the entitlement key, reflected in steps 921 to 928:



As such, a POSITA would have understood that Beetcher's decode resource is configured to decode the encoded first code resource based on first license key.¹⁵⁸

Accordingly, Beetcher discloses claim 13.

¹⁵⁷ Beetcher at 10:27-38.

¹⁵⁸ Silva Declaration at ¶¶ 99-100.

4. Beetcher Anticipates Independent Claim 14.

a) *Preamble: “A method for encoding software code using a computer having a processor and memory, comprising”*

Under the broadest reasonable construction, the preamble is non-limiting. Nevertheless, Beetcher discloses claim 14’s preamble. Claim 14’s preamble is the same as each of claim 12 and 13’s preamble. As explained above, Beetcher discloses a method for encoding software using a computer with a processor and memory. As such, Beetcher teaches this preamble.¹⁵⁹

b) *Element 14.1: “storing a software code in said memory”*

Element 14.1 is identical to element 12.1. As explained above, Beetcher discloses each limitation of element 12.1. For the same reasons, Beetcher teaches element 14.1.¹⁶⁰

c) *Element 14.2: “wherein said software code defines software code interrelationships between code resources that result in a specified underlying functionality when installed on a computer system”*

Beetcher discloses element 14.2. Beetcher details that its software code is compiled into executable code by compiler 126. This compiler works with translator 127 to compile the software sub-objects and insert triggering information.¹⁶¹ And Beetcher specifies that translator 127 “resolves references” in the software code, which corresponds to defining code interrelationships between code resources.¹⁶² As shown in steps 701 and 702 of Figure 7, Beetcher teaches its software code is input into compiler 126 that produces a template of the software code:¹⁶³

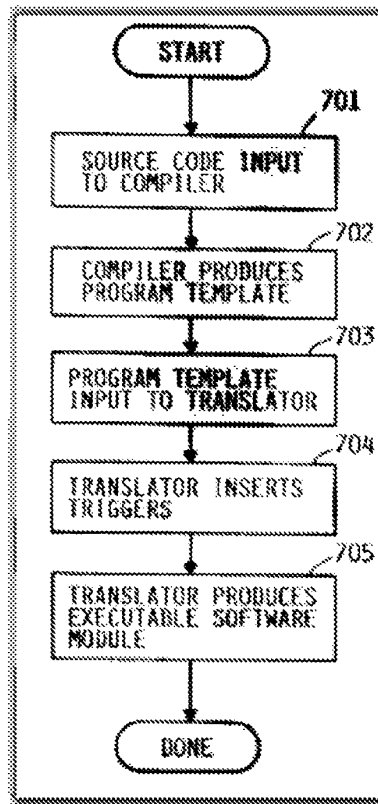
¹⁵⁹ *Id.* at ¶ 103.

¹⁶⁰ *Id.* at ¶ 105.

¹⁶¹ Beetcher at 8:14-17.

¹⁶² *Id.* at 9:11-18; Silva Declaration at ¶ 107.

¹⁶³ Beetcher at 8:14-17, 9:1-20, Fig. 7; *see also id.* at 5:37-39, 6:41-45, 7:63-66



A POSITA would have understood that this software code template also defines the code interrelationships between the code resources.¹⁶⁴ As the Patent Owner specified during the original prosecution, software code interrelationships are defined during the compiling process of conventional software applications:

What the examiner has implied by alleging that the "specification ... fails to teach or mention 'software code interrelationships'" is that software code interrelationships were somehow unknown in the art, which clearly is not the case. As admitted, in the specification at the beginning of paragraph [0051], an "application" comprises "sub-objects" whose "order in the computer memory is of vital importance" in order to perform an intended function. And as admitted further in paragraph [0051], "When a program is compiled, then, it consists of a collection of these sub-objects, whose exact order or arrangement in memory is not important, so long as any sub-object which uses another sub-object knows where in memory it can be found." Paragraph [0051] of course refers to conventional applications. Accordingly, that is admittedly a discussion of

¹⁶⁴ Silva Declaration at ¶ 108.

what is already know by one skilled in the art. Accordingly, the examiner's statement that the specification lacks written description support for "software code interrelationships" is inconsistent with the fact that such **interrelationships were explained in paragraphs [0051] and [0052] as a fundamental basis of pre-existing modem computer programs.**¹⁶⁵

Moreover, during the original prosecution, Patent Owner specified that "interrelationships between code resource are not that which is novel."¹⁶⁶ Based on Patent Owner's concessions, it is clear that a POSITA would have understood that Beetcher's code necessarily defines code interrelationships between code resources¹⁶⁷.

Beetcher further teaches that the code resource interrelationships specify the underlying application functionalities when installed on the customer's computer 101. For instance, Beetcher's software code includes multiple entitlement verification triggers.¹⁶⁸ And Beetcher details that certain code resources include triggering instruction that controls the underlying functionalities of the software code:

[An] additional barrier would be to define the entitlement triggering instruction to simultaneously perform some other function.... The alternative function must be so selected that any compiled software module will be reasonably certain of containing a number of instructions performing the function. If these criteria are met, the compiler can automatically generate the object code to perform the alternative function (and simultaneously, the entitlement verification trigger) as part of its normal compilation procedure. This definition would provide a significant barrier to patching of the object code to nullify the entitlement triggering instructions.¹⁶⁹

Beetcher further explains that "the triggering instruction is also a direct instruction to perform some other useful work [E]xecution of the triggering instruction causes system 101 to

¹⁶⁵ Ex. 2, Prosecution History at 519.

¹⁶⁶ *Id.*

¹⁶⁷ Silva Declaration at ¶ 109.

¹⁶⁸ Beetcher at 4:15-33, 9:1-3, 10:22-34, Fig. 3; *see also id.* at 6:45-65, 8:19-22, 10:52-11:39.

¹⁶⁹ *Id.* at 11:14-28; *see also id.* at 4:25-33, 6:58-65.

perform some other operation simultaneous with the entitlement verification.”¹⁷⁰ As such, a POSITA would have understood that the code interrelationships between Beetcher’s code resources result in a specified underlying functionality once installed.¹⁷¹

- d) ***Element 14.3: “encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key encoded software code”***

Element 14.3 is identical to element 12.3. As explained above, Beetcher discloses each limitation of element 12.3. For the same reasons, Beetcher teaches element 14.3.

Moreover, during the original prosecution, Patent Owner specified that “[e]ncoding using a key and an algorithm is known” and that “an interrelationship in software code is necessarily defined by digital data, and digital data can obviously be encoded by an encoding process.”¹⁷² As such, a POSITA would have understood that Beetcher’s encoding technique necessarily includes a first license key and an encoding algorithm to form a first license key encoded software code.¹⁷³

- e) ***Element 14.4: “in which at least one of said software code interrelationships are encoded”***

Beetcher discloses element 14.4. As described with respect to element 14.2, Beetcher teaches that its software code defines code interrelationships between code resources and triggering information 301 in the code control certain underlying software functionality. And Beetcher details that triggering information 301 is encoded into the software code.¹⁷⁴ For

¹⁷⁰ *Id.* at 6:58-65 (Beetcher specifies that these functions are those “which do not require that an operand for the action be specified in the instruction.”).

¹⁷¹ Silva Declaration at ¶¶ 110-11.

¹⁷² Ex. 2, Prosecution History at 519.

¹⁷³ Silva Declaration at ¶¶ 114-15.

¹⁷⁴ Beetcher at 4:25-33, 6:58-65, 11:4-39.

instance, Beetcher explains that the triggering instructions will be encoded into the code resources controlling software functionality:

[An] additional barrier would be to define the entitlement triggering instruction to simultaneously perform some other function.... The alternative function must be so selected that any compiled software module will be reasonably certain of containing a number of instructions performing the function. If these criteria are met, the compiler can automatically generate the object code to perform the alternative function (and simultaneously, the entitlement verification trigger) as part of its normal compilation procedure. This definition would provide a significant barrier to patching of the object code to nullify the entitlement triggering instructions.¹⁷⁵

And Beetcher details that “the triggering instruction is also a direct instruction to perform some other useful work [E]xecution of the triggering instruction causes system 101 to perform some other operation simultaneous with the entitlement verification.”¹⁷⁶ Accordingly, a POSITA would have understood that this encoded triggering information includes encoded code interrelationship of the coder resources.¹⁷⁷

Accordingly, Beetcher discloses claim 14.

B. SNQ-2: Claims 11, 12, 13, and 14 are Anticipated by Beetcher '072 Under 35 U.S.C. §§ 102(a), (b).

Beetcher '072 anticipates claims 11, 12, 13, and 14 under 35 U.S.C. §§ 102(a), (b).

1. Beetcher '072 Anticipates Independent Claim 11.

a) Preamble: “A method for licensed software use, the method comprising”

Under the broadest reasonable construction, the preamble is non-limiting. Nevertheless, Beetcher '072 discloses claim 11’s preamble. Specifically, Beetcher '072 describes a method of

¹⁷⁵ *Id.* at 11:14-28; *see also id.* at 4:25-33, 6:58-65.

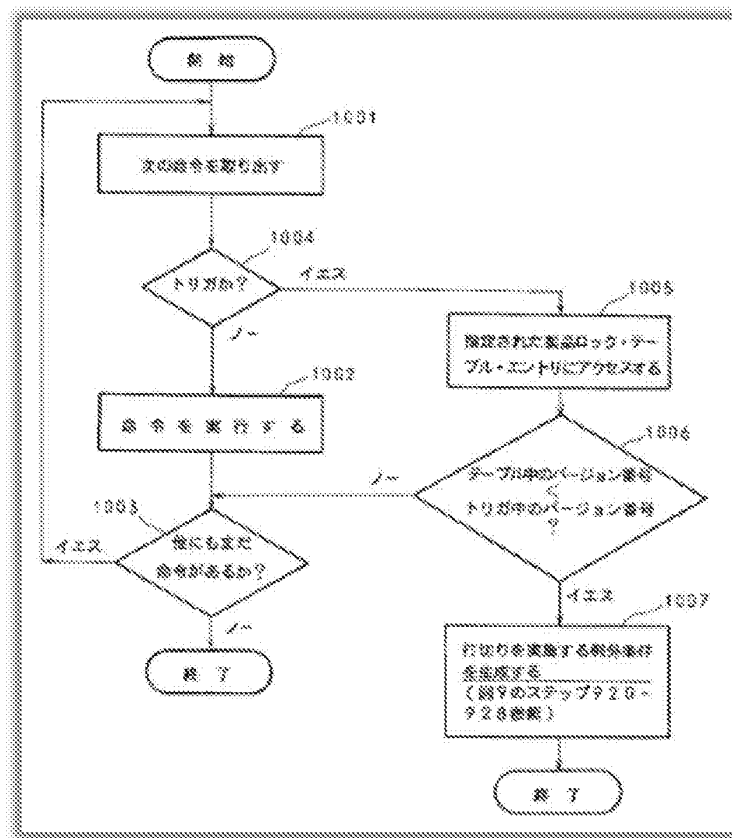
¹⁷⁶ *Id.* at 6:58-65 (Beetcher specifies that these functions are those “which do not require that an operand for the action be specified in the instruction.”).

¹⁷⁷ Silva Declaration at ¶¶ 117-19.

controlling access to licensed software using an encrypted entitlement key.¹⁷⁸ Beetcher '072, for instance, summarizes its invention as:

According to the present invention, software is distributed without the qualification grant for performing. Execution of software is attained by the enciphered qualification grant key which is distributed independently. This qualification grant key contains a plurality of qualification grant bits which instruct the consecutive numbers of the machine with which software is licensed to it, and which software module has the qualification it runs by that machine.¹⁷⁹

Beetcher '072's Figure 10, as provided below, illustrates the use of an entitled version of software based on the customer's license:



¹⁷⁸ Beetcher '072 at Abstract, ¶¶ 0020, 0022, 0043; see also *id.* at ¶¶ 0001, 0004, 0016 (See Ex. 5 for English translation).

¹⁷⁹ Beetcher '072 at ¶ 0020 (See Ex. 5 for English translation).

As such, Beetcher '072 teaches this preamble.¹⁸⁰

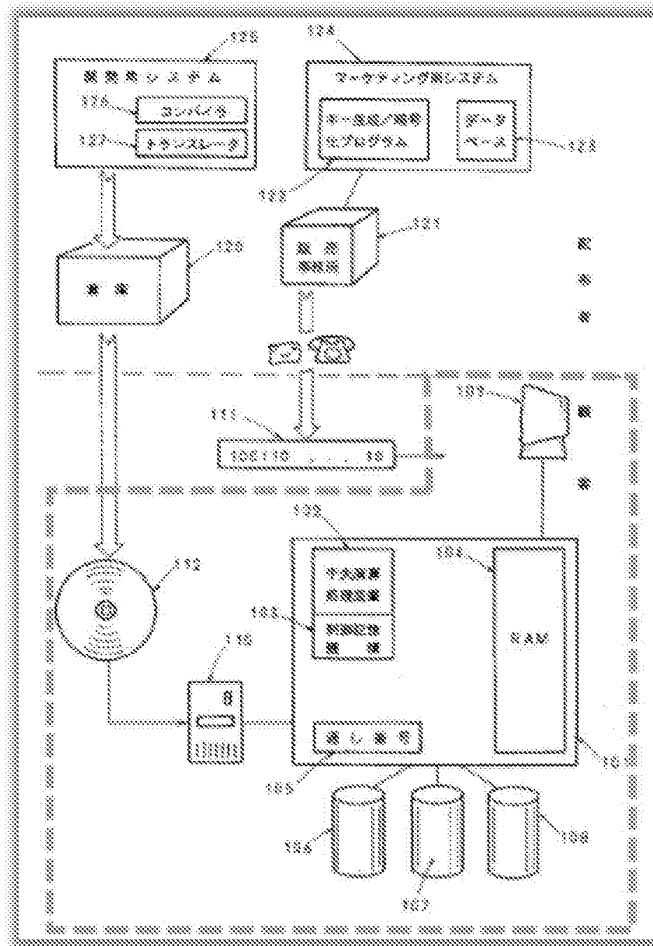
- b) *Element 11.1: “loading a software product on a computer, said computer comprising a processor, memory, an input, and an output, so that said computer is programmed to execute said software product”***

Beetcher '072 discloses element 11.1. Specifically, Beetcher '072's system includes a customer computer 101 including a CPU 102, memory 104, and storage devices 106-108.¹⁸¹ This customer computer 101 also includes a media reader 110 (i.e., an input) and an operator console 109 (i.e., an output).¹⁸² As shown below in annotated Figure 1, Beetcher '072 discloses a computer having software product 112 loaded for execution (dashed perimeter):

¹⁸⁰ Silva Declaration at ¶¶ 122-25.

¹⁸¹ Beetcher '072 at ¶ 0023, Fig. 1 (*See* Ex. 5 for English translation).

¹⁸² Beetcher '072 at ¶¶ 0023, 0027, Fig. 1 (*See* Ex. 5 for English translation).



Beecher '072 details that the customer loads the media, such as an optical disk, containing a software product onto the computer for execution:¹⁸³

[S]oftware media 112 comprise one sheet or a plurality of read-only optical discs, and the medium reader 110 is an optical disc reader. However, please understand that an electronic distribution medium and other distribution media can also be used. If the software media 112 are received, a customer will load a desired software module to the system 101 from the medium reader 110, and will usually memorize the software module to the memory storage 106-108.¹⁸⁴

¹⁸³ Silva Declaration at ¶¶ 127-29.

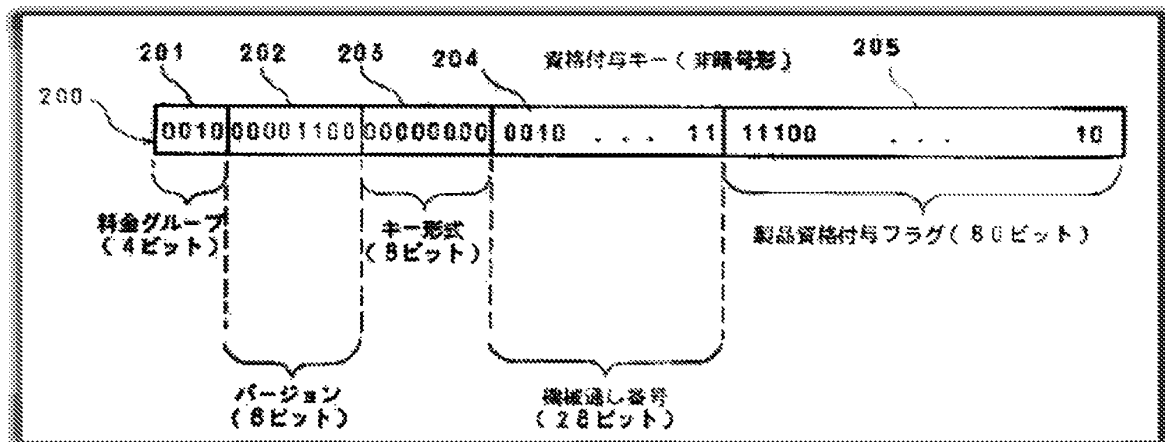
¹⁸⁴ Beecher '072 at ¶ 0027; see also *id.* at Abstract, ¶¶ 0014, 0040, Fig. 1, claim 6 (*See Ex. 5 for English translation*).

c) **Element 11.2: “said software product outputting a prompt for input of license information”**

Beetcher '072 discloses element 11.2. Specifically, Beetcher '072 explains that its software product contains a user interface routine for the customer to input a license key into the computer before the product can be used.¹⁸⁵ For instance, Beetcher '072 explains that the software product prompts the user it input license information:

The support of this operation system contains two user interface routines required to support the input of a qualification grant key on the virtual-machine level 404. The general input routine 441 is used for processing an input in normal operation. The installation input routine 440 special to inputting a qualification grant key is required during the initial introduction of an operation system. The thing which needs this is because the portion of an upper level operating system is treated as other program products by the present invention from the machine interface level 405. Namely, such a portion has product number and the target code is subject to the influence of a qualification verification trigger.¹⁸⁶

Beetcher '072's Figure 2 illustrates this license information in unencrypted form



Beetcher '072 further explains that the software's “installation input routine 440 has a dialog with an operator, and receives an input” of the customer's license information during the

¹⁸⁵ Beetcher '072 at ¶ 0033; *see also id.* at ¶ 0010 (*See Ex. 5* for English translation).

¹⁸⁶ Beetcher '072 at ¶ 0033 (*See Ex. 5* for English translation).

software's initial installation.¹⁸⁷ And as discussed with respect to element 11.1, the customer's computer includes an operator console 109 shown with a monitor and keyboard that "receive the input from an operator."¹⁸⁸

d) *Element 11.3: "said software product using license information entered via said input in response to said prompt in a routine designed to decode a first license code encoded in said software product"*

Beetcher '072 discloses element 11.3. Upon inserting the software's disk 112, Beetcher '072 explains that the operator console prompts the customer to enter license information.¹⁸⁹ Beetcher '072 details that the customer enters entitlement key 111, i.e., license information, in response to the prompt initiated by install input routine 440.¹⁹⁰ After entering that key, Beetcher '072 teaches that the customer's computer uses a decode key to initiate unlock routine 430 to decode the license code encoded in the software product.¹⁹¹ Beetcher '072's Figures 4 and 9a, which are provided below, show the software using the key (i.e., license information) entered by the customer to decode a first license code encoded in the software product. For instance, annotated Figure 4 illustrates that the install input routine 440 starts unlock routine 430 once the customer inputs key 111 into the computer.¹⁹² And "unlocking routine 430 decodes the qualification grant key 111 using a peculiar machine key" (dashed perimeter).¹⁹³

¹⁸⁷ Beetcher '072 at ¶ 0040; *see also id.* at Fig. 4 (reference number 440), claim 6 (*See Ex. 5 for English translation*).

¹⁸⁸ Beetcher '072 at ¶ 0023; *see also id.* at ¶¶ 0025, 0033, 0039, Fig. 1 (*See Ex. 5 for English translation*); Silva Declaration at ¶¶ 131-133.

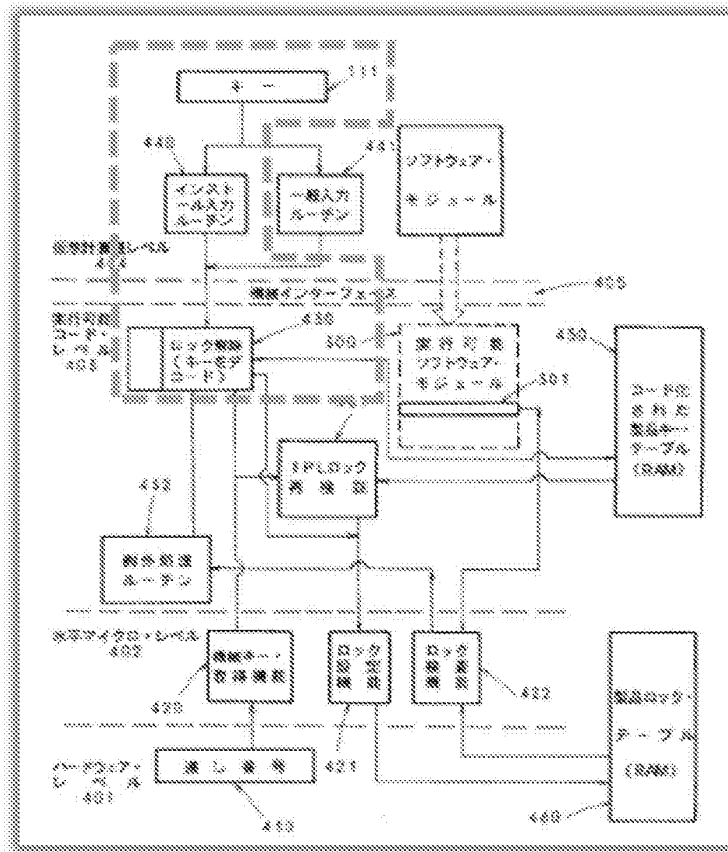
¹⁸⁹ *E.g.*, Beetcher '072 at ¶¶ 0033, 0040, Figs. 1, 9a.

¹⁹⁰ Beetcher '072 at ¶ 0033; *see also id.* at ¶ 0040, Figs. 1, 4, claim 6 (*See Ex. 5 for English translation*).

¹⁹¹ Beetcher '072 at ¶¶ 0032, 0040; *see also id.* at ¶¶ 0030, 0037, Figs. 4, 9a (*See Ex. 5 for English translation*).

¹⁹² Beetcher '072 at ¶¶ 0033, 0040 (*See Ex. 5 for English translation*).

¹⁹³ Beetcher '072 at ¶ 0032; *see also id.* at ¶¶ 0037, 0041 (*See Ex. 5 for English translation*).

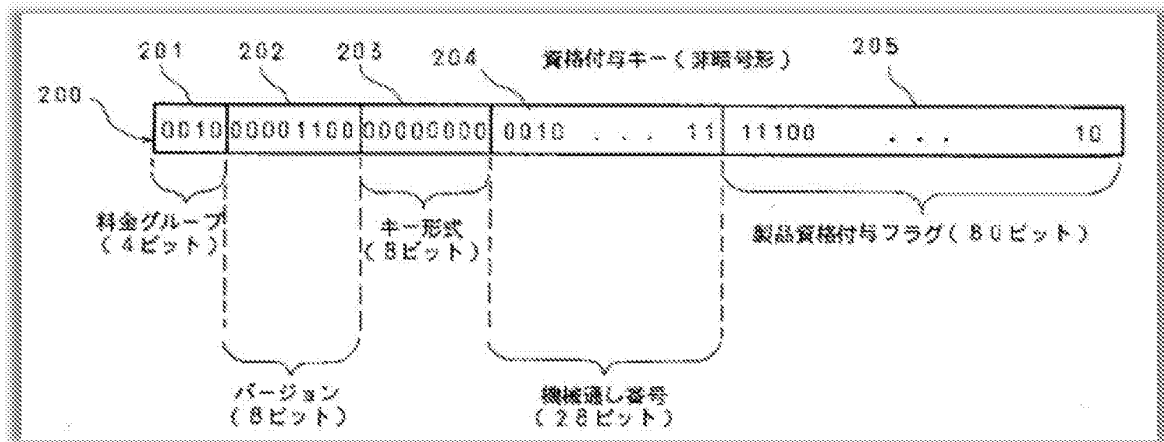


Beetcher '072 details that unlock routine 430 “handles the decoding process,” which is illustrated in Figure 9a’s steps 902-909: “The lock release routine 430 makes the machine-key acquisition function 420 search machine consecutive numbers with Step 902, and makes it generate a machine key at it. Subsequently, the lock release routine 430 decodes the qualification grant key 111 at Step 903 using a machine key.”¹⁹⁴

Beetcher '072 specifies that its unencrypted entitlement key includes multiple fields, which includes version field 202 specifying entitled version levels and product entitlement flags

¹⁹⁴ Beetcher '072 at ¶0040 (See Ex. 5 for English translation).

205 specifying customer's accessible product numbers.¹⁹⁵ Beetcher '072's Figure 2 shows this license information with fields 201 to 205:



Beetcher '072's unlock routine 430 will complete the decoding process by building an encoded product key table (step 904), populating the key table for the relevant software product (steps 905-908), and saving the key table (step 909).¹⁹⁶ Beetcher '072 also specifies that the customer's RAM includes table 460 populated with products having entitlement keys.¹⁹⁷ Beetcher '072's software product uses the key's version and product number fields to decode a license code.

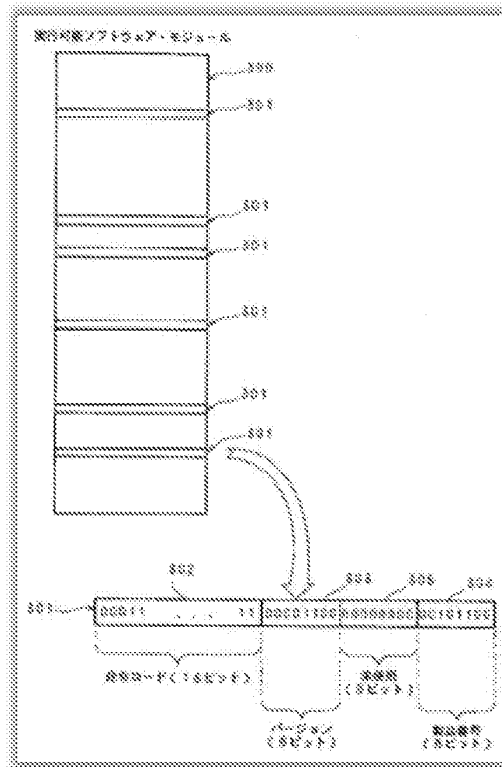
When compiling and translating the software code, Beetcher '072 explains that the code includes entitlement verification triggering instructions encoded into the software.¹⁹⁸ Beetcher '072's triggering instructions are encoded into the software when the software code is compiled and translated, as shown in Figure 3 provided below:

¹⁹⁵ Beetcher '072 at ¶ 0028.

¹⁹⁶ *Id.* at ¶ 0040 (See Ex. 5 for English translation).

¹⁹⁷ Beetcher '072 at ¶¶ 0032, 0036, 0041-42, Fig. 6, Fig. 9a.

¹⁹⁸ *Id.* at ¶¶ 0029, 0044; see also *id.* at ¶¶ 0021, 0033-34, 0037-38 (See Ex. 5 for English translation).



Betcher '072 explains that its software code verifies the customer is entitled to use the software when the code encounters a triggering instruction. When it encounters one of these instructions, Betcher '072's code accesses the license key information stored in the key table 460.¹⁹⁹ As such, a POSITA would have understood that Betcher uses its license information in a routine, such as check lock function 422, designed to decode a first license code encoded in a software product via the triggering instructions:²⁰⁰

When a command is the qualification verification trigger 301 (Step 1004), the lock checking feature 422 is called. At Step 1005, the lock checking feature 422 accesses the product locking table entry 601 to which it corresponds to the product number included in a qualification verification trigger. The qualification for the version number in the product locking table 460 being equal to the version number 303.

¹⁹⁹ Betcher '072 at ¶¶ 0043-44; *see also id.* at Abstract, ¶¶ 0034, 0037-38, Fig. 10 (*See Ex. 5 for English translation*).

²⁰⁰ Silva Declaration at ¶¶ 135-39.

contained in the qualification verification trigger 301, or performing software, in being larger than it is given (Step 1006). In this case, the lock checking feature 422 does not perform treatment beyond it, but a system proceeds to execution of the next target code command in a software module.²⁰¹

Moreover, Beetcher '072 teaches that the triggering instructions will be encoded into the code resources controlling software functionality:

[An] additional barrier[] is defining a qualification verification trigger, as other functions of a certain are performed simultaneously.... This alternate function must be selected so that any compiled software modules may include some commands which perform that function quite reliably. When having coincided in these criteria, the compiler can generate automatically the target code which performs the alternate function (it is also a qualification verification trigger simultaneously with it) as a part of the usual compilation order. This definition should bring about the important barrier to 'patching' of a target code which invalidates a qualification verification trigger.²⁰²

And Beetcher '072 details that "a qualification verification trigger is also the direct instruction ... which performs other useful work of a certain.... [I]f a trigger command is executed, the system 101 will perform other operations of a certain simultaneously with qualification verification."²⁰³

Accordingly, Beetcher '072 discloses claim 11.

2. Beetcher '072 Anticipates Independent Claim 12.

a) ***Preamble: "A method for encoding software code using a computer having a processor and memory, the method comprising"***

Under the broadest reasonable construction, the preamble is non-limiting.²⁰⁴

Nevertheless, Beetcher '072 discloses claim 12's preamble. Specifically, Beetcher '072 describes

²⁰¹ Beetcher '072 at ¶ 0043, Fig. 10.

²⁰² *Id.* at ¶ 0044; *see also id.* at ¶¶ 0021, 0029 (*See Ex. 5* for English translation); Silva Declaration at ¶ 140.

²⁰³ Beetcher '072 at ¶ 0029 (Beetcher '072 specifies that these functions are those "which does not need to divide, does not need to be ordering the operand for the processing and does not need to be specified") (*See Ex. 5* for English translation); Silva Declaration at ¶ 140.

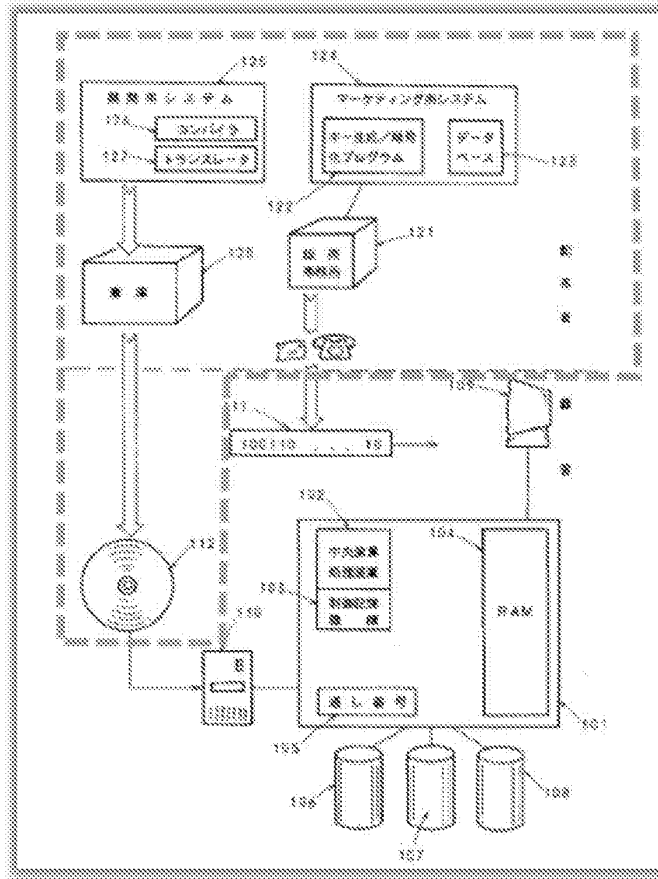
²⁰⁴ Claim 12's preamble recites "a computer" and claim 12's body recites "a computer system." It is unclear whether those elements refer to the same or separate computing devices. For

a method for encoding software code using a computer with a processor and memory. Beetcher '072 details that the software distributor has “computer system 125 for development contain the compiler 126 and the translator 127” where “[a] software module is recorded on the software recording medium 112” and “generation/enciphered program 122 of a qualification grant key, and the data base 123 containing customer data.”²⁰⁵ Beetcher '072 specifies these compiling and key generating functions may be performed by a single computer.²⁰⁶ Below annotated Figure 1 illustrates the distributor’s computer system distributing memory media 112 and complying encoded software code:

purposes of this Request and using the broadest reasonable interpretation consistent with the specification, it is assumed that the “computer” recited in the preamble is a device separate from the “computer system.”

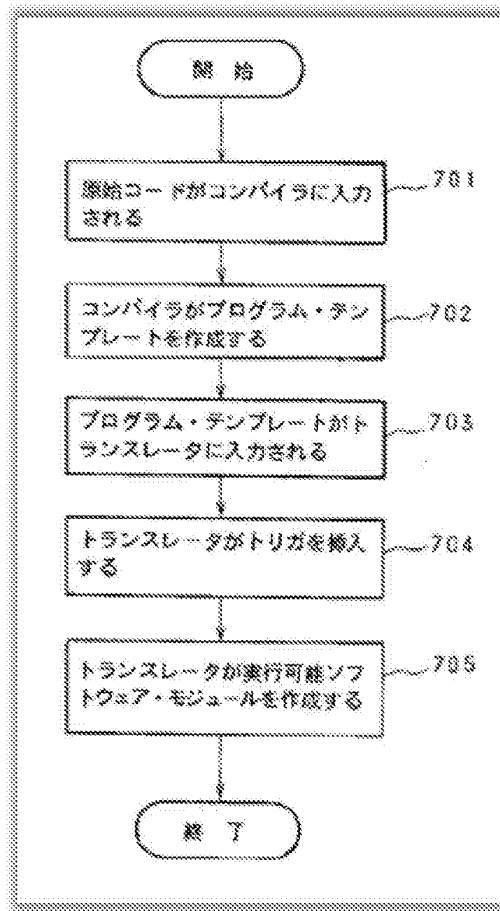
²⁰⁵ Beetcher '072 at ¶ 0024; *see also id.* at ¶ 0038 (*See* Ex. 5 for English translation).

²⁰⁶ Beetcher '072 at ¶ 0024 (*See* Ex. 5 for English translation).



Beetcher '072's Figure 7 illustrates the software code being encoded to include watermarking triggers decoded by the customer's licensing information.²⁰⁷

²⁰⁷ Beetcher '072 at ¶0038, Fig. 7 (See Ex. 5 for English translation).



As such, a POSITA would have understood that Beetcher '072's distributor compiles and stores the encode software code using a processor and memory akin to the console's CPU 102 and memory devices 1106-108. As expert Dr. Silva explains in his declaration (Ex. 9), Beetcher '072's computer would necessarily include a processor and memory in order to function.²⁰⁸

As such, Beetcher '072 teaches this preamble.²⁰⁹

²⁰⁸ Silva Declaration at ¶ 147.

²⁰⁹ *Id.* at ¶¶ 144-47.

b) Element 12.1: “storing a software code in said memory”

Beetcher '072 discloses element 12.1. Specifically, Beetcher '072 discloses a development system 125 for compiling and translating for the software code.²¹⁰ Beetcher '072 details that the software code is stored as disks 112 in warehouse 120. A POSITA would have understood that developer system 125 stores the compiled and translated code in memory and records that code onto disks 112 for distribution to customers. As expert Dr. Silva explains in his declaration (Ex. 9), Beetcher '072's computer would necessarily include store software code in memory in order to function.²¹¹

c) Element 12.2: “wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system”

Beetcher '072 discloses element 12.2. Specifically, Beetcher '072 explains that its software code includes multiple code resources that include a first code resource.²¹² Beetcher '072's code resources include software modules 300 (dashed box) including sub-objects within the code, as shown below in annotated Figure 4 and Figure 3.²¹³ These sub-objects control multiple functions of the software installed on the customer's computer system 101.²¹⁴ And Beetcher '072's software prevents unwanted “patching” of these sub-objects by including entitlement verification triggering instructions 301.²¹⁵

²¹⁰ Beetcher '072 at ¶¶ 0024, 0038 (*See* Ex. 5 for English translation).

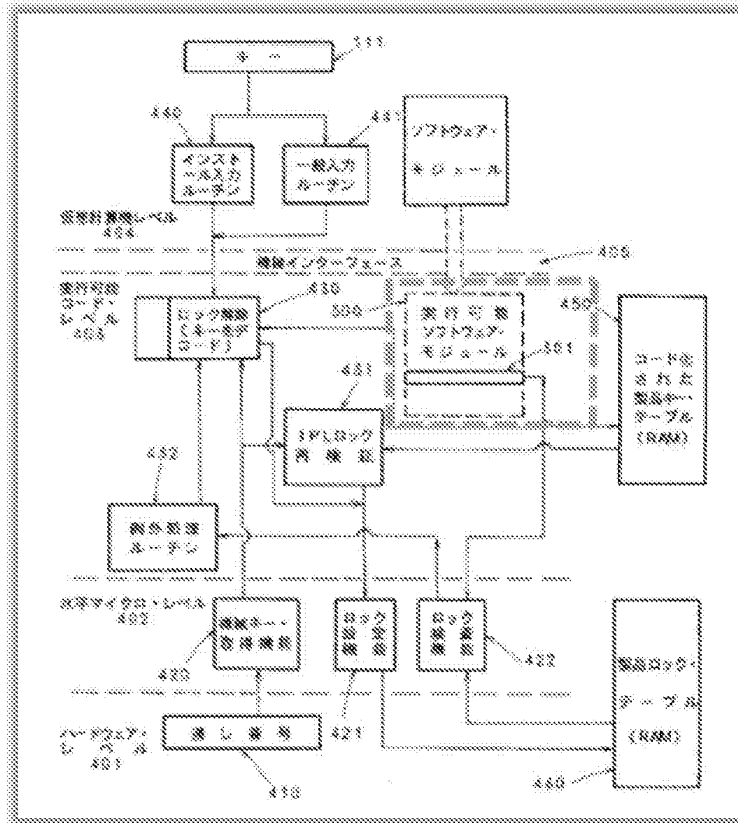
²¹¹ Silva Declaration at ¶ 150.

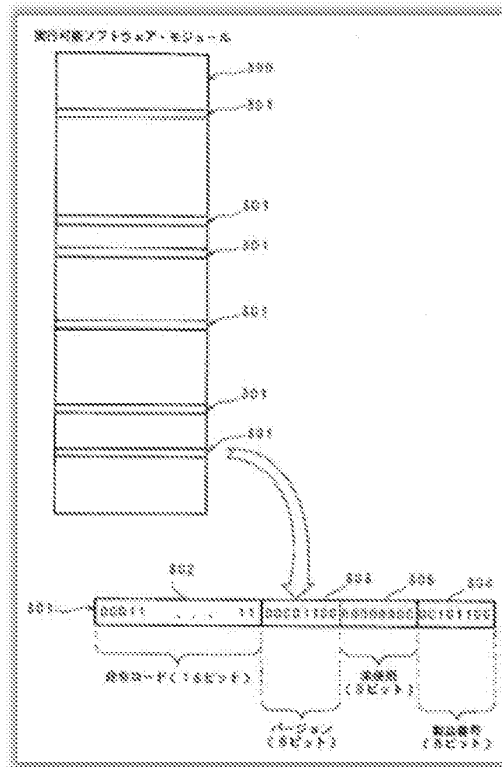
²¹² Beetcher '072 at ¶¶ 0024, 0026-27 (*See* Ex. 5 for English translation).

²¹³ Beetcher '072 at ¶¶ 0029, 0034, Fig. 4; *see also id.* at ¶ 0032, Fig. 3 (*See* Ex. 5 for English translation).

²¹⁴ Beetcher '072 at ¶¶ 0029, 0044; *see also id.* at Abstract, ¶¶ 0021, 0030, claim 3 (*See* Ex. 5 for English translation).

²¹⁵ Beetcher '072 at ¶¶ 0021, 0044; *see also id.* at Abstract, ¶ 0009 (*See* Ex. 5 for English translation).





The '842 Patent refers to sub-objects and a memory scheduler as examples of code resources.²¹⁶ A POSITA would have understood that Beetcher '072's module sub-objects are sub-objects.²¹⁷

Based on Beetcher '072's description, a POSITA would have understood that one sub-object in module 300 is a first code resource providing a specified underlying functionality when installed on the customer's computer system 101 and unlocked using the license information (key).²¹⁸

²¹⁶ '842 Patent at 11:55-65, 15:36-42.

²¹⁷ Silva Declaration at ¶¶ 153-54.

²¹⁸ *Id.* at ¶ 155.

d) *Element 12.3: “encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key encoded software code”*

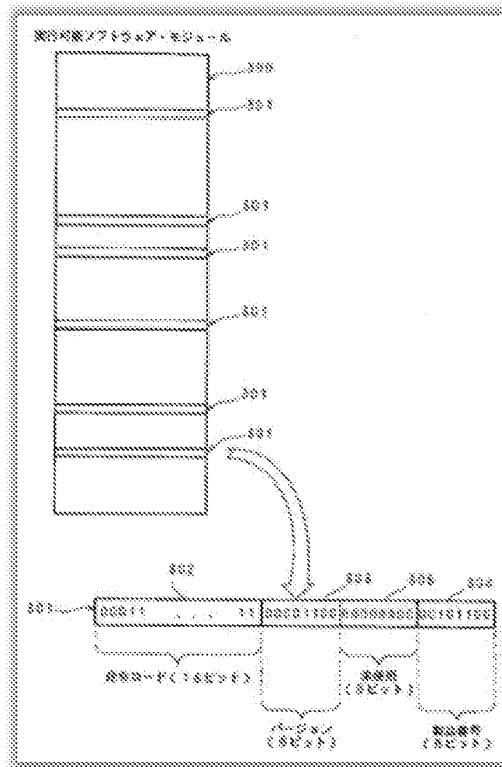
Beetcher '072 discloses element 12.3. Beetcher '072 describes encoding its software code by the distributor system, which includes development system 125 and marketing system 124, via a “single computer systems may be physically used performs both of functions.”²¹⁹ Specifically, Beetcher '072 describes encoding a first license key into the software code where that key is used to authorize access to the software product:

The software modules 300 are some program products of the compiled target code form which is performed on the system 101.... [T]he code which can actually be executed operates on the executable code level 403 as shown by the frame of the broken lines. The executable code contains the qualification verification trigger 301 (only one is shown in the figure) performed by the lock checking feature 422 of a horizontal microcode.²²⁰

This encoding is illustrated in Figure 3:

²¹⁹ Beetcher '072 at ¶¶ 0024, 0029, 0044 (*See* Ex. 5 for English translation).

²²⁰ Beetcher '072 at ¶ 0034; *see also id.* at ¶¶ 0020-21, 0028-29, 0032, 0037, 0040, 0041 (*See* Ex. 5 for English translation).

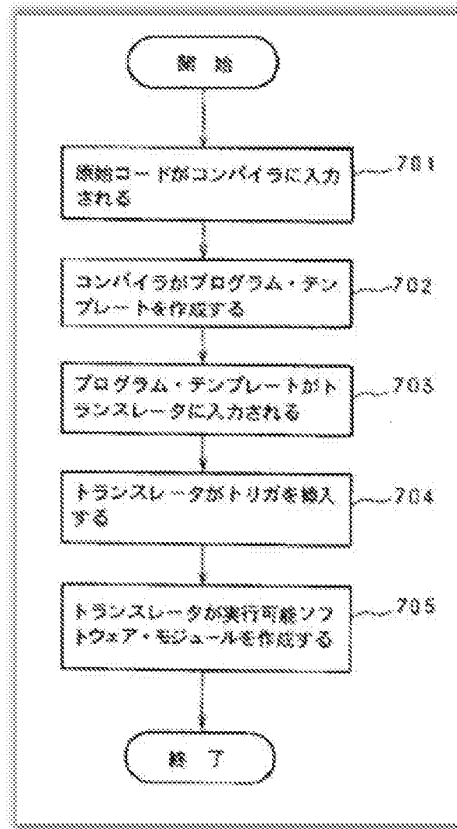


The computer in Beecher '072's development system 125 performs the encoding, as shown in Figure 7 at step 704, detailed as: "At Step 704, a program template identifies the product number and version number, and it works as an input to the translator 127. Automatically, the translator 127 generates most number of qualification verification triggers, inserts this in the random position in a target code"²²¹

Moreover, the computer in Beecher '072's development system 125 uses an encoding algorithm to encode the first license key. Beecher '072's system uses a set of instructions, as shown in Figure 7, to encode triggers into the software code to form the first license key.²²²

²²¹ Beecher '072 at ¶ 0038; *see also id.* at ¶ 0024, Fig. 7 (*See Ex. 5 for English translation*); Silva Declaration at ¶¶ 158-60.

²²² Beecher '072 at ¶ 0038; *see also id.* at ¶ 0024, Fig. 7 (*See Ex. 5 for English translation*); Silva Declaration at ¶ 161.



The compiler begins the process by producing a template (step 702), next the template is input into the translator (step 703), then the translator encodes the triggers/license keys into the code (step 704), and finally the translator resolves references after key insertion to produce the executable module.²²³ As such, a POSITA would have understood Beetcher '072's Figure 7 illustrates an encoding algorithm.²²⁴ Beetcher's encoding process is further described with respect to element 11.3.

²²³ Beetcher '072 at ¶0038, Fig. 7 (See Ex. 5 for English translation).

²²⁴ Silva Declaration at ¶162.

Moreover, during the original prosecution, Patent Owner specified that “[e]ncoding using a key and an algorithm is known.”²²⁵ As such, a POSITA would have understood that Beetcher ’072’s encoding technique necessarily includes a first license key and an encoding algorithm to form a first license key encoded software code²²⁶.

- e) ***Element 12.4: “wherein, when installed on a computer system, said first license key encoded software code will provide said specified underlying functionality only after receipt of said first license key”***

Beetcher ’072 discloses element 12.4. Specifically, Beetcher ’072 explains that its first license key encoded software code provides the specified underlying functionality only after receipt of the first license key.²²⁷ For instance, Beetcher ’072 states:

[I]nvalidating a qualification verification trigger, in order that the format of a target code may support the compile course of the conventional type known by the customer] - - or it may become suitable to add the barrier to ‘patching’ of a target code which is changed. One of such the additional barriers is defining a qualification verification trigger, as other functions of a certain are performed simultaneously. In this case, it is important that the alternate function carried out by the qualification verification trigger cannot carry out with other simple commands. This alternate function must be selected so that any compiled software modules may include some commands which perform that function quite reliably. When having coincided in these criteria, the compiler can generate automatically the target code which performs the alternate function (it is also a qualification verification trigger simultaneously with it) as a part of the usual compilation order. This definition should bring about the important barrier to ‘patching’ of a target code which invalidates a qualification verification trigger.²²⁸

And as described with respect to element 12.3, Beetcher ’072 teaches encoding the triggering instructions into the software code that is decoded via the first license key.²²⁹

²²⁵ Ex. 2, Prosecution History at 519.

²²⁶ Silva Declaration at ¶ 163.

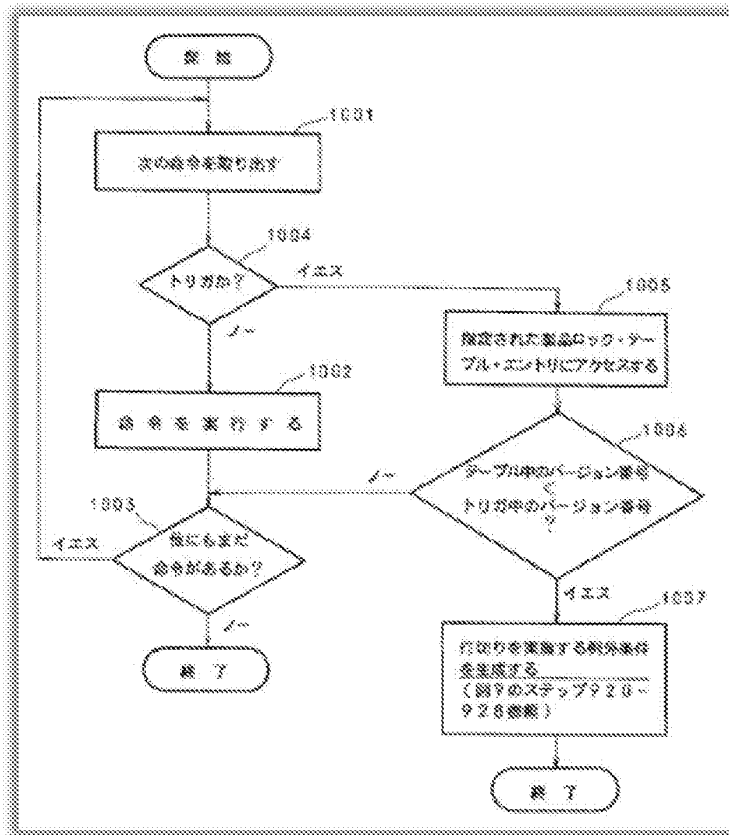
²²⁷ Beetcher ’072 at ¶¶ 0029, 0044; *see also id.* at Abstract, ¶¶ 0009, 0021, 0030, claim 3 (*See* Ex. 5 for English translation).

²²⁸ Beetcher ’072 at ¶ 0044 (*See* Ex. 5 for English translation).

²²⁹ Silva Declaration at ¶ 166.

Beetcher '072's Figure 10, as provided below, illustrates providing the software's underlying functionality based on the first license key (trigger information).²³⁰ For instance, Beetcher '072 explains:

Execution of the software module by the system 101 is made by what this is taken out and performed for (Step 1002) (Step 1001) until a modular target code command is completed (step 1003). When a command is the qualification verification trigger 301 (Step 1004), the lock checking feature 422 is called. At Step 1005, the lock checking feature 422 accesses the product locking table entry 601 to which it corresponds to the product number included in a qualification verification trigger. The qualification for the version number in the product locking table 460 being equal to the version number 303 contained in the qualification verification trigger 301, or performing software, in being larger than it is given (Step 1006).²³¹



²³⁰ *Id.* at ¶¶ 167-69.

²³¹ Beetcher '072 at ¶ 0043 (*See Ex. 5* for English translation).

Accordingly, Beetcher '072 discloses claim 12.

3. Beetcher '072 Anticipates Independent Claim 13.

a) *Preamble: “A method for encoding software code using a computer having a processor and memory, comprising”*

Under the broadest reasonable construction, the preamble is non-limiting. Nevertheless, Beetcher '072 discloses claim 13's preamble. Claim 13's preamble is the same as claim 12's preamble. As explained above, Beetcher '072 discloses a method for encoding software using a computer with a processor and memory. As such, Beetcher '072 teaches this preamble.²³²

b) *Element 13.1: “storing a software code in said memory”*

Element 13.1 is identical to element 12.1. As explained above, Beetcher '072 discloses each limitation of element 12.1. For the same reasons, Beetcher '072 teaches element 13.1.²³³

c) *Element 13.2: “wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system”*

Element 13.2 is identical to element 12.2. As explained above, Beetcher '072 discloses each limitation of element 12.2. For the same reasons, Beetcher '072 teaches element 13.2.²³⁴

d) *Element 13.3: “modifying, by said computer, using a first license key and an encoding algorithm, said software code, to form a modified software code; and wherein said modifying comprises encoding said first code resource to form an encoded first code resource”*

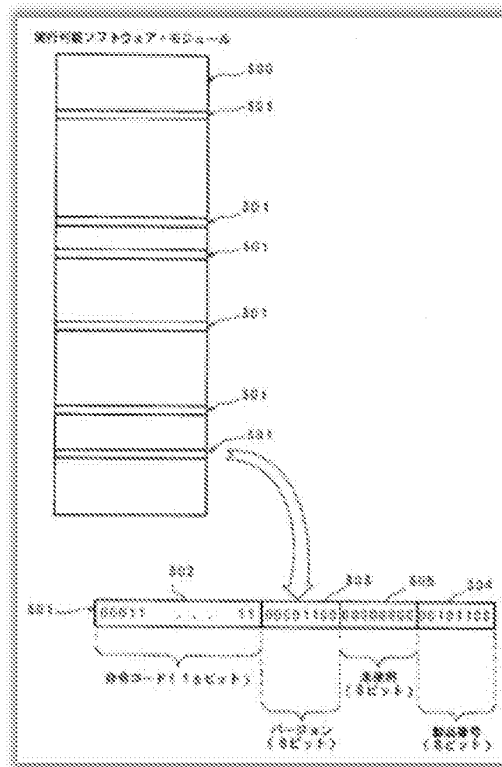
Beetcher '072 discloses element 13.3. As described with respect to element 12.3, Beetcher '072's distributor system includes a computer that encodes software code using a first

²³² Silva Declaration at ¶ 172.

²³³ *Id.* at ¶ 174.

²³⁴ *Id.* at ¶ 176.

license key (e.g., triggering information) and an encoding algorithm (e.g., Figure 7). And Beetcher '072's encoding process modifies the software code by inserting triggering information into the code.²³⁵ For instance, Beetcher '072 details that its system inputs compiled software code into a translator which modifies the code by "automatically ... generat[ing] most number of qualification verification triggers" and "insert[ing] this in the random position in a target code," as shown in Figure 7's steps 703 and 704.²³⁶ Figure 3 illustrates this modifying by inserting triggering information 301 to form a modified software code:



²³⁵ Beetcher '072 at ¶¶ 0034, 0038; *see also id.* at ¶ 0024, Fig. 7 (*See Ex. 5 for English translation*); Silva Declaration at ¶ 178.

²³⁶ Beetcher '072 at ¶ 0038 (*See Ex. 5 for English translation*).

As described with respect to element 12.2, Beetcher '072's software code includes a series of code resources corresponding to sub-objects. And Beetcher '072 teaches a code resource is modified to encode the first code resource via the triggering information.²³⁷ For instance, Beetcher '072 teaches:

[I]nvalidating a qualification verification trigger, in order that the format of a target code may support the compile course of the conventional type known by the customer] - - or it may become suitable to add the barrier to 'patching' of a target code which is changed. One of such the additional barriers is defining a qualification verification trigger, as other functions of a certain are performed simultaneously. In this case, it is important that the alternate function carried out by the qualification verification trigger cannot carry out with other simple commands. This alternate function must be selected so that any compiled software modules may include some commands which perform that function quite reliably. When having coincided in these criteria, the compiler can generate automatically the target code which performs the alternate function (it is also a qualification verification trigger simultaneously with it) as a part of the usual compilation order. This definition should bring about the important barrier to 'patching' of a target code which invalidates a qualification verification trigger.²³⁸

A POSITA would have understood that such modification results in an encoded first code resource.²³⁹

Moreover, during the original prosecution, Patent Owner specified that "[e]ncoding using a key and an algorithm is known."²⁴⁰ As such, a POSITA would have understood that Beetcher '072's encoding technique necessarily includes a first license key and an encoding algorithm to form a modified encoded first code resource.²⁴¹

²³⁷ Beetcher '072 at ¶¶ 0021, 0044; *see also id.* at Abstract, ¶ 0009 (*See Ex. 5 for English translation*).

²³⁸ Beetcher '072 at ¶ 0044 (*See Ex. 5 for English translation*).

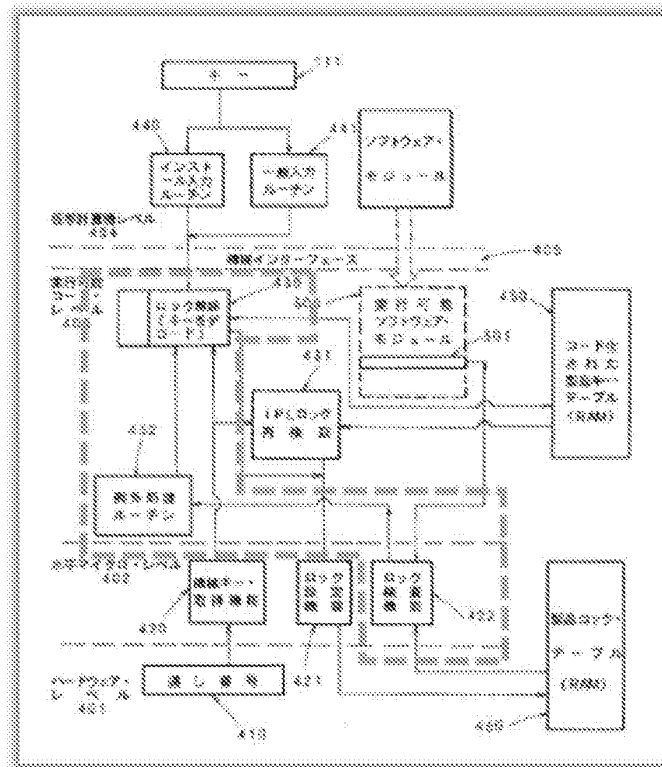
²³⁹ Silva Declaration at ¶¶ 178-79.

²⁴⁰ Ex. 2, Prosecution History at 519.

²⁴¹ Silva Declaration at ¶ 180.

- e) ***Element 13.4: “wherein said modified software code comprises said encoded first code resource, and a decode resource for decoding said encoded first code resource”***

Beetcher '072 discloses element 13.4. Beetcher '072 explains that its modified software code includes a decode resource for decoding the encoded first code resource. Specifically, Beetcher '072 teaches that executing a trigger 301 invokes check lock function 422, which results in accessing “unlock (decode key)” function 430 upon confirmation that the customer possesses the software’s license key.²⁴² Beetcher '072’s Figure 4, as annotated below, illustrates the decode resource (dashed perimeter) of the modified software code:²⁴³



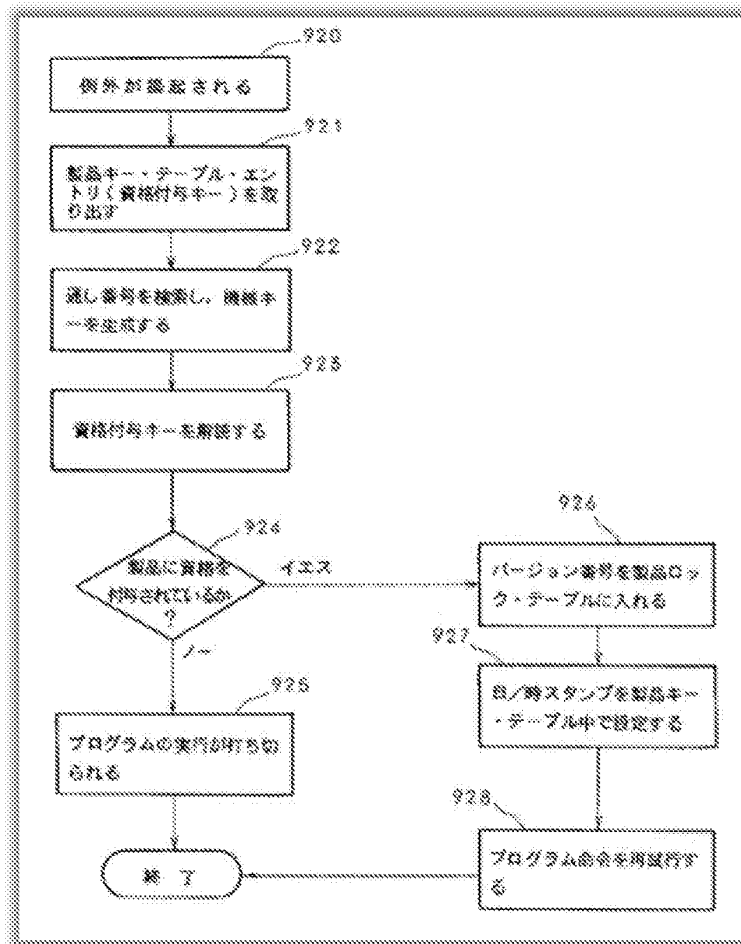
²⁴² Beetcher '072 at ¶¶ 0041, 0043, Figs. 9b, 10; see also *id.* at ¶¶ 0031-32, 0034, 0040 (See Ex. 5 for English translation).

²⁴³ Silva Declaration at ¶ 183.

f) ***Element 13.5: “wherein said decode resource is configured to decode said encoded first code resource upon receipt of said first license key”***

Beetcher '072 discloses element 13.5. Beetcher '072 specifies that its decode resource decodes the encoded first code resource upon receipt of the license key. Beetcher '072, for example, states that “the qualification grant key enciphered from the suitable entry in the product key table 450 in which the lock release routine 430 was coded ... is taken out ... and a qualification grant key is decoded Subsequently, at Step 928, a qualification verification trigger is retried and execution of a program is continued.”²⁴⁴ And Beetcher '072's Figure 9b illustrates accessing the decode resource to decode the encoded first code resources based on the entitlement key, reflected in steps 921 to 928:

²⁴⁴ Beetcher '072 at ¶ 0041 (*See Ex. 5 for English translation*).



As such, a POSITA would have understood that Beetcher '072's decode resource is configured to decode the encoded first code resource based on first license key.²⁴⁵

Accordingly, Beetcher '072 discloses claim 13.

4. Beetcher '072 Anticipates Independent Claim 14.

- a) **Preamble:** “A method for encoding software code using a computer having a processor and memory, comprising”

Under the broadest reasonable construction, the preamble is non-limiting. Nevertheless, Beetcher '072 discloses claim 14's preamble. Claim 14's preamble is the same as each of claim

²⁴⁵ Silva Declaration at ¶ 186.

12 and 13's preamble. As explained above, Beetcher '072 discloses a method for encoding software using a computer with a processor and memory. As such, Beetcher '072 teaches this preamble.²⁴⁶

b) ***Element 14.1: "storing a software code in said memory"***

Element 14.1 is identical to element 12.1. As explained above, Beetcher '072 discloses each limitation of element 12.1. For the same reasons, Beetcher '072 teaches element 14.1.²⁴⁷

c) ***Element 14.2: "wherein said software code defines software code interrelationships between code resources that result in a specified underlying functionality when installed on a computer system"***

Beetcher '072 discloses element 14.2. Beetcher '072 details that its software code is compiled into executable code by compiler 126. This compiler works with translator 127 to compile the software sub-objects and insert triggering information.²⁴⁸ And Beetcher '072 specifies that translator 127 generates the verification triggers and randomly inserts the triggers into the target code.²⁴⁹ Translator 127 then resolves references to the positions of the triggers in the target code, which corresponds to defining code interrelationships between code resources.²⁵⁰ As shown in steps 701 and 702 of Figure 7, Beetcher '072 teaches its software code is input into compiler 126 that produces a template of the software code.²⁵¹

²⁴⁶ *Id.* at ¶ 190.

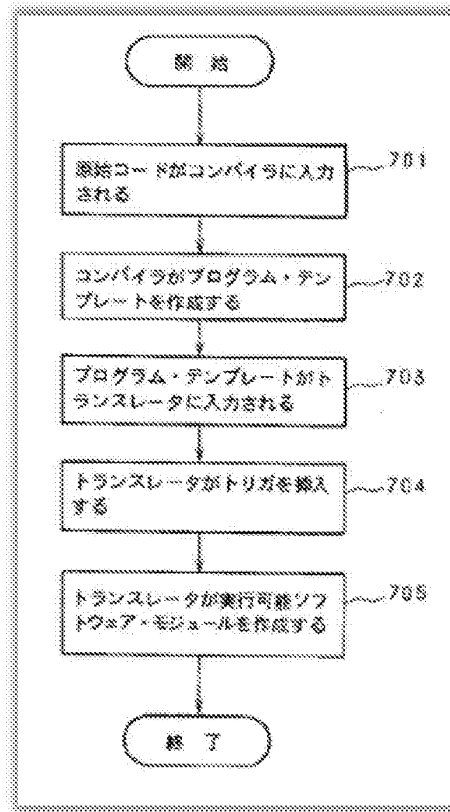
²⁴⁷ *Id.* at ¶ 192.

²⁴⁸ Beetcher '072 at ¶ 0034 (*See* Ex. 5 for English translation).

²⁴⁹ Beetcher '072 at ¶ 0038 (*See* Ex. 5 for English translation).

²⁵⁰ Beetcher '072 at ¶ 0038 (*See* Ex. 5 for English translation); Silva Declaration at ¶ 194.

²⁵¹ Beetcher '072 ¶¶ 0034, 0038, Fig. 7; *see also id.* at ¶¶ 0024, 0029, 0033 (*See* Ex. 5 for English translation).



A POSITA would have understood that this software code template also defines the code interrelationships between the code resources.²⁵² As Patent Owner specified during the original prosecution, software code interrelationships are defined during the compiling process of conventional software applications:

What the examiner has implied by alleging that the "specification ... fails to teach or mention 'software code interrelationships'" is that software code interrelationships were somehow unknown in the art, which clearly is not the case. As admitted, in the specification at the beginning of paragraph [0051], an "application" comprises "sub-objects" whose "order in the computer memory is of vital importance" in order to perform an intended function. And as admitted further in paragraph [0051], **"When a program is compiled, then, it consists of a collection of these sub-objects, whose exact order or arrangement in memory is not important, so long as any sub-object which uses another sub-object knows where in memory it can be found."** Paragraph [0051] of course refers

²⁵² Silva Declaration at ¶¶ 194-95.

to conventional applications. Accordingly, that is admittedly a discussion of what is already know by one skilled in the art. Accordingly, the examiner's statement that the specification lacks written description support for "software code interrelationships" is inconsistent with the fact that such **interrelationships were explained in paragraphs [0051] and [0052] as a fundamental basis of pre-existing modem computer programs.**²⁵³

Moreover, during the original prosecution, Patent Owner specified that

“interrelationships between code resource are not that which is novel.”²⁵⁴ Based on the Patent Owner’s concessions, it is clear that a POSITA would have understood that Beetcher ’072’s code necessarily defines code interrelationships between code resources.²⁵⁵

Beetcher ’072 further teaches that the code resource interrelationships specify the underlying application functionalities when installed on the customer’s computer 101. For instance, Beetcher ’072’s software code includes multiple entitlement verification triggers.²⁵⁶

And Beetcher ’072 details that certain code resources include triggering instructions that control the underlying functionalities of the software code:

[An] additional barrier[] is defining a qualification verification trigger, as other functions of a certain are performed simultaneously.... This alternate function must be selected so that any compiled software modules may include some commands which perform that function quite reliably. When having coincided in these criteria, the compiler can generate automatically the target code which performs the alternate function (it is also a qualification verification trigger simultaneously with it) as a part of the usual compilation order. This definition should bring about the important barrier to ‘patching’ of a target code which invalidates a qualification verification trigger.²⁵⁷

Beetcher ’072 further explains that “a qualification verification trigger is also the direct instruction ... which performs other useful work of a certain.... [I]f a trigger command is

²⁵³ Ex. 2, Prosecution History at 519.

²⁵⁴ *Id.* at 519.

²⁵⁵ Silva Declaration at ¶ 196.

²⁵⁶ Beetcher ’072 at ¶¶ 0021, 0038, 0041, Fig. 3; *see also id.* at ¶¶ 0029, 0034, 0043-44 (*See* Ex. 5 for English translation).

²⁵⁷ Beetcher ’072 at ¶ 0044; *see also id.* at ¶¶ 0021, 0029 (*See* Ex. 5 for English translation).

executed, the system 101 will perform other operations of a certain simultaneously with qualification verification.”²⁵⁸ As such, a POSITA would have understood that the code interrelationships between Beetcher ’072’s code resources result in a specified underlying functionality once installed.²⁵⁹

d) *Element 14.3: “encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key encoded software code”*

Element 14.3 is identical to element 12.3. As explained above, Beetcher ’072 discloses each limitation of element 12.3. For the same reasons, Beetcher ’072 teaches element 14.3.

Moreover, during the original prosecution, Patent Owner specified that “[e]ncoding using a key and an algorithm is known” and that “an interrelationship in software code is necessarily defined by digital data, and digital data can obviously be encoded by an encoding process.”²⁶⁰ As such, a POSITA would have understood that Beetcher ’072’s encoding technique necessarily includes a first license key and an encoding algorithm to form a first license key encoded software code.

e) *Element 14.4: “in which at least one of said software code interrelationships are encoded”*

Beetcher ’072 discloses element 14.4. As described with respect to element 14.2, Beetcher ’072 teaches that its software code defines code interrelationships between code resources and triggering information 301 in the code control certain underlying software functionality. And Beetcher ’072 details that triggering information 301 is encoded into the

²⁵⁸ Beetcher ’072 at ¶ 0029 (Beetcher ’072 specifies that these functions are those “which does not need to divide, does not need to be ordering the operand for the processing and does not need to be specified”) (*See* Ex. 5 for English translation).

²⁵⁹ Silva Declaration at ¶¶ 197-98.

²⁶⁰ Ex. 2, Prosecution History at 519.

software code.²⁶¹ For instance, Beetcher '072 explains that the triggering instructions will be encoded into the code resources controlling software functionality:

[An] additional barrier[] is defining a qualification verification trigger, as other functions of a certain are performed simultaneously.... This alternate function must be selected so that any compiled software modules may include some commands which perform that function quite reliably. When having coincided in these criteria, the compiler can generate automatically the target code which performs the alternate function (it is also a qualification verification trigger simultaneously with it) as a part of the usual compilation order. This definition should bring about the important barrier to 'patching' of a target code which invalidates a qualification verification trigger.²⁶²

And Beetcher '072 details that "a qualification verification trigger is also the direct instruction ... which performs other useful work of a certain.... [I]f a trigger command is executed, the system 101 will perform other operations of a certain simultaneously with qualification verification."²⁶³

Accordingly, a POSITA would have understood that this encoded triggering information includes encoded code interrelationship of the coder resources.²⁶⁴

Thus, Beetcher '072 discloses claim 14.

C. SNQ-3: Claims 11, 12, 13, and 14 are Anticipated by Cooperman Under 35 U.S.C. § 102(a).

Cooperman anticipates claims 11, 12, 13, and 14 under 35 U.S.C. § 102(a).

²⁶¹ Beetcher '072 at ¶¶ 0021, 0029, 0044 (*See* Ex. 5 for English translation).

²⁶² Beetcher '072 at ¶ 0044; *see also id.* at ¶¶ 0021, 0029 (*See* Ex. 5 for English translation).

²⁶³ Beetcher '072 at ¶ 0029 (Beetcher '072 specifies that these functions are those "which does not need to divide, does not need to be ordering the operand for the processing and does not need to be specified") (*See* Ex. 5 for English translation).

²⁶⁴ Silva Declaration at ¶¶ 201-02.

1. Cooperman Anticipates Independent Claim 11.

a) Preamble: “A method for licensed software use, the method comprising”

Under the broadest reasonable construction, the preamble is non-limiting. Nevertheless, Cooperman discloses claim 11’s preamble. Specifically, Cooperman describes a method for use of licensed software.²⁶⁵ Cooperman, for instance, provides a method of encoding a license key into software code where the code operates by “ask[ing] the user for personalization information, which include the license code.”²⁶⁶ And Cooperman specifies that, to extract a digital watermark essential to operate the software, “the user must have a key. The key, in turn, is a function of the license information for the copy of the software in question.”²⁶⁷

As such, Cooperman teaches this preamble.²⁶⁸

b) Element 11.1: “loading a software product on a computer, said computer comprising a processor, memory, an input, and an output, so that said computer is programmed to execute said software product”

Cooperman discloses element 11.1. Specifically, Cooperman’s system includes a computer having a processor, memory, input, and output. Cooperman initially recognizes that “[a] computer application seeks to provide a user with certain utilities or tools, that is, users interact with a computer or similar device to accomplish various tasks and applications provide the relevant interface.”²⁶⁹ And Cooperman discloses loading software object code into “computer memory for the purpose of execution.”²⁷⁰ Cooperman further discusses that software products

²⁶⁵ Cooperman at 5:35-6:5, 11:24-33; *see also id.* at 3:24-31, 11:34-37, 12:13-35, claim 2.

²⁶⁶ *Id.* at 11:24-33.

²⁶⁷ *Id.* at 12:13-16.

²⁶⁸ Silva Declaration at ¶ 208-10.

²⁶⁹ Cooperman at 3:16-20.

²⁷⁰ *Id.* at claim 5; *see also id.* at 13:31-36, claim 7.

include functions made from executable object code whose “order in the computer memory is of vital importance.”²⁷¹ Accordingly, a POSITA would have understood that Cooperman’s computer includes a processor and memory for executing the stored software code because, as expert Dr. Silva explains, inclusion of a processor and memory is standard in such computers.²⁷²

Cooperman explains that the computer may “process[] a digital sample stream for the purpose of modifying it or playing the digital sample stream.”²⁷³ A POSITA would have understood that such digital sample stream processing is performed by a computer’s processor and an output plays the digital sample stream.²⁷⁴

Cooperman further describes loading a software product on the computer, so the computer can execute the software product. For instance, Cooperman further describes the operation of the disclosed software product requires:

1. Installing, i.e., loading, the software on the computer;
2. Asking the user to input a license code;
3. Generating, i.e., outputting, a decoding key after receiving the license code to access the software resources.²⁷⁵

c) ***Element 11.2: “said software product outputting a prompt for input of license information”***

Cooperman discloses element 11.2. Specifically, Cooperman explains that its software product requests that the user input license information, i.e., a license key, into the computer before the product can be used.²⁷⁶ For instance, Cooperman explains that the software product

²⁷¹ *Id.* at 7:1-5.

²⁷² Silva Declaration at ¶ 212.

²⁷³ Cooperman at claim 4; *see also id.* at claims 5, 6 (processing digital sample stream and a map list).

²⁷⁴ Silva Declaration at ¶ 213.

²⁷⁵ Cooperman at 11:24-34; Silva Declaration at ¶ 214.

²⁷⁶ Cooperman 11:24-33; *see also id.* at Abstract, 3:24-28, 5:35-6:5, 11:6-8, 12:10-16.

prompts the user to input license information: “1) when it is run for the first time, after installation, it asks the user for personalization information, which includes the license code. This can include a particular computer configuration.”²⁷⁷ Cooperman specifies that such license codes are entered by the user “when prompted at start-up.”²⁷⁸ A POSITA would have understood this request corresponds to the software product outputting a prompt to input license information.²⁷⁹

During the original prosecution, Patent Owner confirmed that such a teaching disclosed by Cooperman meets element 11.2. For instance, Patent Owner’s May 14, 2012 Appeal Brief states that element 11.2 is taught by: “1) when it is run for the first time, after installation, it asks the user for personalization information, which includes the license code. This can include a particular computer configuration.”²⁸⁰ Cooperman includes this same teaching, and thus discloses element 11.2²⁸¹.

d) *Element 11.3: “said software product using license information entered via said input in response to said prompt in a routine designed to decode a first license code encoded in said software product”*

Cooperman discloses element 11.3. Specifically, Cooperman explains that its system includes a routine designed to decode a first license code encoded in the software product based on inputted license information. For instance, Cooperman states:

Given that there are one or more of these essential resources, what is needed to realize the present invention is the presence of certain data resources of a type which

²⁷⁷ *Id.* at 11:25-28.

²⁷⁸ *Id.* at 1:25-28.

²⁷⁹ Silva Declaration at ¶ 216.

²⁸⁰ Ex. 2, Prosecution History at 577 (original claim 58 issued as claim 11).

²⁸¹ Silva Declaration at ¶ 217.

are amenable to the "stega-cipher" process described in the "Steganographic Method and Device" patent application.²⁸²

And Cooperman discloses: "3) Once it has the license code, it can then generate the proper decoding key to access the essential code resources."²⁸³ As explained regarding element 11.2, Cooperman details that the user enters license information via an input in response to the prompt.

During the original prosecution, Patent Owner confirmed that such a teaching disclosed by Cooperman meets element 11.3. For instance, Patent Owner's May 14, 2012 Appeal Brief states that element 11.3 is taught by:

Given that there are one or more of these essential resources, what is needed to realize the present invention is the presence of certain data resources of a type which are amenable to the "stega-cipher" process described in the "Steganographic Method and Device" patent U.S. Pat. No. 5,613,004 [issued from U.S. Application No. 08/489,172].

* * * *

3) Once it has the license code, it can then generate the proper decoding key to access the essential code resources.²⁸⁴

Cooperman includes these same teachings, and thus discloses element 11.3.²⁸⁵

Accordingly, Cooperman discloses claim 11.

²⁸² Cooperman at 9:22-27; *see also id.* at 2:34-37, 4:7-17 (incorporating by reference U.S. Patent Application No. 08/489,172 entitled "Steganographic Method and Device").

²⁸³ Cooperman at 11:31-33.

²⁸⁴ Ex. 2, Prosecution History at 577 (original claim 58 issued as claim 11); *see also id.* at 664 (Patent Owner explaining that element 11.3 is met by teachings corresponding to Cooperman at 10:7-11:33).

²⁸⁵ Silva Declaration at ¶¶ 220-23.

2. Cooperman Anticipates Independent Claim 12.

- a) ***Preamble: “A method for encoding software code using a computer having a processor and memory, the method comprising”***

Under the broadest reasonable construction, the preamble is non-limiting.²⁸⁶

Nevertheless, Cooperman discloses claim 12’s preamble. Specifically, Cooperman describes a method for encoding software code using a computer with a processor and memory. Cooperman details that, during the software code assembly, the computer system will “choose one or several essential code resources, and encode them into one or several data resources using the stegacipher process.”²⁸⁷ As expert Dr. Silva explains, Cooperman’s computer would necessarily include a processor and memory in order to function.²⁸⁸

As such, Cooperman teaches this preamble.²⁸⁹

- b) ***Element 12.1: “storing a software code in said memory”***

Cooperman discloses element 12.1. Specifically, Cooperman describes techniques for randomizing the location of software code stored in memory.²⁹⁰ Cooperman explains that this randomization makes the software code more resistant to patching and memory capture analysis.²⁹¹ As such, a POSITA would have understood that these techniques are used for code

²⁸⁶ Claim 12’s preamble recites “a computer” and claim 12’s body recites “a computer system.” It is unclear whether those elements refer to the same or separate computing devices. For purposes of this Request and using the broadest reasonable interpretation consistent with the specification, it is assumed that the “computer” recited in the preamble is a device separate from the “computer system.”

²⁸⁷ Cooperman at 10:13-16; *see also id.* at claim 6.

²⁸⁸ Silva Declaration at ¶ 227.

²⁸⁹ *Id.* at ¶¶ 226-28.

²⁹⁰ Cooperman at 3:32-37; *see also id.* at 4:1-6, 6:5-9, 13:23-46, 14:4-9.

²⁹¹ *Id.* at 3:13-16, 14:37-15:18, claim 7.

stored in memory because, as expert Dr. Silva explains, storage of code in memory is standard in computers like Cooperman's.²⁹²

Cooperman further explains that its software code is compiled and assembled: "When code and data resources are compiled and assembled into a precursor of an executable program the next step is to use a utility application for final assembly of the executable application."²⁹³

Cooperman also states that code resources are stored separately from applications, i.e., software, code.²⁹⁴ A POSITA would have understood that Cooperman's compiled and assembled application code is stored in memory. As Dr. Silva explains, Cooperman's computer would necessarily include store software code in memory in order to function.²⁹⁵

During the original prosecution, Patent Owner confirmed that such a teaching disclosed by Cooperman meets element 12.1. For example, Patent Owner's May 14, 2012 Appeal Brief states that element 12.1 is taught by: "When code and data resources are compiled and assembled into a precursor of an executable program the next step is to use a utility application for final assembly of the executable application."²⁹⁶

Cooperman includes this same teaching, and thus discloses element 12.1.

²⁹² Silva Declaration at ¶ 230.

²⁹³ Cooperman at 10:8-11; *see also id.* at 7:1-21.

²⁹⁴ *Id.* at 7:26-30.

²⁹⁵ Silva Declaration at ¶ 231.

²⁹⁶ Ex. 2, Prosecution History at 578 (original claim 59 issued as claim 12); *see also id.* at 415-16 (original claim 61, which issued as claim 13, includes the same limitation "wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system"); Silva Declaration at ¶ 232.

c) ***Element 12.2: “wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system”***

Cooperman discloses element 12.2. Specifically, Cooperman explains that its software code includes multiple code resources that include a first code resource.²⁹⁷ And Cooperman discloses that its software code includes the code resources and provides an underlying functionality when installed on the computer.²⁹⁸ For instance, Cooperman states: “The basic premise for this scheme is that there are a certain sub-set of executable code resources, that comprise an application and that are ‘essential’ to the proper function of the application.”²⁹⁹

As another example, Cooperman details that software applications include code resources providing functionalities specified in the application:

The memory address of the first instruction in one of these sub-objects is called the "entry point" of the function or procedure. The rest of the instructions comprising that sub-object immediately follow from the entry point. Some systems may prefix information to the entry point which describes calling and return conventions for the code which follows, an example is the Apple Macintosh Operating System (MacOS). These sub-objects can be packaged into what are referred to in certain systems as "code resources," which may be stored separately from the application, or shared with other applications, although not necessarily. Within an application there are also data objects, which consist of some data to be operated on by the executable code. These data objects are not executable. That is, they do not consist of executable instructions. The data objects can be referred to in certain systems as "resources."³⁰⁰

²⁹⁷ Cooperman at 10:11-29, 11:13-33; *see also id.* at Abstract, 7:26-30, 9:10-21, 13:31-36, claim 6.

²⁹⁸ *Id.* at 7:19-36, 11:24-37; *see also id.* at 8:30-33, 10:11-29.

²⁹⁹ *Id.* at 8:30-33.

³⁰⁰ *Id.* at 7:19-36.

The '842 Patent refers to sub-objects and a memory scheduler as examples of code resources.³⁰¹

In this additional and alternative way, a POSITA would have understood that Cooperman's sub-objects and code resources.³⁰²

During the original prosecution, Patent Owner confirmed that such teachings disclosed by Cooperman meets element 12.2. For example, Patent Owner's May 14, 2012 Appeal Brief states that element 12.2 is taught by: "The basic premise for this scheme is that there are a certain subset of executable code resources, that comprise an application and that are 'essential' to the proper function of the application."³⁰³ As another example, Patent Owner's May 14, 2012 Appeal Brief states this element is taught by:

The memory address of the first instruction in one of these sub-objects is called the "entry point" of the function or procedure. The rest of the instructions comprising that sub-object immediately follow from the entry point. Some systems may prefix information to the entry point which describes calling and return conventions for the code which follows, an example is the Apple Macintosh Operating System (MacOS). These sub-objects can be packaged into what are referred to in certain systems as "*code resources*," which may be stored separately from the application, or shared with other applications, although not necessarily. Within an application there are also data objects, which consist of some data to be operated on by the executable code. These data objects are not executable. That is, they do not consist of executable instructions. The data objects can be referred to in certain systems as "resources."³⁰⁴

Cooperman includes these same teachings, and thus discloses element 12.2.³⁰⁵

³⁰¹ '842 Patent at 11:55-65, 15:36-42.

³⁰² Silva Declaration at ¶¶ 235-36.

³⁰³ Ex. 2, Prosecution History at 578 (original claim 59 issued as claim 12).

³⁰⁴ *Id.* at 579-80 (original claim 61, which issued as claim 13, includes the same limitation "wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system").

³⁰⁵ Silva Declaration at ¶¶ 235-37.

d) *Element 12.3: “encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key encoded software code”*

Cooperman discloses element 12.3. Specifically, Cooperman describes encoding its software code to form a first license key encode software code.³⁰⁶ Cooperman details that this encoding uses a first license key and an encoding algorithm.³⁰⁷ For instance, Cooperman details that “[t]he assembly utility can be supplied with a key generated from a license code generated for the license in question.”³⁰⁸ And Cooperman states: “The utility will choose one or several essential code resources, and encode them into one or several data resources using the stegacipher process.”³⁰⁹

During the original prosecution, Patent Owner confirmed that such a teaching disclosed by Cooperman meets element 12.3. For instance, Patent Owner’s May 14, 2012 Appeal Brief states that “encoding, by said computer using at least a first license key and an encoding algorithm, said software code” is taught by:

The assembly utility can be supplied with a key generated from a license code generated *for the license in question*.

* * * *

The utility will choose one or several essential code resources, and encode them into one or several data resources using the stegacipher process.³¹⁰

As another example, Patent Owner’s May 14, 2012 Appeal Brief states that “to form a first license key encoded software code” is taught by:

³⁰⁶ Cooperman at 10:28-35, 11:6-15; *see also id.* at 2:27-31, 3:24-31, 12:13-23, claim 6.

³⁰⁷ *Id.* at 10:13-16, 11:9-11, claim 6.

³⁰⁸ *Id.* at 11:9-11.

³⁰⁹ *Id.* at 10:13-16; *see also id.* at claim 6.

³¹⁰ Ex. 2, Prosecution History at 578 (emphasis in original) (original claim 59 issued as claim 12).

The purpose of this scheme is to make a particular licensed copy of an application distinguishable from any other. It is not necessary to distinguish every instance of an application, merely every instance of a license.

* * * *

3) Once it has the license code, it can then generate the proper decoding key to access the essential code resources.³¹¹

Cooperman includes this same teaching, and thus discloses element 12.3.

Moreover, during the original prosecution, Patent Owner specified that “[e]ncoding using a key and an algorithm is known.”³¹² As such, a POSITA would have understood that Cooperman’s encoding technique necessarily includes a first license key and an encoding algorithm to form a first license key encoded software code.³¹³

- e) ***Element 12.4: “wherein, when installed on a computer system, said first license key encoded software code will provide said specified underlying functionality only after receipt of said first license key”***

Cooperman discloses element 12.4. Specifically, Cooperman explains that its first license key encoded software code provides the specified underlying functionality only after receipt of the first license key.³¹⁴ For instance, Cooperman states: “Once it has the license code, it can then generate the proper decoding key to access the essential code resources. Note that the application...must contain the license code issued to the licensed owner, to access its essential code resources.”³¹⁵ Cooperman describes that these essential code resources correspond to the underlying functionalities of the software program installed on the computer.³¹⁶

³¹¹ Ex. 2, Prosecution History at 578-79 (original claim 59 issued as claim 12).

³¹² *Id.* at 519.

³¹³ Silva Declaration at ¶¶ 240-43.

³¹⁴ Cooperman at 10:28-35, 11:6-15; *see also id.* at 2:27-31, 3:24-31, 12:13-23, claim 6.

³¹⁵ *Id.* at 11:31-37.

³¹⁶ *Id.* at 5:35-6:9, 11:6-8, 11:31-37, 12:10-16; *see also id.* at 6:26-30, 7:1-5, 8:25-37, 9:14-21; Silva Declaration at ¶¶ 246-47.

Accordingly, Cooperman discloses claim 12.

3. Cooperman Anticipates Independent Claim 13.

a) *Preamble: “A method for encoding software code using a computer having a processor and memory, comprising”*

Under the broadest reasonable construction, the preamble is non-limiting. Nevertheless, Cooperman discloses claim 13’s preamble. Claim 13’s preamble is the same as claim 12’s preamble. As explained above, Cooperman discloses a method for encoding software using a computer with a processor and memory. As such, Cooperman teaches this preamble.³¹⁷

b) *Element 13.1: “storing a software code in said memory”*

Element 13.1 is identical to element 12.1. As explained above, Cooperman discloses each limitation of element 12.1. For the same reasons, Cooperman teaches element 13.1.

And during the original prosecution, Patent Owner confirmed that such a teaching disclosed by Cooperman meets element 13.1. For instance, Patent Owner’s May 14, 2012 Appeal Brief states that element 13.1 is taught by: “When code and data resources are compiled and assembled into a precursor of an executable program the next step is to use a utility application for final assembly of the executable application.”³¹⁸ As explained with respect to element 12.1, Cooperman includes this same teaching.³¹⁹

c) *Element 13.2: “wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system”*

Element 13.2 is identical to element 12.2. As explained above, Cooperman discloses each limitation of element 12.2. For the same reasons, Cooperman teaches element 13.2.³²⁰

³¹⁷ Silva Declaration at ¶¶ 249-50.

³¹⁸ Ex. 2, Prosecution History at 579 (original claim 61 issued as claim 13).

³¹⁹ Silva Declaration at ¶ 252.

³²⁰ *Id.* at ¶¶ 254-55.

And during the original prosecution, Patent Owner confirmed that such a teaching disclosed by Cooperman meets element 13.2. For instance, Patent Owner's May 14, 2012 Appeal Brief states that element 13.2 is taught by:

The memory address of the first instruction in one of these sub-objects is called the "entry point" of the function or procedure. The rest of the instructions comprising that sub-object immediately follow from the entry point. Some systems may prefix information to the entry point which describes calling and return conventions for the code which follows, an example is the Apple Macintosh Operating System (MacOS). These sub-objects can be packaged into what are referred to in certain systems as "*code resources*," which may be stored separately from the application, or shared with other applications, although not necessarily. Within an application there are also data objects, which consist of some data to be operated on by the executable code. These data objects are not executable. That is, they do not consist of executable instructions. The data objects can be referred to in certain systems as "resources."³²¹

As explained with respect to element 12.2, Cooperman includes this same teaching.³²²

- d) ***Element 13.3: “modifying, by said computer, using a first license key and an encoding algorithm, said software code, to form a modified software code; and wherein said modifying comprises encoding said first code resource to form an encoded first code resource”***

Cooperman discloses element 13.3. Specifically, Cooperman describes modifying its software code using a license key and an encoding algorithm.³²³ And Cooperman's modification includes encoding the first code resource to form an encoded first code resource. For instance, Cooperman teaches code modification using a “digital watermarking” process to encode a code resource: “The first method of the present invention described involves hiding necessary ‘parts’ or code ‘resources’ in digitized sample resources using a ‘digital watermarking’ process, such as

³²¹ Ex. 2, Prosecution History at 579-80 (original claim 61 issued as claim 13).

³²² Silva Declaration at ¶¶ 254-55.

³²³ Cooperman at 3:10-31, 8:25-30, 10:8-31; *see also id.* at 2:19-37, 4:7-17, 11:6-24, claim 6.

that described in the ‘Steganographic Method and Device’ patent application.”³²⁴ Cooperman further discloses “watermarking with ‘keys’ derived from license codes... and using the watermarks encoded with such keys to hide an essential subset for the application code resources.”³²⁵ A POSITA would have understood that such modification results in a modified software code.³²⁶

During the original prosecution, Patent Owner confirmed that such a teaching disclosed by Cooperman meets element 13.3. For instance, Patent Owner’s May 14, 2012 Appeal Brief states that element 13.3 is taught by: “The first method of the present invention described involves hiding necessary ‘parts’ or code ‘resources’ in digitized sample resources using a ‘digital watermarking’ process, such as that described in the ‘Steganographic Method and Device’ patent application.”³²⁷ Cooperman includes this same teaching, and thus discloses element 13.3.³²⁸

Moreover, during the original prosecution, Patent Owner specified that “[e]ncoding using a key and an algorithm is known.”³²⁹ As such, a POSITA would have understood that Cooperman’s encoding technique necessarily includes a first license key and an encoding algorithm to form a modified encoded first code resource.³³⁰

³²⁴ *Id.* at 8:25-30; *see also id.* at 2:34-37, 4:7-17 (incorporating by reference U.S. Patent Application No. 08/489,172 entitled “Steganographic Method and Device”).

³²⁵ Cooperman at 5:15-22.

³²⁶ Silva Declaration at ¶ 257

³²⁷ Ex. 2, Prosecution History at 580 (original claim 61 issued as claim 13).

³²⁸ Silva Declaration at ¶ 258

³²⁹ Ex. 2, Prosecution History at 519.

³³⁰ Silva Declaration at ¶ 259.

e) ***Element 13.4: “wherein said modified software code comprises said encoded first code resource, and a decode resource for decoding said encoded first code resource”***

Cooperman discloses element 13.4. Specifically, Cooperman explains that its modified software code includes a decode resource for decoding the encoded first code resource.³³¹ For instance, Cooperman describes the modified application code has a decoding resource: “Note further that the application contains a code resource which performs the function of decoding an encoded code resource from a data resource.”³³² And Cooperman further discloses that “[o]nce [the application] has the license code, it can then generate the proper decoding key to access the essential code resources.”³³³

During the original prosecution, Patent Owner confirmed that such a teaching disclosed by Cooperman meets element 13.4. For instance, Patent Owner’s May 14, 2012 Appeal Brief states that element 13.4 is taught by: “Note further that the application contains a code resource which performs the function of decoding an encoded code resource from a data resource.”³³⁴ Cooperman includes this same teaching, and thus discloses element 13.4.³³⁵

f) ***Element 13.5: “wherein said decode resource is configured to decode said encoded first code resource upon receipt of said first license key”***

Cooperman discloses element 13.5. Cooperman specifies that its decode resource decodes the encoded first code resource upon receipt of the license key:

The application must also contain a data resource which specifies in which data resource a particular code resource is encoded. This data resource is created and

³³¹ Cooperman at 11:17-20, claim 6; *see also id.* 11:31-33, claim 5.

³³² *Id.* at 11:17-20.

³³³ *Id.* at 11:31-33; Silva Declaration at ¶ 262.

³³⁴ Ex. 2, Prosecution History at 580 (original claim 61 issued as claim 13).

³³⁵ Silva Declaration at ¶ 263.

added at assembly time by the assembly utility. The application can then operate as follows:

- 1) when it is run for the first time, after installation, it asks the user for personalization information, which includes the license code. This can include a particular computer configuration;
- 2) it stores this information in a personalization data resource;
- 3) Once it has the license code, it can then generate the proper decoding key to access the essential code resources.³³⁶

During the original prosecution, Patent Owner confirmed that such a teaching disclosed by Cooperman meets element 13.5. For instance, Patent Owner's May 14, 2012 Appeal Brief states that element 13.5 is taught by:

The application must also contain a data resource which specifies in which data resource a particular code resource is encoded. This data resource is created and added at assembly time by the assembly utility. The application can then operate as follows:

- 1) when it is run for the first time, after installation, it asks the user for personalization information, which includes the license code. This can include a particular computer configuration;
- 2) it stores this information in a personalization data resource;
- 3) Once it has the license code, it can then generate the proper decoding key to access the essential code resources.³³⁷

Cooperman includes this same teaching, and thus discloses element 13.5.³³⁸

Accordingly, Cooperman discloses claim 13.

4. Cooperman Anticipates Independent Claim 14.

- a) ***Preamble: "A method for encoding software code using a computer having a processor and memory, comprising"***

Under the broadest reasonable construction, the preamble is non-limiting. Nevertheless, Cooperman discloses claim 14's preamble. Claim 14's preamble is the same as each of claim 12

³³⁶ Cooperman at 11:20-33; *see also id.* at claims 5 and 6.

³³⁷ Ex. 2, Prosecution History at 580-81 (original claim 61 issued as claim 13).

³³⁸ Silva Declaration at ¶¶ 266-68.

and 13's preamble. As explained above, Cooperman discloses a method for encoding software using a computer with a processor and memory. As such, Cooperman teaches this preamble.³³⁹

b) *Element 14.1: "storing a software code in said memory"*

Element 14.1 is identical to element 12.1. As explained above, Cooperman discloses each limitation of element 12.1. For the same reasons, Cooperman teaches element 14.1.

And during the original prosecution, Patent Owner confirmed that such a teaching disclosed by Cooperman meets element 14.1. And as another example, Patent Owner's May 14, 2012 Appeal Brief states that element 14.1 is taught by: "When code and data resources are compiled and assembled into a precursor of an executable program the next step is to use a utility application for final assembly of the executable application."³⁴⁰ As explained with respect to element 12.1, Cooperman includes this same teaching.³⁴¹

c) *Element 14.2: "wherein said software code defines software code interrelationships between code resources that result in a specified underlying functionality when installed on a computer system"*

Cooperman discloses element 14.2. Specifically, Cooperman explains that its software code establishes software code interrelationships between code resources.³⁴² For instance, Cooperman details that its software code includes a special code resource, such a memory scheduler, that knows the code interrelationships of all other code resources:

Under the present invention, the application contains a special code resource which knows about all the other code resources in memory. During execution time, this special code resource, called a "memory scheduler," can be called periodically, or at random or pseudo random intervals, at which time it intentionally shuffles the

³³⁹ Silva Declaration at ¶¶ 270-71.

³⁴⁰ Ex. 2, Prosecution History at 581 (original claim 62 issued as claim 14); *see also id.* at 415-16 (Patent Owner explaining that element 14.1 is met by teachings corresponding to Cooperman at 13:31-36).

³⁴¹ Silva Declaration at ¶ 273.

³⁴² Cooperman at 14:35-15:17.

other code resources randomly in memory, so that someone trying to analyze snapshots of memory at various intervals cannot be sure if they are looking at the same code or organization from one "break" to the next. This adds significant complexity to their job. The scheduler also randomly relocates itself when it is finished. In order to do this, the scheduler would have to first copy itself to a new location, and then specifically modify the program counter and stack frame, so that it could then jump into the new copy of the scheduler, but return to the correct calling frame. Finally, the scheduler would need to maintain a list of all memory addresses which contain the address of the scheduler, and change them to reflect its new location.³⁴³

Cooperman further describes its software code as including sub-objects that are code resources that provide entries point to the software's various functions:

The memory address of the first instruction in one of these sub-objects is called the "entry point" of the function or procedure. The rest of the instructions comprising that sub-object immediately follow from the entry point. Some systems may prefix information to the entry point which describes calling and return conventions for the code which follows, an example is the Apple Macintosh Operating System (MacOS). These sub-objects can be packaged into what are referred to in certain systems as "code resources," which may be stored separately from the application, or shared with other applications, although not necessarily.³⁴⁴

And Cooperman discloses that these code resources will be fixed once installed on the computer:

"Once the code resources of a program are loaded into memory, they typically remain in a fixed position."³⁴⁵

During the original prosecution, Patent Owner confirmed that such a teaching disclosed by Cooperman meets element 14.2. For example, Patent Owner's February 28, 2011 Remarks explain that element 14.2 is taught by:

Under the present invention, the application contains a special code resource which knows about all the other code resources in memory.

* * * *

During execution time, this special code resource, called a "memory scheduler," can be called periodically, or at random or pseudo random intervals, at which time it intentionally shuffles the other code resources randomly in memory, so that

³⁴³ Cooperman at 14:35-15:17; Silva Declaration at ¶ 275.

³⁴⁴ Cooperman at 7:19-30.

³⁴⁵ *Id.* at 13:31-32; Silva Declaration at ¶ 277.

someone trying to analyze snapshots of memory at various intervals cannot be sure if they are looking at the same code or organization from one "break" to the next. This adds significant complexity to their job. The scheduler also randomly relocates itself when it is finished. In order to do this, the scheduler would have to first copy itself to a new location, and then specifically modify the program counter and stack frame, so that it could then jump into the new copy of the scheduler, but return to the correct calling frame. Finally, the scheduler would need to maintain a list of all memory addresses which contain the address of the scheduler, and change them to reflect its new location.³⁴⁶

And as another example, Patent Owner's May 14, 2012 Appeal Brief states that element 14.2 is taught by:

The memory address of the first instruction in one of these sub-objects is called the "entry point" of the function or procedure. The rest of the instructions comprising that sub-object immediately follow from the entry point. Some systems may prefix information to the entry point which describes calling and return conventions for the code which follows, an example is the Apple Macintosh Operating System (MacOS). These sub-objects can be packaged into what are referred to in certain systems as "code resources," which may be stored separately from the application, or shared with other applications, although not necessarily.

* * * *

Once the code resources of a program are loaded into memory, they typically remain in a fixed position.³⁴⁷

Cooperman includes these same teachings, and thus discloses element 14.2.³⁴⁸

Moreover, during the original prosecution, Patent Owner specified that "interrelationships between code resource are not that which is novel."³⁴⁹ The Patent Owner continues by conceding:

What the examiner has implied by alleging that the "specification ... fails to teach or mention 'software code interrelationships'" is that software code interrelationships were somehow unknown in the art, which clearly is not the case. **As admitted, in the specification at the beginning of paragraph [0051], an**

³⁴⁶ Ex. 2, Prosecution History at 416 (original claim 62 issued as claim 14) *see also id.* at 669-71 (Patent Owner explaining that element 14.2 is met by teachings corresponding to Cooperman at 5:18-22, 6:30-7:36).

³⁴⁷ *Id.* at 581-82 (emphasis in original) (original claim 62 issued as claim 14).

³⁴⁸ Silva Declaration at ¶¶ 278-79.

³⁴⁹ Ex. 2, Prosecution History at 519.

"application" comprises "sub-objects" whose "order in the computer memory is of vital importance" in order to perform an intended function. And as admitted further in paragraph [0051], "When a program is compiled, then, it consists of a collection of these sub-objects, whose exact order or arrangement in memory is not important, so long as any sub-object which uses another sub-object knows where in memory it can be found." **Paragraph [0051] of course refers to conventional applications. Accordingly, that is admittedly a discussion of what is already know by one skilled in the art.** Accordingly, the examiner's statement that the specification lacks written description support for "software code interrelationships" is inconsistent with the fact that such **interrelationships were explained in paragraphs [0051] and [0052] as a fundamental basis of pre-existing modem computer programs.**³⁵⁰

Based on the Patent Owner's concession, it is clear that a POSITA would have understood that Cooperman's code resources necessarily define code interrelationships resulting in specific application functionalities once installed on a computer.³⁵¹

- d) ***Element 14.3: "encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key encoded software code"***

Element 14.3 is identical to element 12.3. As explained above, Cooperman discloses each limitation of element 12.3. For the same reasons, Cooperman teaches element 14.3.

And during the original prosecution, Patent Owner confirmed that such a teaching disclosed by Cooperman meets element 14.3. For example, Patent Owner's May 14, 2012 Appeal Brief states that element 14.3 is taught by:

The assembly utility can be supplied with a key generated from a license code generated *for the license in question.*

* * * *

The utility will choose one or several essential code resources, and encode them into one or several data resources using the stegacipher process.

* * * *

³⁵⁰ *Id.*

³⁵¹ Silva Declaration at ¶¶ 280-82.

The purpose of this scheme is to make a particular licensed copy of an application distinguishable from any other. It is not necessary to distinguish every instance of an application, merely every instance of a license.

* * * *

3) Once it has the license code, it can then generate the proper decoding key to access the essential code resources.³⁵²

As explained with respect to element 12.3, Cooperman includes these same teachings.³⁵³

Moreover, during the original prosecution, Patent Owner specified that “[e]ncoding using a key and an algorithm is known” and that “an interrelationship in software code is necessarily defined by digital data, and digital data can obviously be encoded by an encoding process.”³⁵⁴ As such, a POSITA would have understood that Cooperman’s encoding technique necessarily includes a first license key and an encoding algorithm to form a first license key encoded software code.³⁵⁵

e) ***Element 14.4: “in which at least one of said software code interrelationships are encoded”***

Cooperman discloses element 14.4. Specifically, Cooperman explains that its encoding technique results in the encoding of a software code interrelationship. Cooperman, for instance, states that the software code includes a data resource that specifies where in the code the code resource is encoded:

The application must also contain a data resource which specifies in which data resource a particular code resource is encoded. This data resource is created and added at assembly time by the assembly utility.³⁵⁶

³⁵² Ex. 2, Prosecution History at 582 (original claim 62 issued as claim 14); *see also id.* at 416 (Patent Owner explaining that element 14.3 is met by teachings corresponding to Cooperman at 10:7-20).

³⁵³ Silva Declaration at ¶¶ 284-86.

³⁵⁴ Ex. 2, Prosecution History at 519.

³⁵⁵ Silva Declaration at ¶ 287.

³⁵⁶ Cooperman at 11:20-24

And Cooperman further discloses that one of the code resources, such a memory scheduler, is encoded to include the software code interrelationships:

Under the present invention, the application contains a special code resource which knows about all the other code resources in memory. During execution time, this special code resource, called a "memory scheduler," can be called periodically, or at random or pseudo random intervals, at which time it intentionally shuffles the other code resources randomly in memory, so that someone trying to analyze snapshots of memory at various intervals cannot be sure if they are looking at the same code or organization from one "break" to the next."³⁵⁷

During the original prosecution, Patent Owner confirmed that such teachings disclosed by Cooperman meets element 14.4. For instance, Patent Owner's May 14, 2012 Appeal Brief states that element 14.4 is taught by:

The application must also contain a data resource which specifies in which data resource a particular code resource is encoded. This data resource is created and added at assembly time by the assembly utility.

* * * *

Under the present invention, the application contains a special code resource which knows about all the other code resources in memory. During execution time, this special code resource, called a "memory scheduler," can be called periodically, or at random or pseudo random intervals, at which time it intentionally shuffles the other code resources randomly in memory, so that someone trying to analyze snapshots of memory at various intervals cannot be sure if they are looking at the same code or organization from one "break" to the next."³⁵⁸

Cooperman includes this same teaching, and thus discloses element 14.4.³⁵⁹

Accordingly, Cooperman discloses claim 14.

D. SNQ-4: Claims 11, 12, 13, and 14 are Anticipated by Hasebe Under 35 U.S.C. §§ 102(a), (e).

Hasebe anticipates claims 11, 12, 13, and 14 under 35 U.S.C. §§ 102(a), (e).

³⁵⁷ Cooperman at 14:35-15:8.

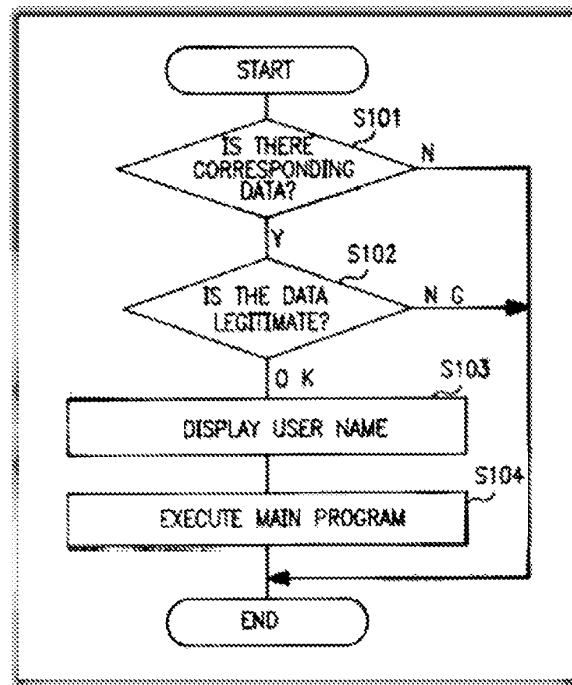
³⁵⁸ Ex. 2, Prosecution History at 577 (original claim 58 was issued as claim 11).

³⁵⁹ Silva Declaration at ¶¶ 289-91.

1. Hasebe Anticipates Independent Claim 11.

- a) **Preamble:** “A method for licensed software use, the method comprising”

Under the broadest reasonable construction, the preamble is non-limiting. Nevertheless, Hasebe discloses claim 11’s preamble. Hasebe describes a method of providing software to a user in a non-executable form as well as separate license information.³⁶⁰ And Hasebe teaches that the user uses the license information to convert the software into an executable form.³⁶¹ Hasebe’s Figure 6 illustrates this method for licensed software use:



Hasebe explains the steps of this method as follows:

When this software is actuated, as shown in FIG. 6, the CPU, first of all, by checking the contents ID in the license file, decides whether or not data corresponding to the software that is being actuated is present in the license file (step S101). Then, if the corresponding data exists (step S101:Y), the CPU performs

³⁶⁰ Hasebe at Abstract, 2:47-3:15.

³⁶¹ *Id.*

a check of the legitimacy of the corresponding data (step 102). In this step, the CPU encodes the information consisting of contents ID and user name stored in the license file using the signature key that is set as data in license display routine 25, and if the result of this encoding agrees with the signature information, decides that the data is legitimate.

If it is legitimate (step S102:OK), the CPU displays the user name which is read from the license file (step S103), and commences operation in accordance with the main program (step S104).

Also, if the corresponding data is not present in the license file (step S101:N) or if the content of the license file is found to be not legitimate (step S102:NG), i.e. if the content of the license file is found to be different from the result of the compilation performed by license file compilation unit 23, the CPU terminates operation without displaying the user name or executing the main program.³⁶²

As such, Hasebe teaches this preamble.³⁶³

- b) ***Element 11.1: “loading a software product on a computer, said computer comprising a processor, memory, an input, and an output, so that said computer is programmed to execute said software product”***

Hasebe discloses element 11.1. Hasebe describes a user’s computer having a processor and memory.³⁶⁴ For instance, Hasebe’s system includes a user terminal with a computer having a “CPU [that operates] when the software that is the subject of the present license system is actuated.”³⁶⁵ And Hasebe’s computer includes memory for storing software:

The user terminal comprises a storage unit, a conversion unit, and license file creating unit. In more detail, **the storage unit is employed for storing the license file and software converted to executable form.** The license information, which is generated by the license information generating unit in the management center, is given to the conversion unit. The conversion unit then converts the software to executable form using the license information and installs it in the storage unit. The license file creating unit creates the license file which contains the user

³⁶² *Id.* at 7:61-8:16.

³⁶³ Silva Declaration at ¶¶ 294-97.

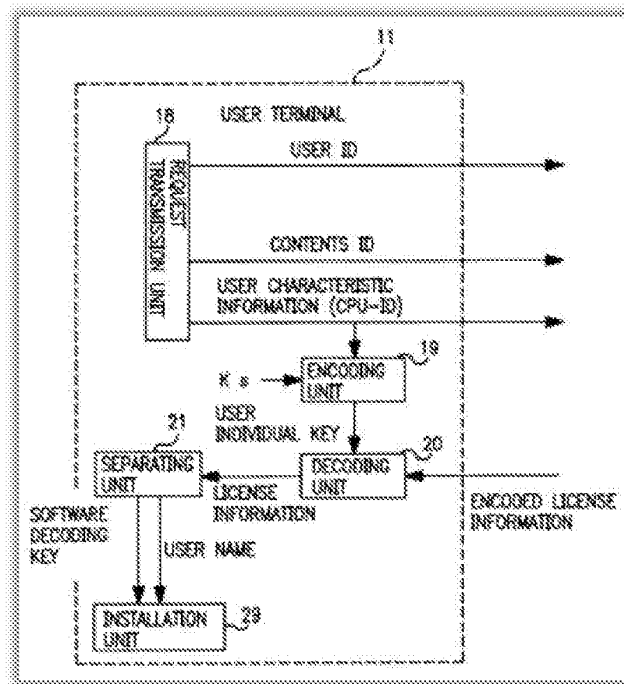
³⁶⁴ Hasebe at 3:62-67, 6:21-25, 7:50-53.

³⁶⁵ *Id.* at 7:50-53; *see also id.* at 6:21-25, 7:7-10, 7:61-8:16, 9:6-9.

identification information contained in the license information, and stores the license file in the storage unit.³⁶⁶

Moreover, Hasebe's computer includes an input (e.g., a keyboard) and output (e.g., a display).³⁶⁷

As shown below, Hasebe illustrates the user's terminal in Figure 7:



Hasebe further discloses loading a software product on the user's computer wherein the computer is programmed to execute the program. For instance, Hasebe details that its "software in non-executable form is presented to a user, and license information for converting the software into executable form is informed to the user on condition of payment of a charge, and the software is converted into executable form using this license information."³⁶⁸ And Hasebe

³⁶⁶ *Id.* at 2:66-3:10; *see also id.* at 3:62-67 ("convert[ing] the software to executable form using the license information stored in the license file and expands it into memory, and commences operation"), 8:53-59, claims 3, 14.

³⁶⁷ Hasebe at 7:1-10, 8:47-53, claim 5; *see also id.* at Abstract, 7:54-60, 8:6-21, 8:38-43, 9:33-39, Figs. 6, 9.

³⁶⁸ Hasebe at 2:47-54; *see also id.* at claim 1.

further describes loading the software onto the user's memory for execution.³⁶⁹ And as shown above, Hasebe's Figure 6 illustrates executing software loaded onto the user's computer using the license information.³⁷⁰

c) ***Element 11.2: "said software product outputting a prompt for input of license information"***

Hasebe discloses element 11.2. Specifically, Hasebe explains that its software product requests that the user input "license information" into the computer via the keyboard before the product can be used.³⁷¹ For instance, Hasebe explains that the software product prompts the user to input license information: "[N]otification of the contents ID etc to the management center and notification of the encoded license information to the user terminal were performed by another information transmission unit, such as the post... The user terminal is constituted such that installation is effected using encoded license information input from the keyboard."³⁷²

Moreover, Hasebe describes the use of a prompt to enter user ID information which management center 12 uses to generate the encoded-version of the license information:

Request transmission unit 18 commences operation when the keyboard (not shown) of user terminal 11 is operated in accordance with a prescribed procedure that is predetermined as the procedure for request of information for removal of functional restrictions. This request procedure includes keyboard input of the user ID and contents ID; request transmission unit 18 transmits to management center 12 the keyboard input information and the user's characteristic information, which is constituted by the ID of the CPU which is employed in user terminal 11.³⁷³

³⁶⁹ *Id.* at 3:28-34, 3:57-67, 8:47-52; *see also id.* at 3:11-15, 8:17-23, Figs. 6, 9.

³⁷⁰ Silva Declaration at ¶¶ 299-301.

³⁷¹ Hasebe at 7:1-10, 8:34-42.

³⁷² *Id.* at 8:34-42.

³⁷³ *Id.* at 7:1-10; *see also id.* at 6:60-7:10, claims 1-2.

A POSITA would have understood Hasebe's request for the user ID and contents ID for removal of functional restrictions corresponds to the software outputting a prompt to input license information.³⁷⁴

- d) ***Element 11.3: "said software product using license information entered via said input in response to said prompt in a routine designed to decode a first license code encoded in said software product"***

Hasebe discloses element 11.3. Hasebe describes that the user's computer receives "encoded license information" from management center 12:

When a request for information for removal of functional restrictions is received from user terminal 11, management center 12 sends to user terminal 11 encoded license information. As a result, after request transmission unit 18 has been operated, user terminal 11 receives encoded license information from management center 12.³⁷⁵

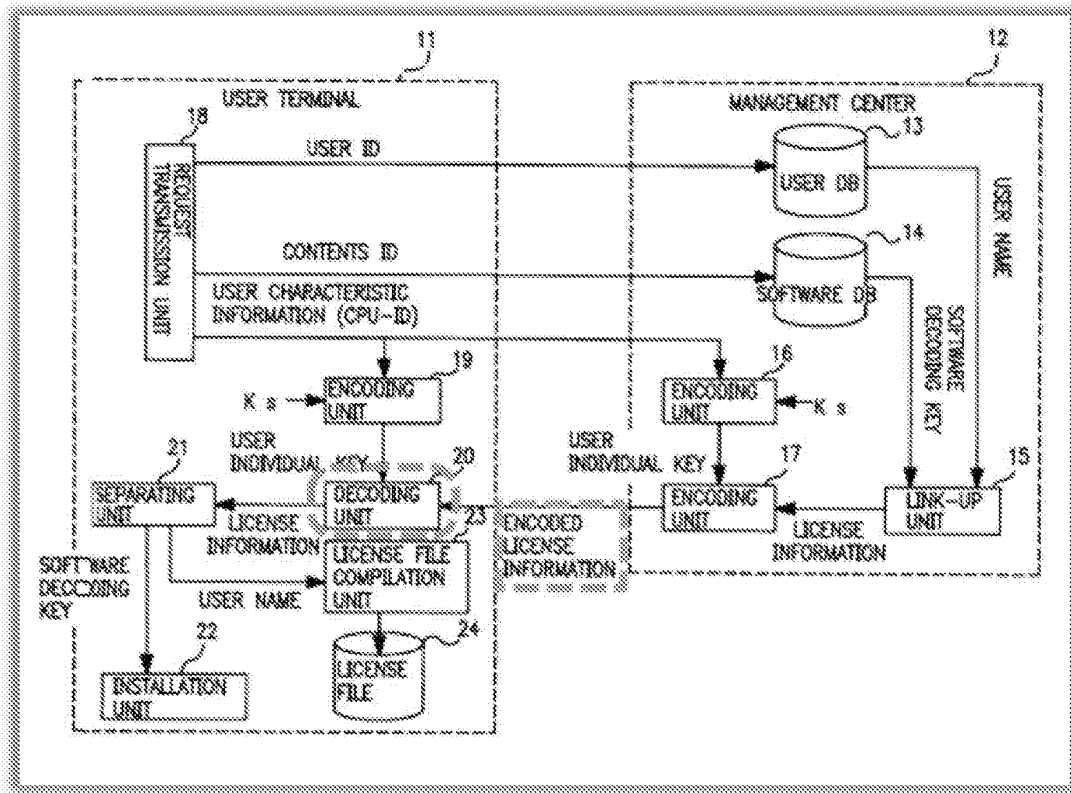
And Hasebe discloses that decoding unit 20 decodes a license code encoded in the software via a decode routine that uses the encoded license information.³⁷⁶ For instance, Hasebe details that its system will "make the software that is presented to the user encoded, and to make the conversion information for decoding the encoded software. Also, ... it is possible to employ information, as license information, which is the result of encoding the conversion information and user identification information, combined in integrated manner."³⁷⁷ As shown below in annotated Figure 1, Hasebe's system includes the input of "encoded license information" (dashed box) into the user's computer 11 which is used to decode the encoded software via decoding unit 20 (dashed oval):

³⁷⁴ Silva Declaration at ¶¶ 303-04.

³⁷⁵ Hasebe at 7:11-16; *see also id.* at 4:39-58, 6:42-50, Figs. 1, 7.

³⁷⁶ *Id.* at 7:17-31, 9:19-35.

³⁷⁷ *Id.* at 4:48-58; *see also id.* at 9:29-36.

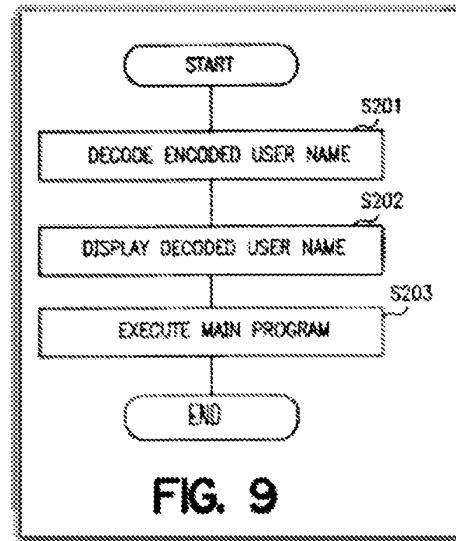
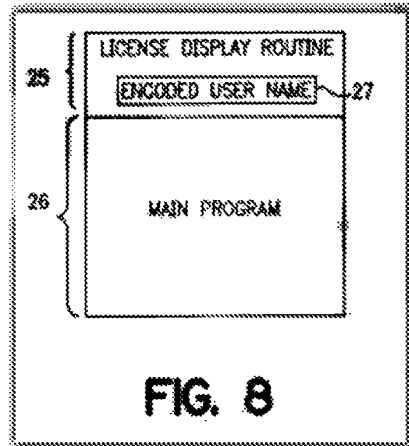


Moreover, Hasebe describes the decoding routine as:

- (a) decoding the license information, which includes the key and user name,
- (b) installing the encoded software using the decoded key,
- (c) writing the user name into the license display routine 25,
- (d) displaying the user name, and
- (e) executing the main portion of software program.³⁷⁸

Hasebe's Figure 8 illustrates the license code (routine 25) encoded into the software and main routine 26, and Figure 9 illustrates the decode routine that uses the license information to decode the license code:

³⁷⁸ *Id.* at 9:19-39.



A POSITA would have understood that Hasebe's routine 25, with the encoded user name 27, is a license code because it is encoded into the software program and controls the accessibility of the program.³⁷⁹ And as explained regarding element 11.2, Hasebe details that the user enters license information via an input in response to the prompt.³⁸⁰

Accordingly, Hasebe discloses claim 11.

2. Hasebe Anticipates Independent Claim 12.

- a) *Preamble: "A method for encoding software code using a computer having a processor and memory, the method comprising"*

Under the broadest reasonable construction, the preamble is non-limiting.³⁸¹

Nevertheless, Hasebe discloses claim 12's preamble. Specifically, Hasebe describes a method for

³⁷⁹ Silva Declaration at ¶¶ 307-11.

³⁸⁰ *Id.* at ¶ 312.

³⁸¹ Claim 12's preamble recites "a computer" and claim 12's body recites "a computer system." It is unclear whether those elements refer to the same or separate computing devices. For purposes of this Request and using the broadest reasonable interpretation consistent with the specification, it is assumed that the "computer" recited in the preamble is a device separate from the "computer system."

encoding software code using a computer with a processor and memory. Hasebe details that management center 32 generates the software code provided to the user via CD-ROM.³⁸² Alternatively, Hasebe's software code may be downloaded from the management center.³⁸³ And Hasebe explains that the link-up unit 15 of the management center performs "processing" reversed by separating unit 21.³⁸⁴ As such, A POSITA would have understood that the management center includes a processor and memory to create these CD-ROMs and to provide the downloading capability. As expert Dr. Silva explains, Hasebe's computer would necessarily include a processor and memory in order to function.³⁸⁵

As such, Hasebe teaches this preamble.

b) Element 12.1: "storing a software code in said memory"

Hasebe discloses element 12.1. As described with respect to claim 12's preamble, Hasebe's management center 32 either generates a CD-ROM containing the software code or provides downloadable versions of the software code.³⁸⁶ A POSITA thus would have understood that Hasebe's management center stores the software code in its memory for CD-ROM generation or user downloading because, as Dr. Silva explains, storage of code in memory is standard in computers like Hasebe's.³⁸⁷ And as shown in Hasebe's Figure 1, as annotated below, management center 14 includes a software database 14 (dashed box) capable of software storage:

³⁸² Hasebe at 1:9-14, 6:9-13, 9:22-26.

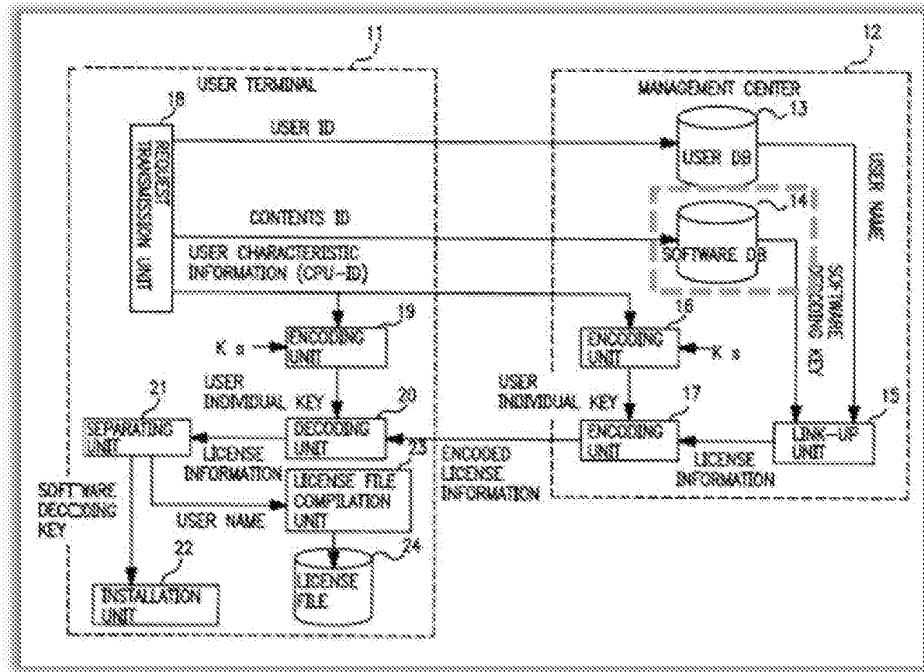
³⁸³ *Id.* at 9:60-64.

³⁸⁴ *Id.* at 7:23-26, Fig. 1.

³⁸⁵ Silva Declaration at ¶¶ 314-17.

³⁸⁶ Hasebe at 6:9-13, 9:22-26, 9:60-64.

³⁸⁷ Silva Declaration at ¶ 319.



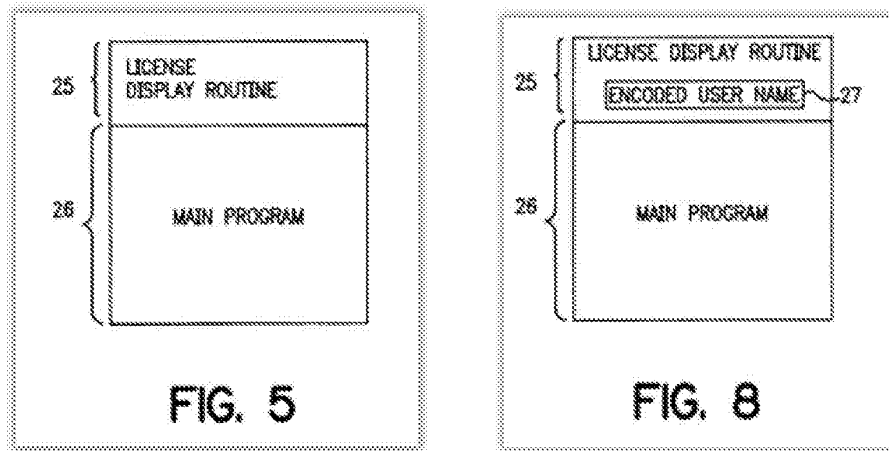
- c) **Element 12.2:** “wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system”

Hasebe discloses element 12.2. Hasebe teaches that its software code includes multiple code resources such as those used in license display routine 25.³⁸⁸ Hasebe explains that routine 25 determines whether the user’s license information is legitimate and, if so, permits access to the main program routine 26.³⁸⁹ For instance, Hasebe states: “In the main program there are defined the operating procedures relating to the proper functions of this software; in license display routine 25, there is defined the content to be executed prior to execution of main program 26.”³⁹⁰ Hasebe illustrates routines 25 and 26 of the software code in Figures 5 and 8:

³⁸⁸ Hasebe at 7:55-8:9, 9:25-35, Figs. 5, 8.

³⁸⁹ *Id.* at 7:65-8:9.

³⁹⁰ *Id.* at 7:55-60.



The '842 Patent refers to sub-objects, a memory scheduler, and data as examples of code resources.³⁹¹ Hasebe's routine 25 consists of software code that controls access to the underlying functionality of the software's main program, or sub-objects.³⁹² In this additional and alternative way, a POSITA would have understood that Hasebe's routine 25 contains a first code resource.³⁹³

Moreover, Hasebe's software code provides underlying functionalities when installed on the user's computer system (terminal 31). Hasebe, for instance, explains that the code's routine 25 provides access to the main program module 26 upon verification of the user's license information.³⁹⁴

³⁹¹ '842 Patent at 11:55-65, 15:36-42.

³⁹² Silva Declaration at ¶¶ 323-24.

³⁹³ *Id.* at ¶ 324.

³⁹⁴ Hasebe at 7:65-8:9, 9:20-36; Silva Declaration at ¶ 325.

d) ***Element 12.3: “encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key encoded software code”***

Hasebe discloses element 12.3. As discussed with respect to element 12.1, Hasebe’s management center 32 provides the user the software code via CD-ROM or download from the seller.³⁹⁵ Hasebe details that the management center 32 encodes the software code:

[I]t is also possible to make the software that is presented to the user encoded, and to make the conversion information for decoding the encoded software. Also, it is possible to employ, in such a licensee notification system, license information containing the user identification information in a form that cannot be separated without special information. For example, it is possible to employ information, as license information, which is the result of encoding the conversion information and user identification information, combined in integrated manner.³⁹⁶

It is also possible to constitute the system such that, instead of the user name and signature information, information representing the user name in encoded form is stored in the license file, and, when the installed software is executed, the information in the license file is decoded by the software and displayed.³⁹⁷

With respect to the code illustrated in Figure 9, Hasebe explains that the customer’s computer system “effects installation by decoding the software in the CD ROM using the software decoding key, and generates the user name in encoded form by encoding the user name.”³⁹⁸

Moreover, Hasebe describes its encoding technique uses a license key and an encoding algorithm. For instance, Hasebe states its system includes: “**a DES (data encryption standard) algorithm** [] employed for encoding and decoding.”³⁹⁹ And Hasebe details that the system uses a license key to encode the software code: “generat[ing] license information including user identification information encoded with **a characteristic key of the software.**”⁴⁰⁰ Figure 3, for

³⁹⁵ Hasebe at 6:9-13, 9:22-26, 9:60-64.

³⁹⁶ *Id.* at 4:48-58; *see also id.* at 7:32-38, 9:22-26.

³⁹⁷ *Id.* at 8:47-53.

³⁹⁸ *Id.* at 9:22-26.

³⁹⁹ *Id.* at 6:48-50.

⁴⁰⁰ *Id.* at 4:40-43; *see also id.* at 6:33-47, 7:33-38, 9:19-26.

example, illustrates a license key in the management center's software database 14 used to encode the software:

FIG. 3

CONTENTS ID	DECODING KEY
ABC00001	XXXXXXXX

As such, a POSITA would have understood that Hasebe's encoded software code utilizes the encoded license information to generate the claimed "first license key encoded software code."⁴⁰¹

Moreover, during the original prosecution, Patent Owner specified that "[e]ncoding using a key and an algorithm is known."⁴⁰² As such, a POSITA would have understood that Hasebe's encoding technique necessarily includes a first license key and an encoding algorithm to form a first license key encoded software code.⁴⁰³

- e) ***Element 12.4:*** "wherein, when installed on a computer system, said first license key encoded software code will provide said specified underlying functionality only after receipt of said first license key"

Hasebe discloses element 12.4. Hasebe describes the installation of the software code upon verification of the first license key by the user's computer.⁴⁰⁴ For instance, Hasebe details

⁴⁰¹ Silva Declaration at ¶¶ 328-30.

⁴⁰² Ex. 2, Prosecution History at 519.

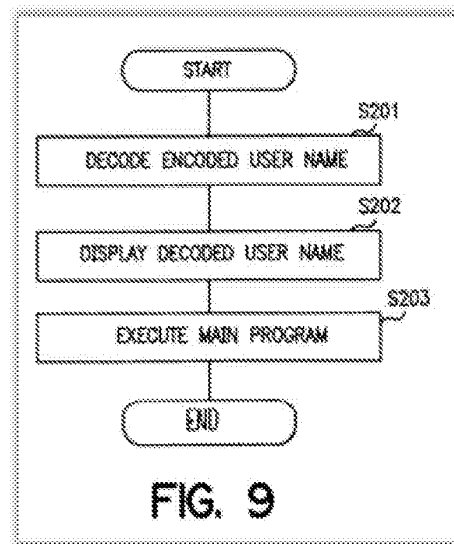
⁴⁰³ Silva Declaration at ¶ 331.

⁴⁰⁴ Hasebe at 3:5-15, 3:30-38, 9:19-39; *see also id.* at 7:32-38, 8:47-53.

the software code will provide access to specified underlying functionality of the code contained in main program routine 26 only after receipt of the first license key in license display routine 25:

- (a) decoding the license information, which includes the key and user name,
- (b) installing the encoded software using the decoded key,
- (c) writing the user name into the license display routine 25,
- (d) displaying the user name, and
- (e) executing the main portion routine 26 of software program.⁴⁰⁵

And Hasebe's Figure 9 illustrates the user's computer providing the underlying functionality of the main program routine 26 after the receipt and decoding of the first license key:



A POSITA would have understood that Hasebe's main program routine 26 includes specified underlying functionality of the first license key encoded software code accessible via confirmation of the encoded license key.⁴⁰⁶

Accordingly, Hasebe discloses claim 12.

⁴⁰⁵ Hasebe at 9:19-39.

⁴⁰⁶ Silva Declaration at ¶¶ 334-36.

3. Hasebe Anticipates Independent Claim 13.

a) *Preamble: “A method for encoding software code using a computer having a processor and memory, comprising”*

Under the broadest reasonable construction, the preamble is non-limiting. Nevertheless, Hasebe discloses claim 13’s preamble. Claim 13’s preamble is the same as claim 12’s preamble. As explained above, Hasebe discloses a method for encoding software using a computer with a processor and memory. As such, Hasebe teaches this preamble.⁴⁰⁷

b) *Element 13.1: “storing a software code in said memory”*

Element 13.1 is identical to element 12.1. As explained above, Hasebe discloses each limitation of element 12.1. For the same reasons, Hasebe teaches element 13.1.⁴⁰⁸

c) *Element 13.2: “wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system”*

Element 13.2 is identical to element 12.2. As explained above, Hasebe discloses each limitation of element 12.2. For the same reasons, Hasebe teaches element 13.2.⁴⁰⁹

d) *Element 13.3: “modifying, by said computer, using a first license key and an encoding algorithm, said software code, to form a modified software code; and wherein said modifying comprises encoding said first code resource to form an encoded first code resource”*

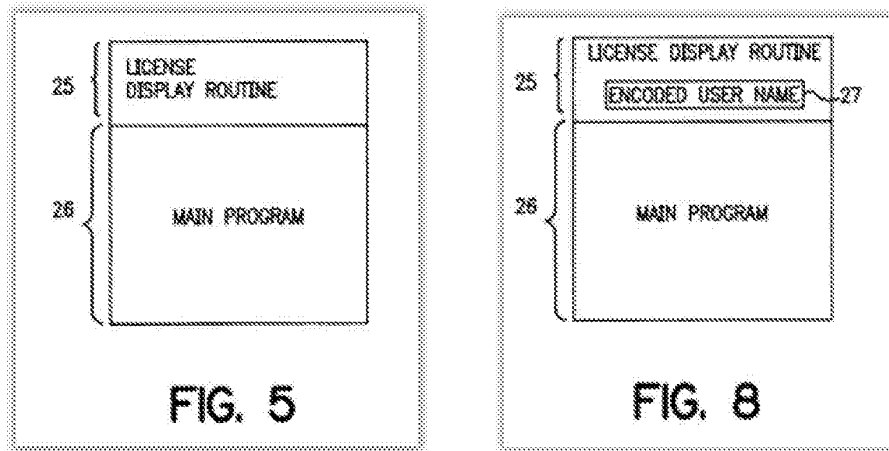
Hasebe discloses element 13.3. As described with respect to element 12.2, Hasebe’s system includes multiple code resources (e.g., license display routine 25) for accessing software functionality.⁴¹⁰ Hasebe illustrates routine 25 and main program routine 26 of the software code in Figures 5 and 8:

⁴⁰⁷ *Id.* at ¶¶ 338-39.

⁴⁰⁸ *Id.* at ¶ 341.

⁴⁰⁹ *Id.* at ¶ 343.

⁴¹⁰ Hasebe at 7:55-8:9, 9:25-35, Figs. 5, 8; Silva Declaration at ¶ 345.



And as described with respect to element 12.3, Hasebe's computer⁴¹¹ in management center 12 modifies the software code to form an encoded first code resource.⁴¹² For example, Hasebe's software code is modified to include routine 25 used for verification of the user's license information, which permits execution of the software code.⁴¹³

Hasebe discloses that its code modification uses a license key and an encoding algorithm, as described with respect to element 12.3.⁴¹⁴ Moreover, during the original prosecution, Patent Owner specified that "[e]ncoding using a key and an algorithm is known."⁴¹⁵ As such, a POSITA would have understood that Hasebe's encoding technique necessarily includes a first license key and an encoding algorithm to form an encoded first code resource.⁴¹⁶

⁴¹¹ Hasebe at 6:21-24.

⁴¹² *Id.* at 4:48-58, 8:47-53; *see also id.* at 7:32-38, 9:22-26; Silva Declaration at ¶ 346.

⁴¹³ Hasebe at 4:48-58, 8:47-53; Silva Declaration at ¶ 346.

⁴¹⁴ Hasebe at 6:48-50, 4:40-43, Fig. 3; *see also id.* at 6:33-47, 7:33-38, 9:19-26.

⁴¹⁵ Ex. 2, Prosecution History at 519.

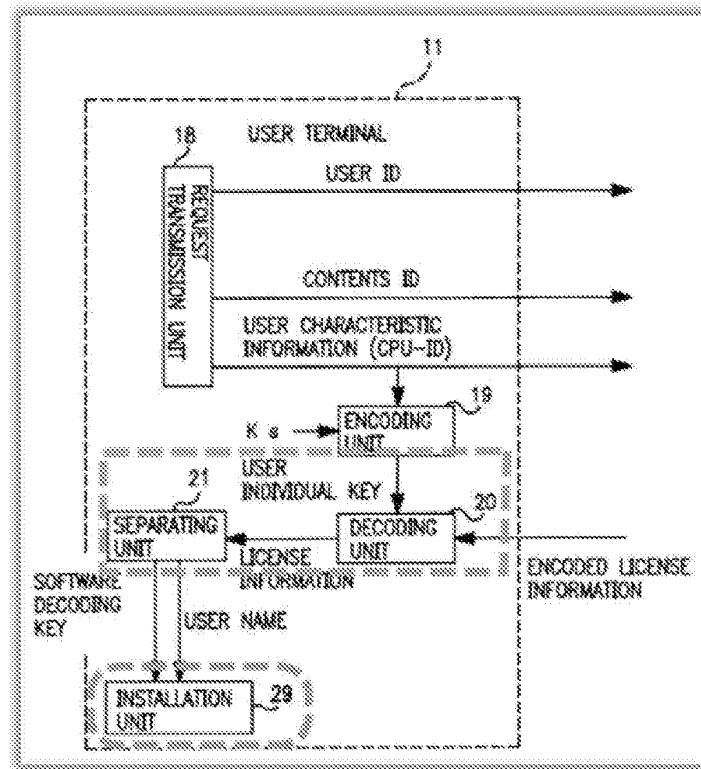
⁴¹⁶ Silva Declaration at ¶ 347.

e) ***Element 13.4: “wherein said modified software code comprises said encoded first code resource, and a decode resource for decoding said encoded first code resource”***

Hasebe discloses element 13.4. As described with respect to element 13.3, Hasebe’s modified software code includes the encoded first code resource. And Hasebe details that user terminal 11 includes decode unit 20 and separating unit 21 to produce the decoding key for the relevant software code.⁴¹⁷ Hasebe’s user terminal sends the decoding key to the software installation unit (Fig. 1’s unit 22 or Fig. 7’s unit 29), and “[i]nallation unit 29 effects installation by decoding the software in the CD ROM using the software decoding key, and generates the user name in encoded form by encoding the user name.”⁴¹⁸ As shown below in annotated Figure 7, Hasebe’s user terminal 11 includes a decode resource including the separating and decoding units 20, 21 (dashed box) and installation unit 22 (dashed oval) to decode the encoded code resource for software execution:

⁴¹⁷ Hasebe at 7:17-31.

⁴¹⁸ *Id.* at 7:27-39, 9:22-26.



As such, Hasebe teaches a decoding resource for decoding the encoded first code resource.⁴¹⁹

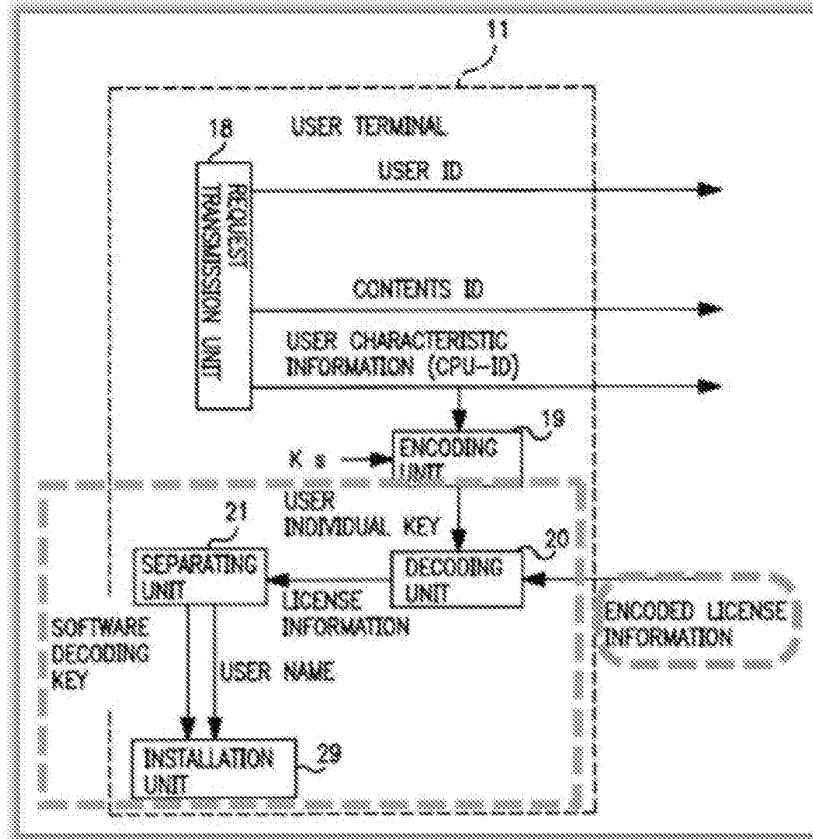
- f) **Element 13.5: “wherein said decode resource is configured to decode said encoded first code resource upon receipt of said first license key”**

Hasebe discloses element 13.5. As described with respect to element 12.3, Hasebe details that the system uses a license key to encode the software code: “generat[ing] license information including user identification information encoded with a **characteristic key of the software**.”⁴²⁰ And Hasebe specifies that its decode resource decodes the encoded first code resource upon receipt of the license key. For instance, Hasebe teaches that the user terminal receives the encoded license information at decoding unit 20, decodes the information to produce the

⁴¹⁹ Silva Declaration at ¶ 350.

⁴²⁰ Hasebe at 4:40-43; *see also id.* at 6:33-47, 7:33-38, 9:19-26, Fig. 3.

decoding key, and decodes the encode first code resource (routine 25) “by decoding the software in the CD ROM using the software decoding key.”⁴²¹ Figure 7, as annotated below, shows the decode resource (dashed box) receiving the first license key (dashed oval) to decode the encoded software—including the encoded first code resource:



Accordingly, Hasebe discloses claim 13.

⁴²¹ *Id.* at 7:27-39, 9:22-26; Silva Declaration at ¶ 353.

4. Hasebe Anticipates Independent Claim 14.

a) *Preamble: “A method for encoding software code using a computer having a processor and memory, comprising”*

Under the broadest reasonable construction, the preamble is non-limiting. Nevertheless, Hasebe discloses claim 14’s preamble. Claim 14’s preamble is the same as each of claim 12 and 13’s preamble. As explained above, Hasebe discloses a method for encoding software using a computer with a processor and memory. As such, Hasebe teaches this preamble.⁴²²

b) *Element 14.1: “storing a software code in said memory”*

Element 14.1 is identical to element 12.1. As explained above, Hasebe discloses each limitation of element 12.1. For the same reasons, Hasebe teaches element 14.1.⁴²³

c) *Element 14.2: “wherein said software code defines software code interrelationships between code resources that result in a specified underlying functionality when installed on a computer system”*

Hasebe discloses element 14.2. Hasebe explains that its software code interrelates code resources relating to routines 25 and 26 upon verification of the license key.⁴²⁴ For instance, Hasebe details that its software code includes routine 25 which permits access to the main program routine 26 upon validation of user’s license information.⁴²⁵ Hasebe states: “In the main program there are defined the operating procedures relating to the proper functions of this software; in license display routine 25, there is defined the content to be executed prior to execution of main program 26.”⁴²⁶ Hasebe illustrates routines 25 and 26 of the software code in Figures 5 and 8:

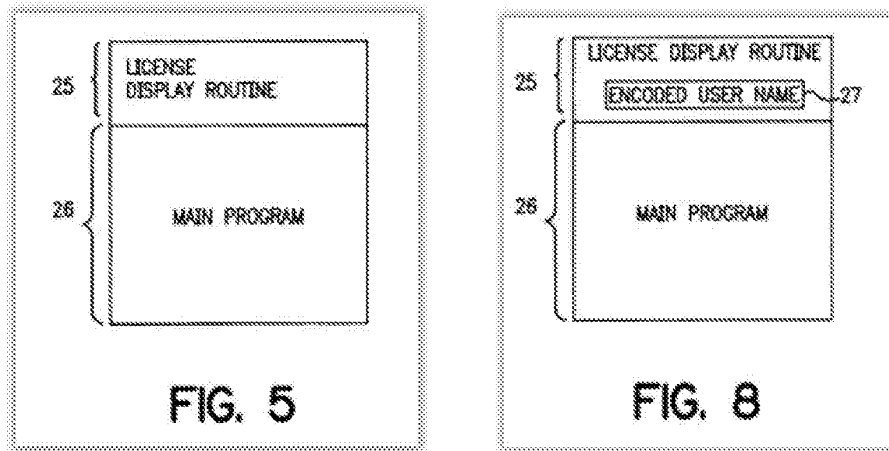
⁴²² Silva Declaration at ¶¶ 356-57.

⁴²³ *Id.* at ¶ 359.

⁴²⁴ Hasebe at 7:55-8:9, Figs. 5, 8, 9.

⁴²⁵ *Id.* at 7:65-8:9.

⁴²⁶ *Id.* at 7:55-60.



Moreover, the '842 Patent refers to sub-objects and a memory scheduler as examples of code resources.⁴²⁷ Hasebe's routine 25 contains a sub-object of the software code because it controls access to the underlying functionality of the software's main program.⁴²⁸ And Hasebe specifies routine 25 "directly rewrite[es]" the software code when the software code is decoded.⁴²⁹ In this additional and alternative way, a POSITA would have understood that Hasebe's routines 25 and 26 are code resources and that the software code defines software code interrelationships between these code resources.⁴³⁰ And a POSITA would have understood that the interrelationship between Hasebe's routines 25 and 26 result in a specified underlying functionality upon code installation.⁴³¹

⁴²⁷ '842 Patent at 11:55-65, 15:36-42.

⁴²⁸ Silva Declaration at ¶¶ 361-62.

⁴²⁹ Hasebe at 5:10-32, 9:22-39.

⁴³⁰ Silva Declaration at ¶ 362.

⁴³¹ *Id.* at ¶ 362.

Moreover, during the original prosecution, Patent Owner specified that “interrelationships between code resource are not that which is novel.”⁴³² Patent Owner continued by conceding:

What the examiner has implied by alleging that the "specification ... fails to teach or mention 'software code interrelationships'" is that software code interrelationships were somehow unknown in the art, which clearly is not the case. **As admitted, in the specification at the beginning of paragraph [0051], an "application" comprises "sub-objects" whose "order in the computer memory is of vital importance" in order to perform an intended function.** And as admitted further in paragraph [0051], "When a program is compiled, then, it consists of a collection of these sub-objects, whose exact order or arrangement in memory is not important, so long as any sub-object which uses another sub-object knows where in memory it can be found." **Paragraph [0051] of course refers to conventional applications. Accordingly, that is admittedly a discussion of what is already know by one skilled in the art.** Accordingly, the examiner's statement that the specification lacks written description support for "software code interrelationships" is inconsistent with the fact that such **interrelationships were explained in paragraphs [0051] and [0052] as a fundamental basis of pre-existing modem computer programs.**⁴³³

Based on the Patent Owner's concession, it is clear that a POSITA would have understood that Hasebe's code resources necessarily define code interrelationships resulting in specific underlying functionality once installed on a computer.⁴³⁴

- d) ***Element 14.3: “encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key encoded software code”***

Element 14.3 is identical to element 12.3. As explained above, Hasebe discloses each limitation of element 12.3. For the same reasons, Hasebe teaches element 14.3.⁴³⁵

Moreover, during the original prosecution, Patent Owner specified that “[e]ncoding using a key and an algorithm is known” and that “an interrelationship in software code is necessarily

⁴³² Ex. 2, Prosecution History at 519.

⁴³³ *Id.*

⁴³⁴ Silva Declaration at ¶¶ 363-64.

⁴³⁵ *Id.* at ¶ 367.

defined by digital data, and digital data can obviously be encoded by an encoding process.”⁴³⁶ As such, a POSITA would have understood that Hasebe’s encoding technique necessarily includes a first license key and an encoding algorithm to form a first license key encoded software code.⁴³⁷

e) ***Element 14.4: “in which at least one of said software code interrelationships are encoded”***

Hasebe discloses element 14.4. As described with respect to element 14.2, Hasebe teaches that its software code defines code interrelationships between code resources and routine 25 control certain underlying software functionality. Hasebe further details that its software code is encoded:

[I]t is also possible to make the software that is presented to the user encoded, and to make the conversion information for decoding the encoded software. Also, it is possible to employ, in such a licensee notification system, license information containing the user identification information in a form that cannot be separated without special information. For example, it is possible to employ information, as license information, which is the result of encoding the conversion information and user identification information, combined in integrated manner.⁴³⁸

It is also possible to constitute the system such that, instead of the user name and signature information, information representing the user name in encoded form is stored in the license file, and, when the installed software is executed, the information in the license file is decoded by the software and displayed.⁴³⁹

And Hasebe states that the software code includes the code interrelationships between routines 25 and 26, all of which would be encoded as part of the software code.⁴⁴⁰

Accordingly, Hasebe discloses claim 14.

⁴³⁶ Ex. 2, Prosecution History at 519.

⁴³⁷ Silva Declaration at ¶ 368.

⁴³⁸ Hasebe at 4:48-58; *see also id.* at 7:32-38, 9:22-26.

⁴³⁹ *Id.* at 8:47-53.

⁴⁴⁰ *Id.* at 7:55-8:9, Figs. 5, 8, 9; Silva Declaration at ¶¶ 370-71.

XI. CONCLUSION

As shown above, the prior art references establish that independent claims 11, 12, 13, and 14 are invalid as anticipated. In light of the substantial new questions of patentability raised by these references, Requester respectfully seeks *ex parte* reexamination of claims 11, 12, 13, and 14 of the '842 Patent.

As identified in the attached Certificate of Service and in accordance with 37 C.F.R. §§ 1.33(c) and 1.510(b)(5), a copy of the present Request, in its entirety, is being served to the address of the attorney of record reflected in the publicly available records of the United States Patent & Trademark Office's Patent Application Information Retrieval system.

Please direct all correspondence in this matter to the undersigned.

Dated: May 16, 2018

By: /Joseph F. Edell/
Joseph F. Edell
Reg. No. 67,625
Counsel for Requester

Fisch Sigler LLP
5301 Wisconsin Avenue NW
Fourth Floor
Washington, DC 20015
Phone: (202) 362-3524
Fax: (202) 362-3501

Exhibit 1



US009104842B2

(12) **United States Patent**
Moskowitz

(10) **Patent No.:** **US 9,104,842 B2**
(45) **Date of Patent:** **Aug. 11, 2015**

(54) **DATA PROTECTION METHOD AND DEVICE**

(76) Inventor: **Scott A. Moskowitz**, Sunny Isles Beach, FL (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 1965 days.

(21) Appl. No.: **11/895,388**

(22) Filed: **Aug. 24, 2007**

(65) **Prior Publication Data**

US 2008/0016365 A1 Jan. 17, 2008

Related U.S. Application Data

(60) Division of application No. 10/602,777, filed on Jun. 25, 2003, now Pat. No. 7,664,263, which is a continuation of application No. 09/046,627, filed on Mar. 24, 1998, now Pat. No. 6,598,162.

(51) **Int. Cl.**

G06F 21/16 (2013.01)
G06F 21/10 (2013.01)
G06F 21/12 (2013.01)
G06F 21/33 (2013.01)
G06T 1/00 (2006.01)
H04L 9/06 (2006.01)
H04L 9/32 (2006.01)

(52) **U.S. Cl.**

CPC **G06F 21/10** (2013.01); **G06F 21/125** (2013.01); **G06F 21/16** (2013.01); **G06F 21/335** (2013.01); **G06T 1/0021** (2013.01); **H04L 9/065** (2013.01); **H04L 9/3236** (2013.01); **H04L 9/3247** (2013.01); **G06F 2211/007** (2013.01); **G06F 2221/0737** (2013.01); **G06F 2221/2107** (2013.01); **G06T 2201/0064** (2013.01); **G06T 2201/0083** (2013.01); **H04L 2209/605** (2013.01); **H04L 2209/608** (2013.01)

(58) **Field of Classification Search**

CPC H04L 63/0428; H04L 2209/608; H04L 2209/60

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

3,947,825 A 3/1976 Cassada
3,984,624 A 10/1976 Waggener
3,986,624 A 10/1976 Cates, Jr. et al.
4,038,596 A 7/1977 Lee
4,200,770 A 4/1980 Hellman et al.
4,218,582 A 8/1980 Hellman et al.
4,339,134 A 7/1982 Macheel
4,390,898 A 6/1983 Bond et al.
4,405,829 A 9/1983 Rivest et al.
4,424,414 A 1/1984 Hellman et al.
4,528,588 A 7/1985 Lofberg

(Continued)

FOREIGN PATENT DOCUMENTS

EP 0372601 6/1990
EP 0372601 A1 6/1990

(Continued)

OTHER PUBLICATIONS

U.S. Appl. No. 08/999,766, filed Jul. 23, 1997, entitled "Steganographic Method and Device".

(Continued)

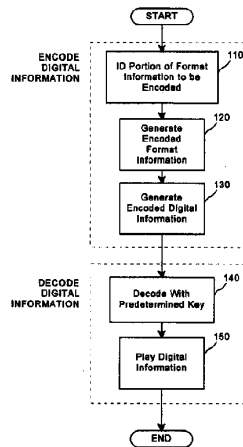
Primary Examiner — Izunna Okeke

(74) *Attorney, Agent, or Firm* — Neifeld IP Law, PC

(57) **ABSTRACT**

An apparatus and method for encoding and decoding additional information into a digital information in an integral manner. More particularly, the invention relates to a method and device for data protection.

14 Claims, 1 Drawing Sheet



(56)

References Cited

U.S. PATENT DOCUMENTS

4,529,870	A	7/1985	Chaum	5,579,124	A	11/1996	Aijala et al.
4,633,462	A	12/1986	Stifle	5,581,703	A	12/1996	Baugher et al.
4,672,605	A	6/1987	Hustig et al.	5,583,488	A	12/1996	Sala et al.
4,748,668	A	5/1988	Shamir et al.	5,598,470	A	1/1997	Cooper et al.
4,749,354	A	6/1988	Kernan	5,606,609	A	* 2/1997	Houser et al. 713/179
4,789,928	A	12/1988	Fujisaki	5,613,004	A	3/1997	Cooperman et al.
4,790,564	A	12/1988	Larcher	5,617,119	A	4/1997	Briggs et al.
4,827,508	A	5/1989	Shear	5,617,506	A	4/1997	Burk
4,855,584	A	8/1989	Tomiyama	5,625,690	A	4/1997	Michel et al.
4,876,617	A	10/1989	Best et al.	5,629,980	A	5/1997	Stefik et al.
4,896,275	A	1/1990	Jackson	5,633,932	A	5/1997	Davis et al.
4,908,873	A	3/1990	Philibert et al.	5,634,040	A	5/1997	Her et al.
4,939,515	A	7/1990	Adelson	5,636,276	A	6/1997	Brugger
4,969,204	A	11/1990	Melnychuk et al.	5,636,292	A	6/1997	Rhoads
4,972,471	A	11/1990	Gross et al.	5,640,569	A	6/1997	Miller et al.
4,977,594	A	12/1990	Shear	5,644,727	A	7/1997	Atkins
4,979,210	A	12/1990	Nagata et al.	5,646,997	A	7/1997	Barton
4,980,782	A	12/1990	Ginkel	5,649,284	A	7/1997	Yoshinobu
5,050,213	A	9/1991	Shear	5,657,461	A	8/1997	Harkins et al.
5,073,925	A	12/1991	Nagata et al.	5,659,726	A	8/1997	Sandford, II et al.
5,077,665	A	12/1991	Silverman et al.	5,664,018	A	9/1997	Leighton
5,103,461	A	4/1992	Tymes	5,673,316	A	9/1997	Auerbach et al.
5,111,530	A	5/1992	Kutaragi	5,675,653	A	10/1997	Nelson
5,113,437	A	5/1992	Best et al.	5,677,952	A	10/1997	Blakley et al.
5,123,045	A	6/1992	Ostrovsky	5,680,462	A	10/1997	Miller et al.
5,136,581	A	8/1992	Muehrcke	5,687,236	A	11/1997	Moskowitz et al.
5,136,646	A	8/1992	Haber et al.	5,689,587	A	11/1997	Bender et al.
5,136,647	A	8/1992	Haber et al.	5,696,828	A	12/1997	Koopman, Jr.
5,142,576	A	8/1992	Nadan	5,719,937	A	2/1998	Warren et al.
5,161,210	A	11/1992	Druyvesteyn et al.	5,721,781	A	2/1998	Deo
5,164,992	A	11/1992	Turk	5,721,788	A	2/1998	Powell et al.
5,189,411	A	2/1993	Collar	5,734,752	A	3/1998	Knox
5,210,820	A	5/1993	Kenyon	5,737,416	A	4/1998	Cooper et al.
5,243,423	A	9/1993	DeJean et al.	5,737,733	A	4/1998	Eller
5,243,515	A	9/1993	Lee	5,740,244	A	4/1998	Indeck et al.
5,287,407	A	* 2/1994	Holmes 705/58	5,745,569	A	4/1998	Moskowitz et al.
5,291,560	A	3/1994	Daugman	5,748,783	A	5/1998	Rhoads
5,293,633	A	3/1994	Robbins	5,751,811	A	5/1998	Magnotti et al.
5,297,032	A	3/1994	Trojan	5,754,697	A	5/1998	Fu et al.
5,319,735	A	6/1994	Preuss et al.	5,754,938	A	5/1998	Hcrz
5,327,520	A	7/1994	Chen	5,757,923	A	5/1998	Koopman, Jr.
5,341,429	A	8/1994	Stringer et al.	5,765,152	A	6/1998	Erickson
5,341,477	A	8/1994	Pitkin et al.	5,768,396	A	6/1998	Sone
5,363,448	A	11/1994	Koopman et al.	5,774,452	A	6/1998	Wolosewicz
5,365,586	A	11/1994	Indeck et al.	5,781,184	A	7/1998	Wasserman
5,369,707	A	11/1994	Follendore, III	5,790,677	A	8/1998	Fox et al.
5,375,055	A	12/1994	Togher	5,790,783	A	8/1998	Lee
5,379,345	A	1/1995	Greenberg	5,799,083	A	8/1998	Brothers et al.
5,394,324	A	2/1995	Clearwater	5,809,139	A	9/1998	Girod et al.
5,398,285	A	3/1995	Borgelt et al.	5,809,160	A	9/1998	Powell et al.
5,406,627	A	4/1995	Thompson et al.	5,818,818	A	10/1998	Soumiya
5,408,505	A	4/1995	Indeck et al.	5,822,432	A	10/1998	Moskowitz et al.
5,410,598	A	4/1995	Shear	5,822,436	A	10/1998	Rhoads
5,412,718	A	5/1995	Narasimhalv et al.	5,828,325	A	10/1998	Wolosewicz et al.
5,418,713	A	5/1995	Allen	5,832,119	A	11/1998	Rhoads
5,428,606	A	6/1995	Moskowitz	5,839,100	A	11/1998	Wegener
5,437,050	A	7/1995	Lamb	5,842,213	A	11/1998	Odom
5,450,490	A	9/1995	Jensen et al.	5,845,266	A	12/1998	Lupien
5,469,536	A	11/1995	Blank	5,848,155	A	12/1998	Cox
5,471,533	A	11/1995	Wang et al.	5,850,481	A	12/1998	Rhoads
5,478,990	A	12/1995	Montanari et al.	5,859,920	A	1/1999	Daly et al.
5,479,210	A	12/1995	Cawley et al.	5,860,099	A	1/1999	Milios et al.
5,487,168	A	1/1996	Geiner et al.	5,862,260	A	1/1999	Rhoads
5,493,677	A	2/1996	Balogh et al.	5,864,827	A	1/1999	Wilson
5,497,419	A	3/1996	Hill	5,870,474	A	2/1999	Wasilewski et al.
5,506,795	A	4/1996	Yamakawa	5,875,437	A	2/1999	Atkins
5,513,126	A	4/1996	Harkins et al.	5,884,033	A	3/1999	Duvall et al.
5,513,261	A	4/1996	Maher	5,889,868	A	3/1999	Moskowitz et al.
5,530,739	A	6/1996	Okada	5,892,900	A	4/1999	Ginter
5,530,751	A	6/1996	Morris	5,893,067	A	4/1999	Bender et al.
5,530,759	A	6/1996	Braudaway et al.	5,894,521	A	4/1999	Conley
5,539,735	A	7/1996	Moskowitz	5,901,178	A	5/1999	Lee
5,548,579	A	8/1996	Lebrun et al.	5,903,721	A	5/1999	Sixtus
5,568,570	A	10/1996	Rabbani	5,905,800	A	5/1999	Moskowitz et al.
5,570,339	A	10/1996	Nagano	5,905,975	A	5/1999	Ausubel
				5,912,972	A	6/1999	Barton
				5,915,027	A	6/1999	Cox et al.
				5,917,915	A	6/1999	Herose
				5,918,223	A	6/1999	Blum

(56)

References Cited

U.S. PATENT DOCUMENTS

5,920,900	A	7/1999	Poole et al.	6,363,488	B1	3/2002	Ginter
5,923,763	A	7/1999	Walker et al.	6,373,892	B1	4/2002	Ichien et al.
5,930,369	A	7/1999	Cox et al.	6,373,960	B1	4/2002	Conover et al.
5,930,377	A	7/1999	Powell et al.	6,374,036	B1	4/2002	Ryan et al.
5,940,134	A	8/1999	Wirtz	6,377,625	B1	4/2002	Kim
5,943,422	A	8/1999	Van Wie et al.	6,381,618	B1	4/2002	Jones et al.
5,949,055	A	9/1999	Fleet	6,381,747	B1	4/2002	Wonfor et al.
5,949,973	A	9/1999	Yarom	6,385,324	B1	5/2002	Koppen
5,963,909	A	10/1999	Warren et al.	6,385,329	B1	5/2002	Sharma et al.
5,973,731	A	10/1999	Schwab	6,385,596	B1	5/2002	Wiser
5,974,141	A	10/1999	Saito	6,389,402	B1	5/2002	Ginter
5,991,426	A	11/1999	Cox et al.	6,389,538	B1	5/2002	Gruse et al.
5,991,431	A	11/1999	Borza	6,398,245	B1	6/2002	Gruse
5,999,217	A	12/1999	Berners-Lee	6,405,203	B1	6/2002	Collart
6,009,176	A	12/1999	Gennaro et al.	6,415,041	B1	7/2002	Oami et al.
6,018,722	A	1/2000	Ray	6,418,421	B1	7/2002	Hurtado
6,029,126	A	2/2000	Malvar	6,425,081	B1	7/2002	Iwamura
6,029,146	A	2/2000	Hawkins	6,427,140	B1	7/2002	Ginter
6,029,195	A	2/2000	Herz	6,430,301	B1	8/2002	Petrovic
6,032,957	A	3/2000	Kiyosaki	6,430,302	B2	8/2002	Rhoads
6,035,398	A	3/2000	Bjorn	6,442,283	B1	8/2002	Tewfik et al.
6,041,316	A	3/2000	Allen	6,446,211	B1	9/2002	Colvin
6,044,471	A	3/2000	Colvin	6,453,252	B1	9/2002	Iaroche
6,049,838	A	4/2000	Miller et al.	6,457,058	B1	9/2002	Ullum et al.
6,051,029	A	4/2000	Paterson et al.	6,463,468	B1	10/2002	Buch et al.
6,061,793	A	5/2000	Tewfik et al.	6,480,937	B1	11/2002	Vorbach
6,067,622	A	5/2000	Moore	6,480,963	B1	11/2002	Tachibana
6,069,914	A	5/2000	Cox	6,484,153	B1	11/2002	Walker
6,078,664	A	6/2000	Moskowitz et al.	6,484,264	B1	11/2002	Colvin
6,081,251	A	6/2000	Sakai et al.	6,493,457	B1	12/2002	Quackenbush
6,081,587	A	6/2000	Reyes et al.	6,502,195	B1	12/2002	Colvin
6,081,597	A	6/2000	Hoffstein	6,510,513	B1	1/2003	Danieli
6,088,455	A	7/2000	Logan et al.	6,522,767	B1	2/2003	Moskowitz et al.
6,108,722	A	8/2000	Troeller	6,522,769	B1	2/2003	Rhoads et al.
6,111,517	A	8/2000	Atick	6,523,113	B1	2/2003	Wehrenberg
6,131,162	A	10/2000	Yoshiura et al.	6,530,021	B1	3/2003	Epstein et al.
6,134,535	A	10/2000	Belzberg	6,532,284	B2	3/2003	Walker et al.
6,138,239	A	10/2000	Veil	6,532,298	B1	3/2003	Cambior
6,141,753	A	10/2000	Zhao et al.	6,539,475	B1	3/2003	Cox et al.
6,141,754	A	10/2000	Choy	6,556,976	B1	4/2003	Callen
6,148,333	A	11/2000	Guedalia	6,557,103	B1	4/2003	Boncelet, Jr. et al.
6,154,571	A	11/2000	Cox et al.	6,574,608	B1	6/2003	Dahod
6,173,322	B1	1/2001	Hu	6,584,125	B1	6/2003	Katto
6,178,405	B1	1/2001	Ouyang	6,587,837	B1	7/2003	Spagna et al.
6,185,683	B1	2/2001	Ginter	6,590,996	B1	7/2003	Reed
6,192,138	B1	2/2001	Yamadaji	6,594,643	B1	7/2003	Freeny
6,199,058	B1	3/2001	Wong et al.	6,598,162	B1	7/2003	Moskowitz
6,205,249	B1	3/2001	Moskowitz	6,601,044	B1	7/2003	Wallman
6,208,745	B1	3/2001	Florencio et al.	6,606,393	B1	8/2003	Xie et al.
6,226,618	B1	5/2001	Downs	6,611,599	B2	8/2003	Natarajan
6,230,268	B1	5/2001	Miwa et al.	6,615,188	B1	9/2003	Breen
6,233,347	B1	5/2001	Chen et al.	6,618,188	B2	9/2003	Iiaga
6,233,566	B1	5/2001	Levine	6,647,424	B1	11/2003	Pearson et al.
6,233,684	B1	5/2001	Stefik et al.	6,650,761	B1	11/2003	Rodriguez
6,240,121	B1	5/2001	Senoh	6,658,010	B1	12/2003	Enns et al.
6,253,193	B1	6/2001	Ginter	6,665,489	B2	12/2003	Collart
6,263,313	B1	7/2001	Milsted et al.	6,668,246	B1	12/2003	Yeung et al.
6,272,474	B1	8/2001	Garcia	6,668,325	B1	12/2003	Collberg et al.
6,272,535	B1	8/2001	Iwamura	6,674,858	B1	1/2004	Kimura
6,272,634	B1	8/2001	Tewfik et al.	6,674,877	B1	1/2004	Jojic
6,275,988	B1	8/2001	Nagashima et al.	6,687,683	B1	2/2004	Harada et al.
6,278,780	B1	8/2001	Shimada	6,704,451	B1	3/2004	Hekstra
6,278,791	B1	8/2001	Honsinger et al.	6,725,372	B1	4/2004	Lewis et al.
6,282,300	B1	8/2001	Bloom et al.	6,735,702	B1	5/2004	Yavatkar
6,282,650	B1	8/2001	Davis	6,754,822	B1	6/2004	Zhao
6,285,775	B1	9/2001	Wu et al.	6,775,772	B1	8/2004	Binding et al.
6,301,663	B1	10/2001	Kato et al.	6,778,968	B1	8/2004	Gulati
6,310,962	B1	10/2001	Chung et al.	6,784,354	B1	8/2004	Lu et al.
6,317,728	B1	11/2001	Kane	6,785,815	B1	8/2004	Serret-Avila et al.
6,324,649	B1	11/2001	Eyres	6,785,825	B2	8/2004	Colvin
6,330,335	B1	12/2001	Rhoads	6,792,424	B1	9/2004	Burns
6,330,672	B1	12/2001	Shur	6,792,548	B2	9/2004	Colvin
6,345,100	B1	2/2002	Levine	6,792,549	B2	9/2004	Colvin
6,351,765	B1	2/2002	Pietropaolo et al.	6,795,925	B2	9/2004	Colvin
6,363,483	B1	3/2002	Keshav	6,799,277	B2	9/2004	Colvin
				6,804,453	B1	10/2004	Sasamoto
				6,813,717	B2	11/2004	Colvin
				6,813,718	B2	11/2004	Colvin
				6,823,455	B1	11/2004	Macy et al.

(56)

References Cited

U.S. PATENT DOCUMENTS

6,834,308 B1	12/2004	Ikezoey et al.	8,121,343 B2	2/2012	Moskowitz
6,839,686 B1	1/2005	Galant	8,161,286 B2	4/2012	Moskowitz
6,842,862 B2	1/2005	Chow et al.	8,179,846 B2	5/2012	Dolganow
6,853,726 B1	2/2005	Moskowitz et al.	8,214,175 B2	7/2012	Moskowitz
6,856,967 B1	2/2005	Woolston et al.	8,265,278 B2	9/2012	Moskowitz
6,857,078 B2	2/2005	Colvin	8,307,213 B2	11/2012	Moskowitz
6,865,747 B1	3/2005	Mercier	8,400,566 B2	3/2013	Terry et al.
6,876,982 B1	4/2005	Lancaster	8,492,633 B2	7/2013	Ellis
6,931,534 B1	8/2005	Jandel et al.	8,949,619 B2	2/2015	Parry
6,950,941 B1	9/2005	Lee	2001/0010078 A1	7/2001	Moskowitz
6,957,330 B1	10/2005	Hughes	2001/0029580 A1	10/2001	Moskowitz
6,966,002 B1	11/2005	Torrubia-Saez	2001/0043594 A1	11/2001	Ogawa et al.
6,968,337 B2	11/2005	Wold	2002/0009208 A1	1/2002	Alattar
6,977,894 B1	12/2005	Achilles et al.	2002/0010684 A1	1/2002	Moskowitz
6,978,370 B1	12/2005	Kocher	2002/0026343 A1	2/2002	Duenke
6,983,058 B1	1/2006	Fukuoka	2002/0056041 A1	5/2002	Moskowitz
6,986,063 B2	1/2006	Colvin	2002/0057651 A1	5/2002	Roberts
6,990,453 B2	1/2006	Wang	2002/0069174 A1	6/2002	Fox
7,003,480 B2	2/2006	Fox	2002/0071556 A1	6/2002	Moskowitz et al.
7,007,166 B1	2/2006	Moskowitz et al.	2002/0073043 A1	6/2002	Herman et al.
7,020,285 B1	3/2006	Kirovski et al.	2002/0097873 A1	7/2002	Petrovic
7,035,049 B2	4/2006	Yamamoto	2002/0103883 A1	8/2002	Haverstock et al.
7,035,409 B1	4/2006	Moskowitz	2002/0152179 A1	10/2002	Racov
7,043,050 B2	5/2006	Yuval	2002/0161741 A1	10/2002	Wang et al.
7,046,808 B1	5/2006	Metois et al.	2002/0188570 A1	12/2002	Holliman
7,050,396 B1	5/2006	Cohen et al.	2003/0002862 A1	1/2003	Rodriguez
7,051,208 B2	5/2006	Venkatesan et al.	2003/0005780 A1	1/2003	Hansen
7,058,570 B1	6/2006	Yu et al.	2003/0023852 A1	1/2003	Wold
7,093,295 B1	8/2006	Saito	2003/0027549 A1	2/2003	Kiel
7,095,715 B2	8/2006	Buckman	2003/0033321 A1	2/2003	Schrempp
7,095,874 B2	8/2006	Moskowitz et al.	2003/0126445 A1	7/2003	Wehrenberg
7,103,184 B2	9/2006	Jian	2003/0133702 A1	7/2003	Collart
7,107,451 B2	9/2006	Moskowitz	2003/0200439 A1	10/2003	Moskowitz
7,123,718 B1	10/2006	Moskowitz et al.	2003/0219143 A1	11/2003	Moskowitz et al.
7,127,615 B2	10/2006	Moskowitz	2004/0028222 A1	2/2004	Sewell et al.
7,150,003 B2	12/2006	Naumovich et al.	2004/0037449 A1	2/2004	Davis et al.
7,152,162 B2	12/2006	Moskowitz et al.	2004/0049695 A1	3/2004	Choi et al.
7,159,116 B2	1/2007	Moskowitz	2004/0059918 A1	3/2004	Xu
7,162,642 B2	1/2007	Schumann et al.	2004/0083369 A1	4/2004	Erlingsson et al.
7,177,429 B2	2/2007	Moskowitz et al.	2004/0086119 A1	5/2004	Moskowitz
7,177,430 B2	2/2007	Kim	2004/0093521 A1	5/2004	Hamadeh et al.
7,206,649 B2	4/2007	Kirovski et al.	2004/0117628 A1	6/2004	Colvin
7,231,524 B2	6/2007	Burns	2004/0117664 A1	6/2004	Colvin
7,233,669 B2	6/2007	Candelore	2004/0125983 A1	7/2004	Reed et al.
7,240,210 B2	7/2007	Mihcak et al.	2004/0128514 A1	7/2004	Rhoads
7,254,538 B1	8/2007	Ellis	2004/0225894 A1	11/2004	Colvin
7,266,697 B2	9/2007	Kirovski et al.	2004/0243540 A1	12/2004	Moskowitz et al.
7,286,451 B2	10/2007	Wirtz	2005/0135615 A1	6/2005	Moskowitz et al.
7,287,275 B2	10/2007	Moskowitz	2005/0160271 A9	7/2005	Brundage et al.
7,289,643 B2	10/2007	Brunk et al.	2005/0177727 A1	8/2005	Moskowitz et al.
7,310,815 B2	12/2007	Yanovsky	2005/0246554 A1	11/2005	Batson
7,343,492 B2	3/2008	Moskowitz et al.	2006/0005029 A1	1/2006	Petrovic et al.
7,346,472 B1	3/2008	Moskowitz et al.	2006/0013395 A1	1/2006	Brundage et al.
7,362,775 B1	4/2008	Moskowitz	2006/0013451 A1	1/2006	Haitsma
7,363,278 B2	4/2008	Schmelzer et al.	2006/0041753 A1	2/2006	Haitsma
7,409,073 B2	8/2008	Moskowitz et al.	2006/0101269 A1	5/2006	Moskowitz et al.
7,444,506 B1	10/2008	Data	2006/0140403 A1	6/2006	Moskowitz
7,457,962 B2	11/2008	Moskowitz	2006/0251291 A1	11/2006	Rhoads
7,460,994 B2	12/2008	Herre et al.	2006/0285722 A1	12/2006	Moskowitz et al.
7,475,246 B1	1/2009	Moskowitz	2007/0011458 A1	1/2007	Moskowitz
7,530,102 B2	5/2009	Moskowitz	2007/0028113 A1	2/2007	Moskowitz
7,532,725 B2	5/2009	Moskowitz et al.	2007/0064940 A1	3/2007	Moskowitz et al.
7,568,100 B1	7/2009	Moskowitz et al.	2007/0079131 A1	4/2007	Moskowitz et al.
7,630,379 B2	12/2009	Morishita	2007/0083467 A1	4/2007	Lindahl et al.
7,647,502 B2	1/2010	Moskowitz	2007/0110240 A1	5/2007	Moskowitz et al.
7,647,503 B2	1/2010	Moskowitz	2007/0113094 A1	5/2007	Moskowitz et al.
7,664,263 B2	2/2010	Moskowitz	2007/0127717 A1	6/2007	Herre et al.
7,672,838 B1	3/2010	Ellis	2007/0226506 A1	9/2007	Moskowitz
7,672,916 B2	3/2010	Poliner	2007/0253594 A1	11/2007	Lu et al.
7,719,966 B2	5/2010	Luft	2007/0294536 A1	12/2007	Moskowitz et al.
7,743,001 B1	6/2010	Vermeulen	2007/0300072 A1	12/2007	Moskowitz
7,761,712 B2	7/2010	Moskowitz	2007/0300073 A1	12/2007	Moskowitz
7,779,261 B2	8/2010	Moskowitz	2008/0005571 A1	1/2008	Moskowitz
7,812,241 B2	10/2010	Ellis	2008/0005572 A1	1/2008	Moskowitz
8,095,949 B1	1/2012	Hendricks	2008/0016365 A1	1/2008	Moskowitz
			2008/0022113 A1	1/2008	Moskowitz
			2008/0022114 A1	1/2008	Moskowitz
			2008/0028222 A1	1/2008	Moskowitz
			2008/0046742 A1	2/2008	Moskowitz

(56)

References Cited

U.S. PATENT DOCUMENTS

2008/0075277 A1 3/2008 Moskowitz et al.
 2008/0109417 A1 5/2008 Moskowitz
 2008/0133927 A1 6/2008 Moskowitz et al.
 2008/0151934 A1 6/2008 Moskowitz et al.
 2009/0037740 A1 2/2009 Moskowitz
 2009/0089427 A1 4/2009 Moskowitz et al.
 2009/0190754 A1 7/2009 Moskowitz et al.
 2009/0210711 A1 8/2009 Moskowitz
 2009/0220074 A1 9/2009 Moskowitz et al.
 2010/0002904 A1 1/2010 Moskowitz
 2010/0005308 A1 1/2010 Moskowitz
 2010/0064140 A1 3/2010 Moskowitz
 2010/0077219 A1 3/2010 Moskowitz
 2010/0077220 A1 3/2010 Moskowitz
 2010/0098251 A1 4/2010 Moskowitz
 2010/0106736 A1 4/2010 Moskowitz
 2010/0153734 A1 6/2010 Moskowitz
 2010/0182570 A1 7/2010 Matsumoto et al.
 2010/0202607 A1 8/2010 Moskowitz
 2010/0220861 A1 9/2010 Moskowitz
 2010/0313033 A1 12/2010 Moskowitz
 2011/0019691 A1 1/2011 Moskowitz
 2011/0069864 A1 3/2011 Moskowitz
 2011/0128445 A1 6/2011 Carriers
 2012/0057012 A1 3/2012 Sitrack
 2013/0145058 A1 6/2013 Shuholm
 2013/0226957 A1 8/2013 Ellis

FOREIGN PATENT DOCUMENTS

EP 0565947 10/1993
 EP 0565947 A1 10/1993
 EP 0581317 2/1994
 EP 0581317 A2 2/1994
 EP 0649261 4/1995
 EP 0651554 5/1995
 EP 0651554 A 5/1995
 EP 0872073 7/1996
 EP 1547337 3/2006
 EP 1547337 B1 3/2006
 EP 1354276 12/2007
 EP 1354276 B1 12/2007
 NL 100523 9/1998
 NL 1005523 9/1998
 WO WO 95/14289 5/1995
 WO WO 95/14289 5/1995
 WO 96/29795 9/1996
 WO WO 96/29795 9/1996
 WO WO 96/42151 12/1996
 WO WO9701892 1/1997
 WO WO9726733 1/1997
 WO 97/24833 7/1997
 WO WO 97/24833 7/1997
 WO WO9726732 7/1997
 WO WO98002864 7/1997
 WO WO 97/44736 11/1997
 WO WO9802864 1/1998
 WO WO98/37513 8/1998
 WO WO9837513 8/1998
 WO WO 99/52271 10/1999
 WO WO 99/62044 12/1999
 WO WO 99/62044 12/1999
 WO WO 99/63443 12/1999
 WO WO 00/57643 9/2000
 WO WO0118628 3/2001
 WO WO0143026 6/2001
 WO WO0203385 1/2002
 WO WO023385 A1 10/2002
 WO WO0203385 A1 10/2002

OTHER PUBLICATIONS

U.S. Appl. No. 11/894,443, filed Aug. 21, 2007, entitled "Steganographic Method and Device".
 U.S. Appl. No. 11/894,476, filed Aug. 21, 2007, entitled "Steganographic Method and Device".

U.S. Appl. No. 08/674,726, filed Jul. 2, 1996, entitled "Exchange Mechanisms for Digital Information Packages with Bandwidth Securitization, Multichannel Digital Watermarks, and Key Management".

U.S. Appl. No. 11/895,388, filed Aug. 24, 2007, entitled "Data Protection Method and Device".

U.S. Appl. No. 11/900,065, filed Sep. 10, 2007, entitled "Methods, Systems and Devices for Packet Watermarking and Efficient Provisioning of Bandwidth".

U.S. Appl. No. 11/900,066, filed Sep. 10, 2007, entitled "Methods, Systems and Devices for Packet Watermarking and Efficient Provisioning of Bandwidth".

U.S. Appl. No. 11/599,964, filed Nov. 15, 2006, entitled "Optimization Methods for the Insertion, Protection, and Detection of Digital Watermarks in Digital Data".

U.S. Appl. No. 11/897,790, filed Aug. 31, 2007, entitled "Optimization Methods for the Insertion, Protection, and Detection of Digital Watermarks in Digital Data".

U.S. Appl. No. 11/897,791, filed Aug. 31, 2007, entitled "Optimization Methods for the Insertion, Protection, and Detection of Digital Watermarks in Digital Data".

U.S. Appl. No. 11/899,661, filed Sep. 7, 2007, entitled "Optimization Methods for the Insertion, Protection, and Detection of Digital Watermarks in Digital Data".

U.S. Appl. No. 11/899,662, filed Sep. 7, 2007, entitled "Optimization Methods for the Insertion, Protection, and Detection of Digital Watermarks in Digital Data".

U.S. Appl. No. 10/049,101, filed Feb. 8, 2002, entitled "A Secure Personal Content Server" (which claims priority to International Application No. PCT/US00/21189, filed Aug. 4, 2000, which claims priority to U.S. Appl. No. 60/147,134, filed Aug. 4, 1999, and to U.S. Appl. No. 60/213,489, filed Jun. 23, 2000).

U.S. Appl. No. 09/657,181, filed Sep. 7, 2000, entitled "Method and Device for Monitoring and Analyzing Signals".

U.S. Appl. No. 11/518,806, filed Sep. 11, 2006, entitled "Improved Security Based on Subliminal and Supraliminal Channels for Data Object".

PCT Application No. PCT/US95/08159, filed Jun. 26, 1995, entitled, "Digital Information Commodities Exchange with Virtual Menuing".

PCT Application No. PCT/US96/10257, filed Jun. 7, 1996, entitled, "Steganographic Method and Device"—corresponding to—EPO Application No. 96919405.9, entitled "Steganographic Method and Device".

PCT Application No. PCT/US97/00651, filed Jan. 16, 1997, entitled "Method for Stega-Cipher Protection of Computer Code"—corresponding to AU19971829A (not available).

PCT Application No. PCT/US97/00652, filed Jan. 17, 1997, entitled "Method for an Encrypted Digital Watermark"—corresponding to AU199718295A (not available).

PCT Application No. PCT/US97/11455, filed Jul. 2, 1997, entitled, "Optimization Methods for the Insertion, Protection and Detection of Digital Watermarks in Digitized Data"—corresponding to AU199735881A (not available).

PCT Application No. PCT/US99/07262, filed Apr. 2, 1999, entitled, "Multiple Transform Utilization and Applications for Secure Digital Watermarking"—corresponding to—Japan App. No. 2000-542907, entitled "Multiple Transform Utilization and Application for Secure Digital Watermarking"(included herein).

PCT Application No. PCT/US00/06522, filed Mar. 14, 2000, entitled, "Utilizing Data Reduction in Steganographic and Cryptographic Systems".

PCT Application No. PCT/US00/21189, filed Aug. 4, 2000, entitled, "A Secure Personal Content Server".

PCT Application No. PCT/US00/33126, filed Dec. 7, 2000, entitled, "Systems, Methods and Devices for Trusted Transactions"—corresponding to AU200120659A5 (not available).

PCT International Search Report, completed Sep. 13, 1995; authorized officer Huy D. Vu (PCT/US95/08159) (2 pages).

PCT International Search Report, completed Jun. 11, 1996; authorized officer Salvatore Cangialosi (PCT/US96/10257) (4 pages).

Supplementary European Search Report, completed Mar. 5, 2004; authorized officer J. Hazel (EP 96 91 9405) (1 page).

(56)

References Cited

OTHER PUBLICATIONS

- PCT International Search Report, completed Apr. 4, 1997; authorized officer Bernarr Earl Gregory (PCT/US97/00651) (1 page).
- PCT International Search Report completed May 6, 1997; authorized officer Salvatore Cangialosi (PCT/US97/00652) (3 pages).
- PCT International Search Report, completed Oct. 23, 1997; authorized officer David Cain (PCT/US97/11455) (1 page).
- PCT International Search Report, completed Jul. 12, 1999; authorized officer R. Hubeau (PCT/US99/07262) (3 pages).
- Supplementary European Search Report, completed Jun. 27, 2002; authorized officer M. Schoeyer (EP 00 91 9398) (1 page).
- PCT International Search Report, date of mailing Mar. 15, 2001; authorized officer Marja Brouwers (PCT/US00/18411) (5 pages).
- PCT International Search Report, completed Jul. 20, 2001; authorized officer A. Sigolo (PCT/US00/18411) (5 pages).
- PCT International Search Report, completed Mar. 20, 2001; authorized officer P. Corcoran (PCT/US00/33126) (6 pages).
- PCT International Search Report, completed Jan. 26, 2001; authorized officer Gilberto Barron (PCT/US00/21189) (3 pages).
- Schneier, Bruce, *Applied Cryptography*, 2nd Ed., John Wiley & Sons, pp. 9-10, 1996.
- Menezes, Alfred J., *Handbook of Applied Cryptography*, CRC Press, p. 46, 1997.
- Merriam-Webster's Collegiate Dictionary, 10th Ed., Merriam Webster, Inc., p. 207.
- Breal, et al., Principles of Corporate Finance, "Appendix A—Using Option Valuation Models", 1984, pp. 448-449.
- Copeland, et al., *Real Options: A Practitioner's Guide*, 2001 pp. 106-107, 201-202, 204-208.
- Low, S.H., "Equilibrium Allocation and Pricing of Variable Resources Among User-Suppliers", 1988. <http://www.citesear.nj.nec.com/366503.html>.
- Gruhl, Daniel et al., Echo Hiding. In Proceeding of the Workshop on Information Hiding, No. 1174 in Lecture Notes in Computer Science, Cambridge, England (May/June 1996).
- Oomen, A.W.J. et al., A Variable Bit Rate Buried Data Channel for Compact Disc, *J. Audio Eng. Soc.*, vol. 43, No. 1/2, pp. 23-28 (1995).
- Ten Kate, W. et al., A New Surround-Stereo-Surround Coding Techniques, *J. Audio Eng. Soc.*, vol. 40, No. 5, pp. 376-383 (1992).
- Sklar, Bernard, *Digital Communications*, pp. 601-603 (1988).
- Jayant, N.S. et al., *Digital Coding of Waveforms*, Prentice Hall Inc., Englewood Cliffs, NJ, pp. 486-509 (1984).
- ten Kate, W. et al., "Digital Audio Carrying Extra Information", IEEE, CH 2847-2/90/0000-1097, (1990).
- van Schyndel, et al. A digital Watermark, IEEE Int'l Computer Processing Conference, Austin, TX, Nov. 13-16, 1994, pp. 86-90.
- Smith, et al. Modulation and Information Hiding in Images, Springer Verlag, 1st Int'l Workshop, Cambridge, UK, May 30-Jun. 1, 1996, pp. 207-227.
- Kutter, Martin et al., Digital Signature of Color Images Using Amplitude Modulation, SPIE-E197, vol. 3022, pp. 518-527.
- Puate, Joan et al., Using Fractal Compression Scheme to Embed a Digital Signature into an Image, SPIE-96 Proceedings, vol. 2915, Mar. 1997, pp. 108-118.
- Swanson, Mitchell D. et al., Transparent Robust Image Watermarking, Proc. of the 1996 IEEE Int'l Conf. on Image Processing, vol. 111, 1996, pp. 211-214.
- Swanson, Mitchell D., et al. Robust Data Hiding for Images, 7th IEEE Digital Signal Processing Workshop, Leon, Norway, Sep. 1-4, 1996, pp. 37-40.
- Zhao, Jian et al., Embedding Robust Labels into Images for Copyright Protection, Proceeding of the Know Right '95 Conference, pp. 242-251.
- Van Schyndel, et al., Towards a Robust Digital Watermark, Second Asian Image Processing Conference, Dec. 6-8, 1995, Singapore, vol. 2, pp. 504-508.
- Tirkel, A.Z., A Two-Dimensional Digital Watermark, DICTA '95, Univ. of Queensland, Brisbane, Dec. 5-8, 1995, pp. 7.
- Tirkel, A.Z., Image Watermarking—A Spread Spectrum Application, ISSSTA '96, Sep. 1996, Mainz, Germany, pp. 6.
- O'Ruanaidh, et al. Watermarking Digital Images for Copyright Protection, IEEE Proceedings, vol. 143, No. 4, Aug. 1996, pp. 250-256.
- Cox, et al., Secure Spread Spectrum Watermarking for Multimedia, NEC Research Institute, Technical Report 95-10, pp. 33.
- Kahn, D., *The Code Breakers*, The MacMillan Company, 1969, pp. xiii, 81-83, 513, 515, 522-526, 863.
- Boney, et al., Digital Watermarks for Audio Signals, EVSIPCO, 96, pp. 473-480.
- F. Hartung, et al., Digital Watermarking of Raw and Compressed Video, SPIE vol. 2952, pp. 205-213.
- Craver, et al., Can Invisible Watermarks Resolve Rightful Ownership? IBM Research Report, RC 20509 (Jul. 25, 1996) 21 pp.
- Press, et al., *Numerical Recipes in C*, Cambridge Univ. Press, 1988, pp. 398-417.
- Pohlmann, Ken C., *Principles of Digital Audio*, 3rd Ed., 1995, pp. 32-37, 40-48, 138, 147-149, 332, 333, 364, 499-501, 508-509, 564-571.
- Pohlmann, Ken C., *Principles of Digital Audio*, 2nd Ed., 1991, pp. 1-9, 19-25, 30-33, 41-48, 54-57, 86-107, 375-387.
- Schneier, Bruce, *Applied Cryptography*, John Wiley & Sons, inc., New York, 1994, pp. 68, 69, 387-392, 1-57, 273-275, 321-324.
- Bender, et al., Techniques for Data Hiding, IBM Systems Journal, vol. 35, No. 3 & 4, 1996, pp. 313-336.
- Moskowitz, Bandwidth as Currency, IEEE Multimedia, Jan.-Mar. 2003, pp. 14-21.
- Steinauer D. D., et al., "Trust and Traceability in Electronic Commerce", Standard View, Sep. 1997, pp. 118-124, vol. 5 No. 3, ACM, USA.
- Rivest, et al., PayWord and MicroMint: Two simple micropayment schemes, MIT Laboratory for Computer Science, Cambridge, MA 02139, Apr. 27 2001, pp. 1-18.
- Horowitz, et al., *The Art of Electronics*, 2nd Ed., 1989, pp. 7.
- Delaigle, J.-F., et al. "Digital Watermarking." Proceedings of the SPIE, vol. 2659, Feb 1, 1996, pp. 99-110 (Abstract).
- Cox, I.J., et al. "Secure Spread Spectrum Watermarking for Multimedia," IEEE Transactions on Image Processing, vol. 6, No. 12, Dec. 1, 1997, pp. 1673-1686.
- Wong, Ping Wah. "A Public Key Watermark for Image Verification and Authentication," IEEE International Conference on Image Processing, vol. 1, Oct. 4-7, 1998, pp. 455-459.
- Ross Anderson, "Stretching the Limits of Steganography," LNCS, vol. 1174, May/June 1996, 10 pages, ISBN: 3-540-61996-8.
- European Search Report, completed Oct. 15, 2007; authorized officer James Hazel (EP 07 11 2420) (9 pages).
- Arctic Monkeys (Whatever People Say I Am, That's What I'm Not), Domino Recording Co. Ltd., Pre-Release CD image, 2005, 1 page.
- Radiohead ("Hail To The Thief"), EMI Music Group—Capitol, Pre-Release CD image, 2003, 1 page.
- Oasis (Dig Out Your Soul), Big Brother Recordings Ltd., Promotion CD image, 2009, 1 page.
- Rivest, R. "Charring and Winnowing: Confidentiality without Encryption", MIT Lab for Computer Science, <http://people.csail.mit.edu/rivest/Chaffing.txt>, Apr. 24, 1998, 9 pp.
- VeriDisc, "The search for a Rational Solution to Digital Rights Management (DRM)", http://64.244.235.240/news/whitepaper/docs/veridisc_white_paper.pdf, 2001, 15 pp.
- Cayre, et al., "Kerckhoffs-Based Embedding Security Classes for WOA Data Hiding". IEEE Transactions on Information Forensics and Security, vol. 3, No. 1, Mar. 2008, 15 pp.
- Namgoong, H. "An Integrated Approach to Legacy Data for Multimedia Applications", Proceedings of the 23rd EUROMICRO Conference, Vol., Issue 1-4, Sep. 1997, pp. 387-391.
- Wayback Machine, dated Aug. 26, 2007, <http://web.archive.org/web/20070826151732/http://www.screenplaysmag.com/tabid/96/articleType/ArticleView/articleId/495/Default.aspx/>.
- EPO Application No. 96919405.9, entitled "Steganographic Method and Device"; published as EP0872073 (A2), published Oct. 21, 1998.
- Jap. App. No. 2000-542907, entitled "Multiple Transform Utilization and Application for Secure Digital Watermarking", JP national stage of PCT/US1999/007262, published as WO99052271, Oct. 14, 1999.
- PCT Application No. PCT/US00/21189, filed Aug. 4, 2000, entitled, "A Secure Personal Content Server", Pub. No. WO018628; Publication Date: Mar. 15, 2001.

(56)

References Cited

OTHER PUBLICATIONS

- Merriam-Webster's Collegiate Dictionary, 10th Ed., Merriam Webster, Inc., p. 207, 1997.
- Sarkar, M. "An Assessment of Pricing Mechanisms for the Internet—A Regulatory Imperative", presented MIT Workshop on Internet Economics, Mar. 1995, <http://www.press.vmich.edu/iep/works/SarkAsses.html> on.
- Crawford, D.W. "Pricing Network Usage: A Market for Bandwidth of Market Communication?" presented MIT Workshop on Internet Economics, Mar. 1995 <http://www.press.vmich.edu/iep/works/CrawMarket.html> on March.
- Kutter, Martin et al., "Digital Signature of Color Images Using Amplitude Modulation", SPIE-E197, vol. 3022, pp. 518-527, 1997.
- U.S. Appl. No. 09/671,739, filed Sep. 29, 2000, entitled "Method and Device for Monitoring and Analyzing Signals", abandoned.
- PCT International Search Report, completed Jun. 30, 2000; authorized officer Paul E. Callahan (PCT/US00/06522) (7 pages).
- Rivest, R. "Chaffing and Winnowing: Confidentiality without Encryption", MIT Lab for Computer Science, <http://people.csail.mit.edu/rivest/Chaffing.txt> Apr. 24, 1998, 9 pages.
- PortalPlayer, PP502 digital media management system-on-chip, May 1, 2003, 4 pp.
- Wayback Machine, dated Jan. 17, 1999, <http://web.archive.org/web/19990117020420/http://www.netzero.com/>, accessed on Feb. 19, 2008.
- "YouTube Copyright Policy: Video Identification tool—YouTube Help", accessed Jun. 4, 2009, <http://www.google.com/support/youtube/bin/answer.py?hl=en&answer=83766>, 3 pp.
- PCT Application No. PCT/US00/18411, filed Jul. 5, 2000, entitled, "Copy Protection of Digital Data Combining Steganographic and Cryptographic Techniques"—corresponding to AU200060709A5 (not available).
- EPO Divisional Patent Application No. 07112420.0, entitled "Steganographic Method and Device" corresponding to PCT Application No. PCT/US96/10257.
- EPO Application No. 96919405.9, entitled "Steganographic Method and Device"; published as EP0872073 (A2), Oct. 21, 1998.
- Jap. App. No. 2000-542907, entitled "Multiple Transform Utilization and Application for Secure Digital Watermarking"; which is a JP national stage of PCT/US1999/007262, published as WO/1999/052271, Oct. 14, 1999.
- PCT Application No. PCT/US00/21189, filed Aug. 4, 2000, entitled, "A Secure Personal Content Server", Pub. No. WO/2001/018628; Publication Date: Mar. 15, 2001.
- Menezes, Alfred J., Handbook of Applied Cryptography, CRC Press, p. 46, 1997.
- Merriam-Webster's Collegiate Dictionary, 10th Ed., Merriam Webster, Inc., p. 207, 1997.
- Copeland, et al., Real Options: A Practitioner's Guide, 2001 pp. 106-107, 201-202, 204-208.
- Sarkar, M. "An Assessment of Pricing Mechanisms for the Internet—A Regulatory Imperative", presented MIT Workshop on Internet Economics, Mar. 1995 <http://www.press.vmich.edu/iep/works/SarkAsses.html> on.
- Crawford, D.W. "Pricing Network Usage: A Market for Bandwidth of Market Communication?" presented MIT Workshop on Internet Economics, Mar. 1995 <http://www.press.vmich.edu/iep/works/CrawMarket.html> on March.
- Caronni, Germano, "Assuring Ownership Rights for Digital Images", published proceeds of reliable IT systems, v15 '95, H.H. Bruggemann and W. Gerhardt-Hackel (Ed) Viewing Publishing Company Germany 1995.
- Zhao, Jian. "A WWW Service to Embed and Prove Digital Copyright Watermarks", Proc. of the European conf. on Multimedia Applications, Services & Techniques Louvain-La-Neuve Belgium May 1996.
- Gruhl, Daniel et al., Echo Hiding. In Proceeding of the Workshop on Information Hiding. No. 1174 in Lecture Notes in Computer Science, Cambridge, England (May/June. 1996).
- Oomen, A.W.J. et al., A Variable Bit Rate Buried Data Channel for Compact Disc, J.AudioEng. Sc., vol. 43, No. 1/2, pp. 23-28 (1995).
- Ten Kate, W. et al., A New Surround-Stereo-Surround Coding Techniques, J. Audio Eng.Soc., vol. 40, No. 5, pp. 376-383 (1992).
- Gerzon, Michael et al., A High Rate Buried Data Channel for Audio CD, presentation notes, Audio Engineering Soc. 94th Convention (1993).
- Sklar, Bernard, Digital Communications, pp. 601-603 (1988).
- Jayant, N.S. et al., Digital Coding of Waveforms, Prentice Hall Inc., Englewood Cliffs, NJ, pp. 486-509 (1984).
- Bender, Walter R. et al., Techniques for Data Hiding, SPIE Int. Soc. Opt. Eng., vol. 2420, pp. 164-173, 1995.
- Zhao, Jian et al., Embedding Robust Labels into Images for Copyright Protection, (xp 000571976), pp. 242-251, 1995.
- Menezes, Alfred J., Handbook of Applied Cryptography, CRC Press, p. 175, 1997.
- Schneier, Bruce, Applied Cryptography, 1st Ed., pp. 67-68, 1994.
- Van Schyndel, et al., "A digital Watermark," IEEE Int'l Computer Processing Conference, Austin, TX, Nov. 13-16, 1994, pp. 86-90.
- Smith, et al. "Modulation and Information Hiding in Images", Springer Verlag, 1st Int'l Workshop, Cambridge, UK, May 30-Jun. 1, 1996, pp. 207-227.
- Kutter, Martin et al., "Digital Signature of Color Images Using Amplitude Modulation", SPIE-E197, vol. 3022, pp. 518-527.
- Puate, Joan et al., "Using Fractal Compression Scheme to Embed a Digital Signature into an Image", SPIE-96 Proceedings, vol. 2915, Mar. 1997, pp. 108-118.
- Swanson, Mitchell D., et al., "Transparent Robust Image Watermarking", Proc. of the 1996 IEEE Int'l Conf. on Image Processing, vol. 111, 1996, pp. 211-214.
- Swanson, Mitchell D., et al. "Robust Data Hiding for Images", 7th IEEE Digital Signal Processing Workshop, Leon, Norway, Sep. 1-4, 1996, pp. 37-40.
- Zhao, Jian et al., "Embedding Robust Labels into Images for Copyright Protection", Proceeding of the Know Right '95 Conference, pp. 242-251.
- Koch, E., et al., "Towards Robust and Hidden Image Copyright Labeling", 1995 IEEE Workshop on Nonlinear Signal and Image Processing, Jun. 1995 Neos Marmaras pp. 4.
- Van Schyndel, et al., "Towards a Robust Digital Watermark", Second Asian Image Processing Conference, Dec. 6-8, 1995, Singapore, vol. 2, pp. 504-508.
- Tirkel, A.Z., "A Two-Dimensional Digital Watermark", DICTA '95, Univ. of Queensland, Brisbane, Dec. 5-8, 1995, pp. 7.
- Tirkel, A.Z., "Image Watermarking—A Spread Spectrum Application", ISSSTA '96, Sep. 1996, Mainz, Germany, pp. 6.
- O'Ruanaidh, et al. "Watermarking Digital Images for Copyright Protection", IEEE Proceedings, vol. 143, No. 4, Aug. 1996, pp. 250-256.
- Kahn, D., "The Code Breakers", The MacMillan Company, 1969, pp. xlii, 81-83, 513, 515, 522-526, 863.
- Boney, et al., Digital Watermarks for Audio Signals, EVSIPCO, 96, pp. 473-480 (Mar. 14, 1997).
- Dept. of Electrical Engineering, Delft University of Technology, Delft the Netherlands, Cr.C. Langelaar et al., "Copy Protection for Multimedia Data based on Labeling Techniques", Jul. 1996 9 pp.
- F. Hartung, et al., "Digital Watermarking of Raw and Compressed Video", SPIE vol. 2952, pp. 205-213.
- Craver, et al., "Can Invisible Watermarks Resolve Rightful Ownerships?", IBM Research Report, RC 20509 (Jul. 25, 1996) 21 pp.
- Press, et al., "Numerical Recipes in C", Cambridge Univ. Press, 1988, pp. 398-417.
- Pohlmann, Ken C., "Principles of Digital Audio", 3rd Ed., 1995, pp. 32-37, 40-48:138, 147-149, 332, 333, 364, 499-501, 508-509, 564-571.
- Pohlmann, Ken C., "Principles of Digital Audio", 2nd Ed., 1991, pp. 1-9, 19-25, 30-33, 41-48, 54-57, 86-107, 375-387.
- Schneier, Bruce, Applied Cryptography, John Wiley & Sons, Inc., New York, 1994, pp. 68, 69, 387-392, 1-57, 273-275, 321-324.
- Boney, et al., Digital Watermarks for Audio Signals, Proceedings of the International Conf. on Multimedia Computing and Systems, Jun. 17-23, 1996 Hiroshima, Japan, 0-8186-7436-9/96, pp. 473-480.

(56)

References Cited

OTHER PUBLICATIONS

- Johnson, et al., "Transform Permuted Watermarking for Copyright Protection of Digital Video", IEEE Globecom 1998, Nov. 8-12, 1998, New York New York vol. 2 1998 pp. 684-689 (ISBN 0-7803-4985-7).
- Rivest, et al., "Pay Word and Micromint: Two Simple Micropayment Schemes," MIT Laboratory for Computer Science, Cambridge, MA, May 7, 1996 pp. 1-18.
- Bender, et al., "Techniques for Data Hiding", IBM Systems Journal, (1996) vol. 35, No. 3 & 4, 1996, pp. 313-336.
- Moskowitz, "Bandwidth as Currency", IEEE Multimedia, Jan.-Mar. 2003, pp. 14-21.
- Moskowitz, Multimedia Security Technologies for Digital Rights Management, 2006, Academic Press, "Introduction—Digital Rights Management" pp. 3-22.
- Rivest, et al., "PayWord and Micromint: Two Simple Micropayment Schemes," MIT Laboratory for Computer Science, Cambridge, MA, Apr. 27, 2001, pp. 1-18.
- Tomsich, et al., "Towards a secure and de-centralized digital watermarking infrastructure for the protection of Intellectual Property", in Electronic Commerce and Web Technologies, Proceedings (FCWEB)(2000).
- Moskowitz, "What is Acceptable Quality in the Application of Digital Watermarking: Trade-offs of Security, Robustness and Quality", IEEE Computer Society Proceedings of ITCC 2002 Apr. 10, 2002 pp. 80-84.
- Lemma, et al. "Secure Watermark Embedding through Partial Encryption, International Workshop on Digital Watermarking" ("IWDW" 2006). Springer Lecture Notes in Computer Science 2006 (to appear) 13.
- Koehler, et al., "Self Protecting Digital Content", Technical Report from the CRI Content Security Research Initiative, Cryptography Research, Inc. 2002-2003 14 pages.
- Sirbu, M. et al., "Net Bill: An Internet Commerce System Optimized for Network Delivered Services", Digest of Papers of the Computer Society Computer Conference (Spring) Mar. 5, 1995 pp. 20-25 vol. CONF40.
- Schunter, M. et al., "A Status Report on the SEMPER framework for Secure Electronic Commerce", Computer Networks and ISDN Systems, Sep. 30, 1998, pp. 1501-1510 vol. 30 No. 16-18 NL North Holland.
- Konrad, K. et al., "Trust and Electronic Commerce—more than a technical problem," Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems Oct. 19-22, 1999, pp. 360-365 Lausanne.
- Kini, et al., "Trust in Electronic Commerce: Definition and Theoretical Considerations", Proceedings of the 31st Hawaii Int'l Conf on System Sciences (Cat. No. 98TB100216). Jan. 6-9, 1998, pp. 51-61. Los.
- Hartung, et al. "Multimedia Watermarking Techniques", Proceedings of the IEEE, Special Issue, Identification & Protection of Multimedia Information, pp. 1079-1107 Jul. 1999 vol. 87 No. 7 IEEE.
- European Search Report & European Search Opinion in EP07112420.
- STAIN'D (The Singles 1996-2006), Warner Music—Atlantic, Pre-Release CD image, 2006, 1 page.
- Radiohead ("Hail to the Thief"), EMI Music Group—Capitol, Pre-Release CD image, 2003, 1 page.
- U.S. Appl. No. 60/169,274, filed Dec. 7, 1999, entitled "Systems, Methods and Devices for Trusted Transactions".
- U.S. Appl. No. 60/234,199, filed Sep. 20, 2000, "Improved Security Based on Subliminal and Supraliminal Channels for Data Objects".
- U.S. Appl. No. 09/671,739, filed Sep. 29, 2000, entitled "Method and Device for Monitoring and Analyzing Signals".
- Tirkel, A.Z., "A Two-Dimensional Digital Watermark", Scientific Technology, 686, 14, date unknown.
- EP0581317A2.
- PCT International Search Report in PCT/US95/08159.
- PCT International Search Report in PCT/US96/10257.
- Supplementary European Search Report in EP 96919405.
- PCT International Search Report in PCT/US97/00651.
- PCT International Search Report in PCT/US97/00652.
- PCT International Search Report in PCT/US97/11455.
- PCT International Search Report in PCT/US99/07262.
- PCT International Search Report in PCT/US00/06522.
- Supplementary European Search Report in EP00919398.
- PCT International Search Report in PCT/US00/18411.
- PCT International Search Report in PCT/US00/33126.
- PCT International Search Report in PCT/US00/21189.
- Delaigle, J.-F., et al. "Digital Watermarking," Proceedings of the SPIE, vol. 2659, Feb 1, 1996, pp. 99-110.
- Schneider, M., et al. "A Robust Content Based Digital Signature for Image Authentication," Proceedings of the International Conference on Image Processing (IC. Lausanne) Sep. 16-19, 1996, pp. 227-230, IEEE ISBN.
- Cox, I. J., et al. "Secure Spread Spectrum Watermarking for Multimedia," IEEE Transactions on Image Processing, vol. 6 No. 12, Dec. 1, 1997, pp. 1673-1686.
- Wong, Ping Wah. "A Public Key Watermark for Image Verification and Authentication," IEEE International Conference on Image Processing, vol. 1 Oct. 4-7, 1998, pp. 455-459.
- Fabien A.P. Petitcolas, Ross J. Anderson and Markkus G. Kuhn, "Attacks on Copyright Marking Systems," LNCS, vol. 1525, Apr. 14-17, 1998, pp. 218-238 ISBN: 3-540-65386-4.
- Joseph J.K. O'Ruanaidh and Thierry Pun, "Rotation, Scale and Translation Invariant Digital Image Watermarking", pre-publication, Summer 1997 4 pages.
- Joseph J.K. O'Ruanaidh and Thierry Pun, "Rotation, Scale and Translation Invariant Digital Image Watermarking", Submitted to Signal Processing Aug. 21, 1997, 19 pages.
- Oasis (Dig Out Your Soul), Big Brother Recordings Ltd, Promotional CD image, 2008, 1 page.
- Rivest, R. "Chaffing and Winnowing: Confidentiality without Encryption", MIT Lab for Computer Science, <http://people.csail.mit.edu/rivest/Chaffing.txt> Apr. 24, 1998, 9 pp.
- PortalPlayer, PP5002 digital media management system-on-chip, May 1, 2003, 4 pp.
- VeriDisc, "The Search for a Rational Solution to Digital Rights Management (DRM)", <http://64.244.235.240/news/whitepaper/docs/veridisc.sub.--white.sub.--paper.pdf>, 2001, 15 pp.
- Cayre, et al., "Kerckhoff's-Based Embedding Security Classes for WOA Data Hiding", IEEE Transactions on Information Forensics and Security, vol. 3 No. 1, Mar. 2008, 15 pp.
- Wayback Machine, dated Jan. 17, 1999, <http://web.archive.org/web/19990117020420/http://www.net-zero.com/>, accessed on Feb. 19, 2008.
- Namgoong, H., "An Integrated Approach to Legacy Data for Multimedia Applications", Proceedings of the 23rd EUROMICRO Conference, vol. 1, Issue 1-4, Sep. 1997, pp. 387-391.
- Wayback Machine, dated Aug. 26, 2007, <http://web.archive.org/web/20070826151732/http://www.screenplaysmag.com/t-abid/96/articleType/ArticleView/articleId/495/Default.aspx/>.
- "YouTube Copyright Policy: Video Identification tool—YouTube Help", accessed Jun. 4, 2009, <http://www.google.com/support/youtube/bin/answer.py?hl=en&answer=83766>, 3 pp.
- PCT Application No. PCT/US95/08159, filed Jun. 26, 1995, entitled, "Digital Information Commodities Exchange with Virtual Menuing", published as WO/1997/001892; Publication Date: Jan. 16, 1997.
- PCT Application No. PCT/US96/10257, filed Jun. 7, 1996, entitled "Steganographic Method and Device"—corresponding to—EPO Application No. 96919405.9, entitled "Steganographic Method and Device", published as WO/1996/042151; Publication Date: Dec. 27, 1996.
- PCT Application No. PCT/US97/00651, filed Jan. 16, 1997, entitled, "Method for Stega-Cipher Protection of Computer Code", published as WO/1997/026732; Publication Date: Jul. 24, 1997.
- PCT Application No. PCT/US97/00652, filed Jan. 17, 1997, entitled, "Method for an Encrypted Digital Watermark", published as WO/1997/026733; Publication Date: Jul. 24, 1997.
- PCT Application No. PCT/US97/11455, filed Jul. 2, 1997, entitled, "Optimization Methods for the Insertion, Protection and Detection of Digital Watermarks in Digitized Data", published as WO/1998/002864; Publication Date: Jan. 22, 1998.

(56)

References Cited

OTHER PUBLICATIONS

- PCT Application No. PCT/US99/07262, filed Apr. 2, 1999, entitled, "Multiple Transform Utilization and Applications for Secure Digital Watermarking", published as WO/1999/052271; Publication Date: Oct. 14, 1999.
- PCT Application No. PCT/US00/06522, filed Mar. 14, 2000, entitled, "Utilizing Data Reduction in Steganographic and Cryptographic Systems", published as WO/2000/057643; Publication Date: Sep. 28, 2000.
- PCT Application No. PCT/US00/18411, filed Jul. 5, 2000, entitled, "Copy Protection of Digital Data Combining Steganographic and Cryptographic Techniques".
- PCT Application No. PCT/US00/33126, filed Dec. 7, 2000, entitled "Systems, Methods and Devices for Trusted Transactions", published as WO/2001/043026; Publication Date: Jun. 14, 2001.
- FPO Divisional Patent Application No. 07112420.0, entitled "Steganographic Method and Device" corresponding to PCT Application No. PCT/US96/10257, published as WO/1996/042151, Dec. 27, 1996.
- U.S. Appl. No. 60/222,023 filed Jul. 31, 2007 entitled "Method and apparatus for recognizing sound and signals in high noise and distortion".
- "Techniques for Data Hiding in Audio Files," by Morimoto, 1995.
- Howe, Dennis Jul. 13, 1998 <http://foldoc.org/steganography>.
- CSG, Computer Support Group and CSGNetwork.com 1973 <http://www.csgnetwork.com/glossarys.html>.
- QuinStreet Inc. 2010 What is steganography?—A word definition from the Webopedia Computer Dictionary <http://www.webopedia.com/terms/steganography.html>.
- Graham, Robert Aug. 21, 2000 "Hacking Lexicon" <http://robertgraham.com/pubs/hacking-dict.html>.
- Farkey, Inc 2010 "Steganography definition of steganography in the Free Online Encyclopedia" <http://encyclopedia2.thefreedictionary.com/steganography>.
- Horowitz, et al., The Art of Electronics. 2nd Ed., 1989, pp. 7.
- Jimmy eat world ("futures"), Interscope Records, Pre-Release CD image, 2004, 1 page.
- Aerosmith ("Just Push Play"), Pre-Release CD image, 2001, 1 page.
- Phil Collins(Testify) Atlantic, Pre-Release CD image, 2002, 1 page.
- U. are U. Reviewer's Guide (U are U Software, 1998).
- U. are U. wins top honors! —Marketing Flyer (U. are U. Software, 1998).
- Digital Persona, Inc., *U. are U. Fingerprint Recognition System: User Guide* (Version 1.0, 1998).
- Digital Persona White Paper pp. 8-9 published Apr. 15, 1998.
- Digital Persona, Inc., "Digital Persona Releases U. are U Pro Fingerprint Security Systems for Windows NT, 2000, '98, '95", (Feb. 2000).
- SonicWall, Inc. 2011 "The Network Security SonicOS Platform-Deep Packet Inspection" http://www.sonicwall.com/us/en/products/Deep_Packet_Inspection.html.
- Rick Merritt, PARC hosts summit on content-centric nets, EETimes, Aug. 12, 2011, <http://www.eetimes.com/electronics-news/4218741/PARC-hosts-summit-on-content-centric-nets>.
- Afanasyev, et. al., Communications of the ACM: Privacy Preserving Network Forensics 2011.
- SonicWall, Inc., 2008 "The Advantages of a Multi-core Architecture in Network Security Appliances" http://www.sonicwall.com/downloads/WP-ENG-010_Multicore.
- Voip-Pal.Com Inc's Lawful Intercept Patent Application Receives the Allowance for Issuance as a Patent, <http://finance.yahoo.com/news/voip-pal-com-inc-lawful-133000133.html>.
- Deep Content Inspection—Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/Deep_content_inspection (last visited Apr. 4, 2013).
- Dexter, et. al, "Multi-view Synchronization of Human Actions and Dynamic Scenes" pp. 1-11, 2009.
- Kudrle, et al., "Fingerprinting for Solving A/V Synchronization Issues within Broadcast Environments", 2011.
- Junego, et. al., "View-Independent Action Recognition from Temporal Self-Similarities", 2011.
- Dexter, et al., "Multi-view Synchronization of Image Sequences", 2009.
- Blue Spike, LLC v. Texas Instruments, Inc et. al.*, (No: 6:12-CV-499-MHS), Audible Magic Corporations's amended Answer (E.D. TX filed Jul. 15, 2013) (Document 885 page ID 9581), (PACER).
- Moskowitz, "Introduction-Digital Rights Management," Multimedia Security Technologies for Digital Rights Management (2006), Elsevier.
- George, Mercy; Chouinard, Jean-Yves; Georgana, Nicolas. Digital Watermarking of Images and video using Direct Sequence Spread Spectrum Techniques. 1999 IEEE Canadian Conference on Electrical and Computer Engineering vol. 1. Pub. Date: 1999 Relevant pp. 116-121. <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=arnumber=807181>.
- Apr. 4, 2014, Shazam Entertainment Limited's Amended Answer to Blue Spike, LLC's complaint and counterclaims against Blue Spike LLC, Blue Spike, Inc and Scott A. Moskowitz, *Shazam Entertainment Ltd v. Blue Spike, LLC, Blue Spike, Inc, and Scott Moskowitz* (E.D.TX Dist Ct.) Case No. 6:12-CV-00499-MHS.
- Apr. 4, 2014, Audible Magic Corporation's Second Amended Answer to Blue Spike LLC's Original Complaint for patent infringement and counterclaims against Blue Spike LLC, Blue Spike, Inc and Scott Moskowitz, *Blue Spike LLC v. Texas Instruments, Audible Magic Corporation* (E.D.TX Dist Ct.) Case No. 6:12-CV-499-MHS.
- Dec. 19, 2011, Shrivastava, et al., "Data-Driven Visual Similarity for Cross-Domain Image Matching", 2011 ACM Transaction of Graphics (TOG), ACM SIGGRAPH Asia vol. 30 No. 6, <http://graphics.cs.cmu.edu/projects/crossDomainMatching/>.
- Spice, Byron, "Carnegie Mellon Researchers Develop Computerized Method for Finding Similar Images in Photos, Paintings, Sketches", Carnegie Mellon News, Dec. 6, 2011, Carnegie Mellon University, http://www.cmu.edu/news/stories/archives/2011/december/dec6_matchingimages.html.
- Oct. 16, 2014, Memorandum Opinion and Order, *Blue Spike LLC v. Texas Instruments, Inc. et al.*, (E.D.TX Dist Ct), Case No. 6:12CV-0499-MHS-CMC.
- Yu, Che-Fn, "Access Control and Authorization Plan for Customer Control of Network Services", IEEE Globecom 1989 Pub 1989, pp. 862-869. <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=64085>.
- Jaeger, Trent; Prakash, Atul; Rubin, Avid D, "A System Architecture for Flexible Control of Downloaded Executable Content." Proceedings of the Fifth International Workshop on Object-Oriented in Operating Systems. Pub 1996, pp. 14-18. <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=557855>.
- "Activate Your Product Through the Online License Management System (LMS)", May 2011 Juniper Networks, Inc., USA.
- "Activate Your Software Capacity and/or Features", May 2011, Juniper Networks, USA.
- "Download and Activate Your Software", May 2011, Juniper Networks, Inc., USA.
- "Electronic Fulfillment of Feature, Capacity and Subscription License Activation Keys via the License Management System (LMS)", Sep. 2009, Juniper Networks, Inc., USA.
- "Juniper Networks License Management System (LMS) FAQ", Jul. 2009, Juniper Networks, Inc., USA.
- "License Activation Keys", Dec. 14, 2014, http://www.juniper.net/generate_license/.
- "License code and configuration key reference [AX 2012]", Mar. 25, 2014, Microsoft <http://technet.microsoft.com/en-us/library/hh378074.aspx>.
- "License Codes", Dec. 14, 2014, Oracle <http://www.oracle.com/us/support/licensescodes/index.html>.
- "PeopleSoft Enterprise: License Codes", Dec. 14, 2014, <http://www.oracle.com/us/support/licensescodes/peoplesoft-enterprise/index.html>.
- "Primavera License Key Files", Dec. 14, 2014, <http://www.oracle.com/us/support/licensescodes/primavera/index.html>.
- "Siebel License Keys", Dec. 14, 2014, <http://www.oracle.com/us/support/licensescodes/siebel/index.html>.

(56)

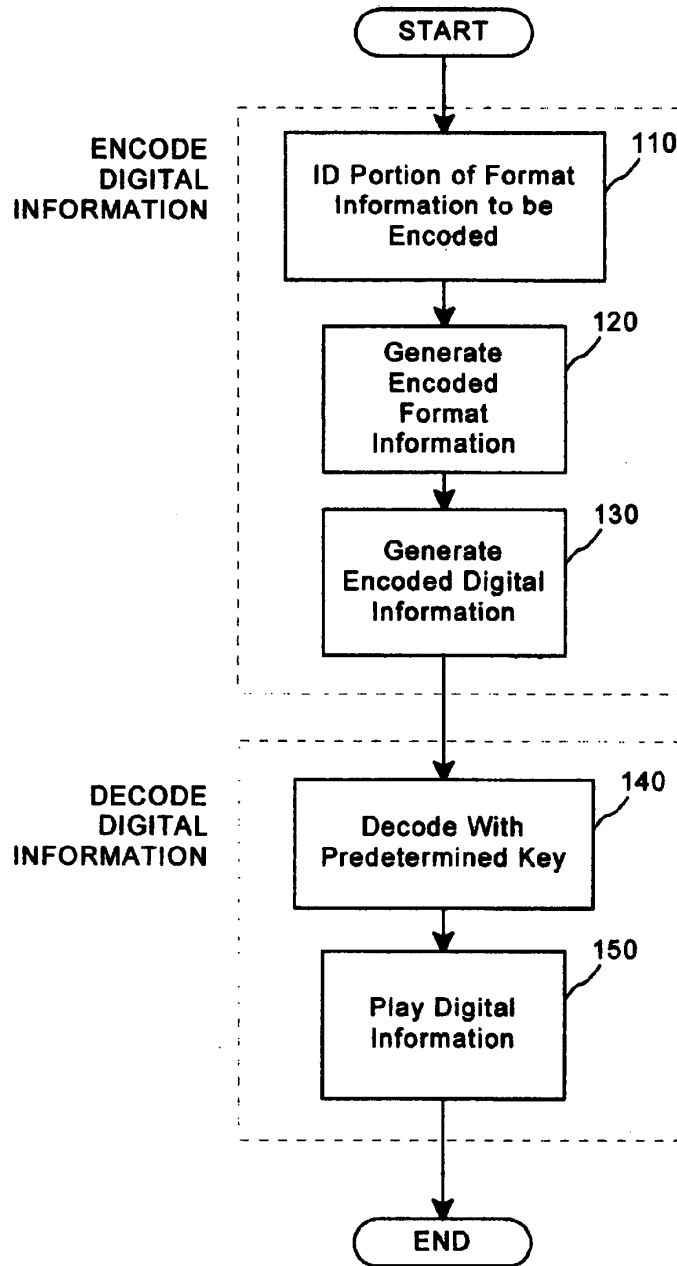
References Cited

OTHER PUBLICATIONS

"How to transfer a license activation key to an RMA replacement device". Mar. 2009, Juniper Networks, Inc. USA.
"How to register a license key in My VMware (2011177)". Dec. 14, 2014, http://kb.vmware.com/selfservice/microsites/search.do?cmd=displayKc&docType=ex&bbid=TSEBB_1334428459608&url=&stateId=1%200%20462914399&dialogID=462898852&docTypeID=DT_KB_1_1&externalId=2011177&sliceId=1&rflD=
Chaussee, "Inside Windows Product Activation", Jul. 2001, <http://www.licenturion.com/xp>.

"How to generate and validate a software key license", Dec 14, 2014, Stack Overflow, <http://stackoverflow.com/questions/599837/how-to-generate-and-validate-a-software-license-key>.
Donsw, "License Key Generation", Jul 2005, Code Project, <http://www.codeproject.com/articles/11012/License-Key-Generation>.
"How are Software License Keys generated?", Dec. 14, 2014, Stack Overflow, <http://stackoverflow.com/questions/3002067/how-are-software-licncse-keys-generated>.
Decision on Appeal, USPTO PTAB Appeal No. 2012-011854 for U.S. Appl. No. 11/895,388 issued Mar. 12, 2015.

* cited by examiner



DATA PROTECTION METHOD AND DEVICE

CROSS-REFERENCE TO RELATED APPLICATIONS

This application is a division of application Ser. No. 10/602,777, filed Jun. 25, 2003, now U.S. Pat. No. 7,664,263, issued Feb. 16, 2010, which is a continuation of application Ser. No. 09/046,627, filed Mar. 24, 1998, now U.S. Pat. No. 6,598,162, issued Jul. 22, 2003. The entire disclosure of U.S. patent application Ser. No. 09/046,627 (which issued Jul. 22, 2003, as U.S. Pat. No. 6,598,162) and U.S. patent application Ser. No. 08/587,943, filed Jan. 17, 1996, (which issued Apr. 28, 1998, as U.S. Pat. No. 5,745,943) are hereby incorporated by reference in their entireties.

FIELD OF THE INVENTION

The invention relates to the protection of digital information. More particularly, the invention relates to a method and device for data protection.

With the advent of computer networks and digital multimedia, protection of intellectual property has become a prime concern for creators and publishers of digitized copies of copyrightable works, such as musical recordings, movies, video games, and computer software. One method of protecting copyrights in the digital domain is to use "digital watermarks."

The prior art includes copy protection systems attempted at many stages in the development of the software industry. These may be various methods by which a software engineer can write the software in a clever manner to determine if it has been copied, and if so to deactivate itself. Also included are undocumented changes to the storage format of the content. Copy protection was generally abandoned by the software industry, since pirates were generally just as clever as the software engineers and figured out ways to modify the software and deactivate the protection. The cost of developing such protection was not justified considering the level of piracy which occurred despite the copy protection.

Other methods for protection of computer software include the requirement of entering certain numbers or facts that may be included in a packaged software's manual, when prompted at start-up. These may be overcome if copies of the manual are distributed to unintended users, or by patching the code to bypass these measures. Other methods include requiring a user to contact the software vendor and to receive "keys" for unlocking software after registration attached to some payment scheme, such as credit card authorization. Further methods include network-based searches of a user's hard drive and comparisons between what is registered to that user and what is actually installed on the user's general computing device. Other proposals, by such parties as AT&T's Bell Laboratories, use "kerning" or actual distance in pixels, in the rendering of text documents, rather than a varied number of ASCII characters. However, this approach can often be defeated by graphics processing analogous to sound processing, which randomizes that information. All of these methods require outside determination and verification of the validity of the software license.

Digital watermarks can be used to mark each individual copy of a digitized work with information identifying the title, copyright holder, and even the licensed owner of a particular copy. When marked with licensing and ownership information, responsibility is created for individual copies where before there was none. Computer application programs can be watermarked by watermarking digital content

resources used in conjunction with images or audio data. Digital watermarks can be encoded with random or pseudo random keys, which act as secret maps for locating the watermarks. These keys make it impossible for a party to find the watermark without having the key. In addition, the encoding method can be enhanced to force a party to cause damage to a watermarked data stream when trying to erase a random-key watermark. Other information is disclosed in "Technology: Digital Commerce", Denise Caruso, New York Times, Aug. 7, 1995; and "Copyrighting in the Information Age", Harley Ungar, ONLINE MARKETPLACE, September 1995, Jupiter Communications.

Additionally, other methods for hiding information signals in content signals, are disclosed in U.S. Pat. No. 5,319,735—Preuss et al. and U.S. Pat. No. 5,379,345—Greenberg.

It is desirable to use a "stega-cipher" or watermarking process to hide the necessary parts or resources of the executable object code in the digitized sample resources. It is also desirable to further modify the underlying structure of an executable computer application such that it is more resistant to attempts at patching and analysis by memory capture. A computer application seeks to provide a user with certain utilities or tools, that is, users interact with a computer or similar device to accomplish various tasks and applications provide the relevant interface. Thus, a level of authentication can also be introduced into software, or "digital products," that include digital content, such as audio, video, pictures or multimedia, with digital watermarks. Security is maximized because erasing this code watermark without a key results in the destruction of one or more essential parts of the underlying application, rendering the "program" useless to the unintended user who lacks the appropriate key. Further, if the key is linked to a license code by means of a mathematical function, a mechanism for identifying the licensed owner of an application is created.

It is also desirable to randomly reorganize program memory structure intermittently during program run time, to prevent attempts at memory capture or object code analysis aimed at eliminating licensing or ownership information, or otherwise modifying, in an unintended manner, the functioning of the application.

In this way, attempts to capture memory to determine underlying functionality or provide a "patch" to facilitate unauthorized use of the "application," or computer program, without destroying the functionality and thus usefulness of a copyrightable computer program can be made difficult or impossible.

It is thus the goal of the present invention to provide a higher level of copyright security to object code on par with methods described in digital watermarking systems for digitized media content such as pictures, audio, video and multimedia content in its multifarious forms, as described in previous disclosures, "Steganographic Method and Device" Ser. No. 08/489,172, filed Jun. 7, 1995, now U.S. Pat. No. 5,613,004, and "Human Assisted Random Key Generation and Application for Digital Watermark System", Ser. No. 08/587,944, filed on Jan. 17, 1996, now U.S. Pat. No. 5,822,432, the disclosure of which is hereby incorporated by reference.

It is a further goal of the present invention to establish methods of copyright protection that can be combined with such schemes as software metering, network distribution of code and specialized protection of software that is designed to work over a network, such as that proposed by Sun Microsystems in their HotJava browser and Java programming language, and manipulation of application code in proposed distribution of documents that can be exchanged with resources or the look and feel of the document being pre-

served over a network. Such systems are currently being offered by companies including Adobe, with their Acrobat software. This latter goal is accomplished primarily by means of the watermarking of font, or typeface, resources included in applications or documents, which determine how a bitmap representation of the document is ultimately drawn on a presentation device.

The present invention includes an application of the technology of "digital watermarks." As described in previous disclosures, "Steganographic Method and Device" and "Human Assisted Random Key Generation and Application for Digital Watermark System," watermarks are particularly suitable to the identification, metering, distributing and authenticating digitized content such as pictures, audio, video and derivatives thereof under the description of "multimedia content." Methods have been described for combining both cryptographic methods, and steganography, or hiding something in plain view. Discussions of these technologies can be found in Applied Cryptography by Bruce Schneier and The Code Breakers by David Kahn. For more information on prior art public-key cryptosystems see U.S. Pat. No. 4,200,770 Diffie-Hellman, U.S. Pat. No. 4,218,582 Hellman, U.S. Pat. No. 4,405,829 RSA, U.S. Pat. No. 4,424,414 Hellman Pohl. Computer code, or machine language instructions, which are not digitized and have zero tolerance for error, must be protected by derivative or alternative methods, such as those disclosed in this invention, which focuses on watermarking with "keys" derived from license codes or other ownership identification information, and using the watermarks encoded with such keys to hide an essential subset of the application code resources.

BACKGROUND OF THE INVENTION

Increasingly, commercially valuable information is being created and stored in "digital" form. For example, music, photographs and video can all be stored and transmitted as a series of numbers, such as 1's and 0's. Digital techniques let the original information be recreated in a very accurate manner. Unfortunately, digital techniques also let the information be easily copied without the information owner's permission.

Because unauthorized copying is clearly a disincentive to the digital distribution of valuable information, it is important to establish responsibility for copies and derivative copies of such works. For example, if each authorized digital copy of a popular song is identified with a unique number, any unauthorized copy of the song would also contain the number. This would allow the owner of the information, such as a song publisher, to investigate who made the unauthorized copy. Unfortunately, it is possible that the unique number could be erased or altered if it is simply tacked on at the beginning or end of the digital information.

As will be described, known digital "watermark" techniques give creators and publishers of digitized multimedia content localized, secured identification and authentication of that content. In considering the various forms of multimedia content, such as "master," stereo, National Television Standards Committee (NTSC) video, audio tape or compact disc, tolerance of quality will vary with individuals and affect the underlying commercial and aesthetic value of the content. For example, if a digital version of a popular song sounds distorted, it will be less valuable to users. It is therefore desirable to embed copyright, ownership or purchaser information, or some combination of these and related data, into the content in a way that will damage the content if the watermark is removed without authorization.

To achieve these goals, digital watermark systems insert ownership information in a way that causes little or no noticeable effects, or "artifacts," in the underlying content signal. For example, if a digital watermark is inserted into a digital version of a song, it is important that a listener not be bothered by the slight changes introduced by the watermark. It is also important for the watermark technique to maximize the encoding level and "location sensitivity" in the signal to force damage to the content signal when removal is attempted. Digital watermarks address many of these concerns, and research in the field has provided extremely robust and secure implementations.

What has been overlooked in many applications described in the art, however, are systems which closely mimic distribution of content as it occurs in the real world. For instance, many watermarking systems require the original un-watermarked content signal to enable detection or decode operations. These include highly publicized efforts by NEC, Digimarc and others. Such techniques are problematic because, in the real world, original master copies reside in a rights holders vaults and are not readily available to the public.

With much activity overly focused on watermark survivability, the security of a digital watermark is suspect. Any simple linear operation for encoding information into a signal may be used to erase the embedded signal by inverting the process. This is not a difficult task, especially when detection software is a plug-in freely available to the public, such as with Digimarc. In general, these systems seek to embed cryptographic information, not cryptographically embed information into target media content.

Other methods embed ownership information that is plainly visible in the media signal, such as the method described in U.S. Pat. No. 5,530,739 to Braudaway et al. The system described in Braudaway protects a digitized image by encoding a visible watermark to deter piracy. Such an implementation creates an immediate weakness in securing the embedded information because the watermark is plainly visible. Thus, no search for the embedded signal is necessary and the watermark can be more easily removed or altered. For example, while certainly useful to some rights owners, simply placing the symbol "©" in the digital information would only provide limited protection. Removal by adjusting the brightness of the pixels forming the "©" would not be difficult with respect to the computational resources required.

Other relevant prior art includes U.S. Pat. Nos. 4,979,210 and 5,073,925 to Nagata et al., which encodes information by modulating an audio signal in the amplitude/time domain. The modulations introduced in the Nagata process carry a "copy/don't copy" message, which is easily found and circumvented by one skilled in the art. The granularity of encoding is fixed by the amplitude and frequency modulation limits required to maintain inaudibility. These limits are relatively low, making it impractical to encode more information using the Nagata process.

Although U.S. Pat. No. 5,661,018 to Leighton describes a means to prevent collusion attacks in digital watermarks, the disclosed method may not actually provide the security described. For example, in cases where the watermarking technique is linear, the "insertion envelope" or "watermarking space" is well-defined and thus susceptible to attacks less sophisticated than collusion by unauthorized parties. Over-encoding at the watermarking encoding level is but one simple attack in such linear implementations. Another consideration not made by Leighton is that commercially-valuable content may already exist in a un-watermarked form somewhere, easily accessible to potential pirates, gutting the need for any type of collusive activity. Digitally signing the

embedded signal with preprocessing of watermark data is more likely to prevent successful collusion. Furthermore, a "baseline" watermark as disclosed is quite subjective. It is simply described elsewhere in the art as the "perceptually significant" regions of a signal. Making a watermarking function less linear or inverting the insertion of watermarks would seem to provide the same benefit without the additional work required to create a "baseline" watermark. Indeed, watermarking algorithms should already be capable of defining a target insertion envelope or region without additional steps. What is evident is the Leighton patent does not allow for initial prevention of attacks on an embedded watermark as the content is visibly or audibly unchanged.

It is also important that any method for providing security also function with broadcasting media over networks such as the Internet, which is also referred to as "streaming." Commercial "plug-in" products such as RealAudio and RealVideo, as well as applications by vendors VDO Net and Xtreme, are common in such network environments. Most digital watermark implementations focus on common file base signals and fail to anticipate the security of streamed signals. It is desirable that any protection scheme be able to function with a plug-in player without advanced knowledge of the encoded media stream.

Other technologies focus solely on file-based security. These technologies illustrate the varying applications for security that must be evaluated for different media and distribution environments. Use of cryptolopes or cryptographic containers, as proposed by IBM in its Cryptolope product, and InterTrust, as described in U.S. Pat. Nos. 4,827,508, 4,977,594, 5,050,213 and 5,410,598, may discourage certain forms of piracy. Cryptographic containers, however, require a user to subscribe to particular decryption software to decrypt data. IBM's InfoMarket and InterTrust's DigiBox, among other implementations, provide a generalized model and need proprietary architecture to function. Every user must have a subscription or registration with the party which encrypts the data. Again, as a form of general encryption, the data is scrambled or encrypted without regard to the media and its formatting. Finally, control over copyrights or other neighboring rights is left with the implementing party, in this case, IBM, InterTrust or a similar provider.

Methods similar to these "trusted systems" exist, and Cerberus Central Limited and Liquid Audio, among a number of companies, offer systems which may functionally be thought of as subsets of IBM and InterTrust's more generalized security offerings. Both Cerberus and Liquid Audio propose proprietary player software which is registered to the user and "locked" in a manner parallel to the locking of content that is distributed via a cryptographic container. The economic trade-off in this model is that users are required to use each respective companies' proprietary player to play or otherwise manipulate content that is downloaded. If, as is the case presently, most music or other media is not available via these proprietary players and more companies propose non-compatible player formats, the proliferation of players will continue. Cerberus and Liquid Audio also by way of extension of their architectures provide for "near-CD quality" but proprietary compression. This requirement stems from the necessity not to allow content that has near-identical data make-up to an existing consumer electronic standard, in Cerberus and Liquid Audio's case the so-called Red Book audio CD standard of 16 bit 44.1 kHz, so that comparisons with the proprietary file may not yield how the player is secured. Knowledge of the player's file format renders its security ineffective as a file may be replicated and played on any common player, not the intended proprietary player of the provider of previously

secured and uniquely formatted content. This is the parallel weakness to public key crypto-systems which have gutted security if enough plain text and cipher text comparisons enable a pirate to determine the user's private key.

Many approaches to digital watermarking leave detection and decoding control with the implementing party of the digital watermark, not the creator of the work to be protected. A set of secure digital watermark implementations address this fundamental control issue forming the basis of key-based approaches. These are covered by the following patents and pending applications, the entire disclosures of which are hereby incorporated by reference: U.S. Pat. No. 5,613,004 entitled "Steganographic Method and Device" and its derivative U.S. patent application Ser. No. 08/775,216 (which issued Nov. 11, 1997, as U.S. Pat. No. 5,687,236), U.S. patent application Ser. No. 08/587,944 entitled "Human Assisted Random Key Generation and Application for Digital Watermark System" (which issued Oct. 13, 1998, as U.S. Pat. No. 5,822,432), U.S. patent application Ser. No. 08/587,943 entitled "Method for Stega-Cipher Protection of Computer Code" (which issued Apr. 28, 1998, as U.S. Pat. No. 5,748,569), U.S. patent application Ser. No. 08/677,435 entitled "Optimization Methods for the Insertion, Protection, and Detection of Digital Watermarks in Digitized Data" (which issued Mar. 30, 1999, as U.S. Pat. No. 5,889,868) and U.S. patent application Ser. No. 08/772,222 entitled "Z-Transform Implementation of Digital Watermarks" (which issued Jun. 20, 2000, as U.S. Pat. No. 6,078,664). Public key crypto-systems are described in U.S. Pat. Nos. 4,200,770, 4,218,582, 4,405,829 and 4,424,414, the entire disclosures of which are also hereby incorporated by reference.

In particular, an improved protection scheme is described in "Method for Stega-Cipher Protection of Computer Code," U.S. patent application Ser. No. 08/587,943 (which issued Apr. 28, 1998, as U.S. Pat. No. 5,748,569). This technique uses the key-based insertion of binary executable computer code within a content signal that is subsequently, and necessarily, used to play or otherwise manipulate the signal in which it is encoded. With this system, however, certain computational requirements, such as one digital player per digital copy of content, may be necessitated. For instance, a consumer may download many copies of watermarked content. With this technique, the user would also be downloading as many copies of the digital player program. While this form of security may be desirable for some applications, it is not appropriate in many circumstances.

Finally, even when digital information is distributed in encoded form, it may be desirable to allow unauthorized users to play the information with a digital player, perhaps with a reduced level of quality. For example, a popular song may be encoded and freely distributed in encoded form to the public. The public, perhaps using commonly available plug-in digital players, could play the encoded content and hear the music in some degraded form. The music may sound choppy, or fuzzy or be degraded in some other way. This lets the public decide, based on the available lower quality version of the song, if they want to purchase a key from the publisher to decode, or "clean-up," the content. Similar approaches could be used to distribute blurry pictures or low quality video. Or even "degraded" text, in the sense that only authenticated portions of the text can be determined with the predetermined key or a validated digital signature for the intended message.

In view of the foregoing, it can be appreciated that a substantial need exists for a method allowing encoded content to

be played, with degraded quality, by a plug-in digital player, and solving the other problems discussed above.

SUMMARY OF THE INVENTION

The disadvantages of the art are alleviated to a great extent by a method for combining transfer functions with predetermined key creation. In one embodiment, digital information, including a digital sample and format information, is protected by identifying and encoding a portion of the format information. Encoded digital information, including the digital sample and the encoded format information, is generated to protect the original digital information.

In another embodiment, a digital signal, including digital samples in a file format having an inherent granularity, is protected by creating a predetermined key. The predetermined key is comprised of a transfer function-based mask set to manipulate data at the inherent granularity of the file format of the underlying digitized samples.

It is thus a goal of the present invention, to provide a level of security for executable code on similar grounds as that which can be provided for digitized samples. Furthermore, the present invention differs from the prior art in that it does not attempt to stop copying, but rather, determines responsibility for a copy by ensuring that licensing information must be preserved in descendant copies from an original. Without the correct license information, the copy cannot function.

An improvement over the art is disclosed in the present invention, in that the software itself is a set of commands, compiled by software engineer, which can be configured in such a manner as to tie underlying functionality to the license or authorization of the copy in possession by the user. Without such verification, the functions sought out by the user in the form of software cease to properly work. Attempts to tamper or "patch" substitute code resources can be made highly difficult by randomizing the location of said resources in memory on an intermittent basis to resist most attacks at disabling the system.

With these and other advantages and features of the invention that will become hereinafter apparent, the nature of the invention may be more clearly understood by reference to the following detailed description of the invention, the appended claims and to the several drawings attached herein.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block flow diagram of a method for copy protection or authentication of digital information according to an embodiment of the present invention.

DETAILED DESCRIPTION

In accordance with an embodiment of the present invention, a method combines transfer functions with predetermined key creation. Increased security is achieved in the method by combining elements of "public-key steganography" with cryptographic protocols, which keep in-transit data secure by scrambling the data with "keys" in a manner that is not apparent to those with access to the content to be distributed. Because different forms of randomness are combined to offer robust, distributed security, the present invention addresses an architectural "gray space" between two important areas of security: digital watermarks, a subset of the more general art of steganography, and cryptography. One form of randomness exists in the mask sets that are randomly created to map watermark data into an otherwise unrelated digital signal. The second form of randomness is the random permu-

tations of data formats used with digital players to manipulate the content with the predetermined keys. These forms can be thought of as the transfer function versus the mapping function inherent to digital watermarking processes.

According to an embodiment of the present invention, a predetermined, or randomly generated, key is used to scramble digital information in a way that is unlike known "digital watermark" techniques and public key crypto-systems. As used herein, a key is also referred to as a "mask set" which includes one or more random or pseudo-random series of bits. Prior to encoding, a mask can be generated by any cryptographically secure random generation process. A block cipher, such as a Data Encryption Standard (DES) algorithm, in combination with a sufficiently random seed value, such as one created using a Message Digest 5 (MD5) algorithm, emulates a cryptographically secure random bit generator. The keys are saved in a database, along with information matching them to the digital signal, for use in descrambling and subsequent viewing or playback. Additional file format or transfer property information is prepared and made available to the encoder, in a bit addressable manner. As well, any authenticating function can be combined, such as Digital Signature Standard (DSS) or Secure Hash Algorithm (SHA).

Using the predetermined key comprised of a transfer function-based mask set, the data representing the original content is manipulated at the inherent granularity of the file format of the underlying digitized samples. Instead of providing, or otherwise distributing, watermarked content that is not noticeably altered, a partially "scrambled" copy of the content is distributed. The key is necessary both to register the sought-after content and to descramble the content into its original form.

The present invention uses methods disclosed in "Method for Stega-Cipher Protection of Computer Code," U.S. patent application Ser. No. 08/587,943 (which issued Apr. 28, 1998, as U.S. Pat. No. 5,748,569), with respect to transfer functions related to the common file formats, such as PICT, TIFF, AIFF, WAV, etc. Additionally, in cases where the content has not been altered beyond being encoded with such functional data, it is possible for a digital player to still play the content because the file format has not been altered. Thus, the encoded content could still be played by a plug-in digital player as discrete, digitally sampled signals, watermarked or not. That is, the structure of the file can remain basically unchanged by the watermarking process, letting common file format based players work with the "scrambled" content.

For example, the Compact Disc-Digital Audio (CD-DA) format stores audio information as a series of frames. Each frame contains a number of digital samples representing, for example, music, and a header that contains file format information. As shown in FIG. 1, according to an embodiment of the present invention some of the header information can be identified and "scrambled" using the predetermined key at steps 110 to 130. The music samples can remain unchanged. Using this technique, a traditional CD-DA player will be able to play a distorted version of the music in the sample. The amount of distortion will depend on the way, and extent, that the header, or file format, information has been scrambled. It would also be possible to instead scramble some of the digital samples while leaving the header information alone. In general, the digital signal would be protected by manipulating data at the inherent granularity, or "frames," of the CD-DA file format. To decode the information, a predetermined key is used before playing the digital information at steps 140 and 150.

A key-based decoder can act as a "plug-in" digital player of broadcast signal streams without foreknowledge of the

encoded media stream. Moreover, the data format orientation is used to partially scramble data in transit to prevent unauthorized descrambled access by decoders that lack authorized keys. A distributed key can be used to unscramble the scrambled content because a decoder would understand how to process the key. Similar to on-the-fly decryption operations, the benefits inherent in this embodiment include the fact that the combination of watermarked content security, which is key-based, and the descrambling of the data, can be performed by the same key which can be a plurality of mask sets. The mask sets may include primary, convolution and message delimiter masks with file format data included.

The creation of an optimized "envelope" for insertion of watermarks provides the basis of much watermark security, but is also a complementary goal of the present invention. The predetermined or random key that is generated is not only an essential map to access the hidden information signal, but is also the descrambler of the previously scrambled signal's format for playback or viewing.

In a system requiring keys for watermarking content and validating the distribution of the content, different keys may be used to encode different information while secure one way hash functions or one-time pads may be incorporated to secure the embedded signal. The same keys can be used to later validate the embedded digital signature, or even fully decode the digital watermark if desired. Publishers can easily stipulate that content not only be digitally watermarked but that distributors must check the validity of the watermarks by performing digital signature-checks with keys that lack any other functionality. The system can extend to simple authentication of text in other embodiments.

Before such a market is economically feasible, there are other methods for deploying key-based watermarking coupled with transfer functions to partially scramble the content to be distributed without performing full public key encryption, i.e., a key pair is not necessarily generated, simply, a predetermined key's function is created to re-map the data of the content file in a lossless process. Moreover, the scrambling performed by the present invention may be more dependent on the file in question. Dissimilarly, encryption is not specific to any particular media but is performed on data. The file format remains unchanged, rendering the file useable by any conventional viewer/player, but the signal quality can be intentionally degraded in the absence of the proper player and key. Public-key encryption seeks to completely obscure the sensitive "plaintext" to prevent comparisons with the "ciphertext" to determine a user's private keys. Centralized encryption only differs in the utilization of a single key for both encryption and decryption making the key even more highly vulnerable to attacks to defeat the encryption process. With the present invention, a highly sought after photograph may be hazy to the viewer using any number of commonly available, nonproprietary software or hardware, without the authorized key. Similarly, a commercially valuable song may sound poor.

The benefit of some form of cryptography is not lost in the present invention. In fact, some piracy can be deterred when the target signal may be known but is clearly being protected through scrambling. What is not anticipated by known techniques, is an ala carte method to change various aspects of file formatting to enable various "scrambled states" for content to be subsequently distributed. An image may lack all red pixels or may not have any of the most significant bits activated. An audio sample can similarly be scrambled to render it less-than-commercially viable.

The present invention also provides improvements over known network-based methods, such as those used for the

streaming of media data over the Internet. By manipulating file formats, the broadcast media, which has been altered to "fit" within electronic distribution parameters, such as bandwidth availability and error correction considerations, can be more effectively utilized to restrict the subsequent use of the content while in transit as well as real-time viewing or playing.

The mask set providing the transfer function can be read on a per-use basis by issuing an authorized or authenticating "key" for descrambling the signal that is apparent to a viewer or a player or possessor of the authenticating key. The mask set can be read on a per-computer basis by issuing the authorized key that is more generalized for the computer that receives the broadcast signals. Metering and subscription models become viable advantages over known digital watermark systems which assist in designating the ownership of a copy of digitized media content, but do not prevent or restrict the copying or manipulation of the sampled signal in question. For broadcast or streamed media, this is especially the case. Message authentication is also possible, though not guaranteeing the same security as an encrypted file as with general crypto systems.

The present invention thus benefits from the proprietary player model without relying on proprietary players. No new players will be necessary and existing multimedia file formats can be altered to exact a measure of security which is further increased when coupled with digital watermarks. As with most consumer markets for media content, predominant file formats exist, de facto, and corresponding formats for computers likewise exist. For a commercial compact disc quality audio recording, or 16 bit 44.1 kHz, corresponding file formats include: Audio Interchange File Format (AIFF), Microsoft WAV, Sound Designer II, Sun's .au, Apple's Quicktime, etc. For still image media, formats are similarly abundant: TIFF, PICT, JPEG, GIF, etc. Requiring the use of additional proprietary players, and their complementary file formats, for limited benefits in security is wasteful. Moreover, almost all computers today are multimedia-capable, and this is increasingly so with the popularity of Intel's MMX chip architecture and the PowerPC line of microchips. Because file formatting is fundamental in the playback of the underlying data, the predetermined key can act both as a map, for information to be encoded as watermark data regarding ownership, and a descrambler of the file that has been distributed. Limitations will only exist in how large the key must be retrofitted for a given application, but any manipulation of file format information is not likely to exceed the size of data required versus that for an entire proprietary player.

As with previous disclosures by the inventor on digital watermarking techniques, the present invention may be implemented with a variety of cryptographic protocols to increase both confidence and security in the underlying system. A predetermined key is described as a set of masks. These masks may include primary, convolution and message delimiter mask. In previous disclosures, the functionality of these masks is defined solely for mapping. The present invention includes a mask set which is also controlled by the distributing party of a copy of a given media signal. This mask set is a transfer function which is limited only by the parameters of the file format in question. To increase the uniqueness or security of each key used to scramble a given media file copy, a secure one way hash function can be used subsequent to transfer properties that are initiated to prevent the forging of a particular key. Public and private keys may be used as key pairs to further increase the unlikelihood that a key may be compromised.

These same cryptographic protocols can be combined with the embodiments of the present invention in administering streamed content that requires authorized keys to correctly display or play the streamed content in an unscrambled manner. As with digital watermarking, symmetric or asymmetric public key pairs may be used in a variety of implementations. Additionally, the need for certification authorities to maintain authentic key-pairs becomes a consideration for greater security beyond symmetric key implementations. The cryptographic protocols makes possible, as well, a message of text to be authenticated by a message authenticating function in a general computing device that is able to ensure secure message exchanges between authorizing parties.

An executable computer program is variously referred to as an application, from the point of view of a user, or executable object code from the point of view of the engineer. A collection of smaller, atomic (or indivisible) chunks of object code typically comprise the complete executable object code or application which may also require the presence of certain data resources. These indivisible portions of object code correspond with the programmers' function or procedure implementations in higher level languages, such as C or Pascal. In creating an application, a programmer writes "code" in a higher level language, which is then compiled down into "machine language," or, the executable object code, which can actually be run by a computer, general purpose or otherwise. Each function, or procedure, written in the programming language, represents a self-contained portion of the larger program, and implements, typically, a very small piece of its functionality. The order in which the programmer types the code for the various functions or procedures, and the distribution of and arrangement of these implementations in various files which hold them is unimportant. Within a function or procedure, however, the order of individual language constructs, which correspond to particular machine instructions is important, and so functions or procedures are considered indivisible for purposes of this discussion. That is, once a function or procedure is compiled, the order of the machine instructions which comprise the executable object code of the function is important and their order in the computer memory is of vital importance. Note that many "compilers" perform "optimizations" within functions or procedures, which determine, on a limited scale, if there is a better arrangement for executable instructions which is more efficient than that constructed by the programmer, but does not change the result of the function or procedure. Once these optimizations are performed, however, making random changes to the order of instructions is very likely to "break" the function. When a program is compiled, then, it consists of a collection of these sub-objects, whose exact order or arrangement in memory is not important, so long as any sub-object which uses another sub-object knows where in memory it can be found.

The memory address of the first instruction in one of these sub-objects is called the "entry point" of the function or procedure. The rest of the instructions comprising that sub-object immediately follow from the entry point. Some systems may prefix information to the entry point which describes calling and return conventions for the code which follows, an example is the Apple Macintosh Operating System (MacOS). These sub-objects can be packaged into what are referred to in certain systems as "code resources," which may be stored separately from the application, or shared with other applications, although not necessarily. Within an application there are also data objects, which consist of some data to be operated on by the executable code. These data objects

are not executable. That is, they do not consist of executable instructions. The data objects can be referred to in certain systems as "resources."

When a user purchases or acquires a computer program, she seeks a computer program that "functions" in a desired manner. Simply, computer software is overwhelmingly purchased for its underlying functionality. In contrast, persons who copy multimedia content, such as pictures, audio and video, do so for the entertainment or commercial value of the content. The difference between the two types of products is that multimedia content is not generally interactive, but is instead passive, and its commercial value relates more on passive not interactive or utility features, such as those required in packaged software, set-top boxes, cellular phones, VCRs, PDAs, and the like. Interactive digital products which include computer code may be mostly interactive but can also contain content to add to the interactive experience of the user or make the underlying utility of the software more aesthetically pleasing. It is a common concern of both of these creators, both of interactive and passive multimedia products, that "digital products" can be easily and perfectly copied and made into unpaid or unauthorized copies. This concern is especially heightened when the underlying product is copyright protected and intended for commercial use.

The first method of the present invention described involves hiding necessary "parts" or code "resources" in digitized sample resources using a "digital watermarking" process, such as that described in the "Steganographic Method and Device" patent application. The basic premise for this scheme is that there are a certain sub-set of executable code resources, that comprise an application and that are "essential" to the proper function of the application. In general, any code resource can be considered "essential" in that if the program proceeds to a point where it must "call" the code resource and the code resource is not present in memory, or cannot be loaded, then the program fails. However, the present invention uses a definition of "essential" which is more narrow. This is because, those skilled in the art or those with programming experience, may create a derivative program, not unlike the utility provided by the original program, by writing additional or substituted code to work around unavailable resources. This is particularly true with programs that incorporate an optional "plug-in architecture," where several code resources may be made optionally available at run-time. The present invention is also concerned with concentrated efforts by technically skilled people who can analyze executable object code and "patch" it to ignore or bypass certain code resources. Thus, for the present embodiment's purposes, "essential" means that the function which distinguishes this application from any other application depends upon the presence and use of the code resource in question. The best candidates for this type of code resources are NOT optional, or plug-in types, unless special care is taken to prevent work-arounds.

Given that there are one or more of these essential resources, what is needed to realize the present invention is the presence of certain data resources of a type which are amenable to the "stega-cipher" process described in the "Steganographic Method and Device" patent U.S. Pat. No. 5,613,004. Data which consists of image or audio samples is particularly useful. Because this data consists of digital samples, digital watermarks can be introduced into the samples. What is further meant is that certain applications include image and audio samples which are important to the look and feel of the program or are essential to the processing of the application's functionality when used by the user. These computer programs are familiar to users of computers but also less obvious

to users of other devices that run applications that are equivalent in some measure of functionality to general purpose computers including, but not limited to, set-top boxes, cellular phones, "smart televisions," PDAs and the like. However, programs still comprise the underlying "operating systems" of these devices and are becoming more complex with increases in functionality.

One method of the present invention is now discussed. When code and data resources are compiled and assembled into a precursor of an executable program the next step is to use a utility application for final assembly of the executable application. The programmer marks several essential code resources in a list displayed by the utility. The utility will choose one or several essential code resources, and encode them into one or several data resources using the stegacipher process. The end result will be that these essential code resources are not stored in their own partition, but rather stored as encoded information in data resources. They are not accessible at run-time without the key. Basically, the essential code resources that provide functionality in the final end-product, an executable application or computer program, are no longer easily and recognizably available for manipulation by those seeking to remove the underlying copyright or license, or its equivalent information, or those with skill to substitute alternative code resources to "force" the application program to run as an unauthorized copy. For the encoding of the essential code resources, a "key" is needed. Such a key is similar to those described in U.S. Pat. No. 5,613,004, the "Steganographic Method and Device" patent. The purpose of this scheme is to make a particular licensed copy of an application distinguishable from any other. It is not necessary to distinguish every instance of an application, merely every instance of a license. A licensed user may then wish to install multiple copies of an application, legally or with authorization. This method, then, is to choose the key so that it corresponds, is equal to, or is a function of, a license code or license descriptive information, not just a text file, audio clip or identifying piece of information as desired in digital watermarking schemes extant and typically useful to stand-alone, digitally sampled content. The key is necessary to access the underlying code, i.e., what the user understands to be the application program.

The assembly utility can be supplied with a key generated from a license code generated for the license in question. Alternatively, the key, possibly random, can be stored as a data resource and encrypted with a derivative of the license code. Given the key, it encodes one or several essential resources into one or several data resources. Exactly which code resources are encoded into which data resources may be determined in a random or pseudo random manner. Note further that the application contains a code resource which performs the function of decoding an encoded code resource from a data resource. The application must also contain a data resource which specifies in which data resource a particular code resource is encoded. This data resource is created and added at assembly time by the assembly utility. The application can then operate as follows:

- 1) when it is run for the first time, after installation, it asks the user for personalization information, which includes the license code. This can include a particular computer configuration;
- 2) it stores this information in a personalization data resource;
- 3) Once it has the license code, it can then generate the proper decoding key to access the essential code resources.

Note that the application can be copied in an uninhibited manner, but must contain the license code issued to the licensed owner, to access its essential code resources. The goal of the invention, copyright protection of computer code and establishment of responsibility for copies, is thus accomplished.

This invention represents a significant improvement over prior art because of the inherent difference in use of purely informational watermarks versus watermarks which contain executable object code. If the executable object code in a watermark is essential to an application which accesses the data which contains the watermark, this creates an all-or-none situation. Either the user must have the extracted watermark, or the application cannot be used, and hence the user cannot gain full access to the presentation of the information in the watermark bearing data. In order to extract a digital watermark, the user must have a key. The key, in turn, is a function of the license information for the copy of the software in question. The key is fixed prior to final assembly of the application files, and so cannot be changed at the option of the user. That, in turn, means the license information in the software copy must remain fixed, so that the correct key is available to the software. The key and the license information are, in fact, interchangeable. One is merely more readable than the other. In U.S. Pat. No. 5,613,004, the "Steganographic Method and Device, patent", the possibility of randomization erasure attacks on digital watermarks was discussed. Simply, it is always possible to erase a digital watermark, depending on how much damage you are willing to do to the watermark-bearing content stream. The present invention has the significant advantage that you must have the watermark to be able to use the code it contains. If you erase the watermark you have lost a key piece of the functionality of the application, or even the means to access the data which bear the watermark.

A preferred embodiment would be implemented in an embedded system, with a minimal operating system and memory. No media playing "applets," or smaller sized applications as proposed in new operating environments envisioned by Sun Microsystems and the advent of Sun's Java operating system, would be permanently stored in the system, only the bare necessities to operate the device, download information, decode watermarks and execute the applets contained in them. When an applet is finished executing, it is erased from memory. Such a system would guarantee that content which did not contain readable watermarks could not be used. This is a powerful control mechanism for ensuring that content to be distributed through such a system contains valid watermarks. Thus, in such networks as the Internet or set-top box controlled cable systems, distribution and exchange of content would be made more secure from unauthorized copying to the benefit of copyright holders and other related parties. The system would be enabled to invalidate, by default, any content which has had its watermark(s) erased, since the watermark conveys, in addition to copyright information, the means to fully access, play, record or otherwise manipulate, the content.

A second method according to the present invention is to randomly re-organize program memory structure to prevent attempts at memory capture or object code analysis. The object of this method is to make it extremely difficult to perform memory capture-based analysis of an executable computer program. This analysis is the basis for a method of attack to defeat the system envisioned by the present invention.

Once the code resources of a program are loaded into memory, they typically remain in a fixed position, unless the computer operating system finds it necessary to rearrange

certain portions of memory during “system time,” when the operating system code, not application code, is running. Typically, this is done in low memory systems, to maintain optimal memory utilization. The MacOS for example, uses Handles, which are double-indirect pointers to memory locations, in order to allow the operating system to rearrange memory transparently, underneath a running program. If a computer program contains countermeasures against unlicensed copying, a skilled technician can often take a snapshot of the code in memory, analyze it, determine which instructions comprise the countermeasures, and disable them in the stored application file, by means of a “patch.” Other applications for designing code that moves to prevent scanning-tunnelling microscopes, and similar high sensitive hardware for analysis of electronic structure of microchips running code, have been proposed by such parties as Wave Systems. Designs of Wave Systems’ microchip are intended for preventing attempts by hackers to “photograph” or otherwise determine “burn in” to microchips for attempts at reverse engineering. The present invention seeks to prevent attempts at understanding the code and its organization for the purpose of patching it. Unlike systems such as Wave Systems’, the present invention seeks to move code around in such a manner as to complicate attempts by software engineers to reengineer a means to disable the methods for creating licensed copies on any device that lacks “trusted hardware.” Moreover, the present invention concerns itself with any application software that may be used in general computing devices, not chipsets that are used in addition to an underlying computer to perform encryption. Wave Systems’ approach to security of software, if interpreted similarly to the present invention, would dictate separate microchip sets for each piece of application software that would be tamperproof. This is not consistent with the economics of software and its distribution.

Under the present invention, the application contains a special code resource which knows about all the other code resources in memory. During execution time, this special code resource, called a “memory scheduler,” can be called periodically, or at random or pseudo random intervals, at which time it intentionally shuffles the other code resources randomly in memory, so that someone trying to analyze snapshots of memory at various intervals cannot be sure if they are looking at the same code or organization from one “break” to the next. This adds significant complexity to their job. The scheduler also randomly relocates itself when it is finished. In order to do this, the scheduler would have to first copy itself to a new location, and then specifically modify the program counter and stack frame, so that it could then jump into the new copy of the scheduler, but return to the correct calling frame. Finally, the scheduler would need to maintain a list of all memory addresses which contain the address of the scheduler, and change them to reflect its new location.

The methods described above accomplish the purposes of the invention—to make it hard to analyze captured memory containing application executable code in order to create an identifiable computer program or application that is different from other copies and is less susceptible to unauthorized use by those attempting to disable the underlying copyright protection system. Simply, each copy has particular identifying information making that copy different from all other copies.

Although various embodiments are specifically illustrated and described herein, it will be appreciated that modifications and variations of the present invention are covered by the above teachings and within the purview of the appended claims without departing from the spirit and intended scope of the invention.

What is claimed is:

1. A computer-based method for modifying software, comprising: receiving, in a computer having a processor and memory, software, wherein said software provides a specified functionality; embedding a watermark into said software, using said computer, said watermark encoding at least one first license code, thereby resulting in a first license code encoded watermarked software; and wherein said first license code encoded watermarked software is configured to query a user for personalization information during its installation.

2. A computer-based method for modifying software, comprising: receiving, in a computer having a processor and memory, software, wherein said software provides a specified functionality; embedding a watermark into said software, using said computer, said watermark encoding at least one first license code, thereby resulting in a first license code encoded watermarked software; wherein said watermark is accessible with a key; and said key enables said first license code encoded watermarked software to provide said specified functionality.

3. A computer-based method for modifying software, comprising: receiving, in a computer having a processor and memory, software, wherein said software provides a specified functionality; embedding a watermark into said software, using said computer, said watermark encoding at least one first license code, thereby resulting in a first license code encoded watermarked software; and wherein the step of embedding the software with a watermark is performed during execution of the software.

4. An article of manufacture comprising a machine readable medium, having thereon stored instructions adapted to be executed by a processor of a computer system, said computer system including a memory, which instructions when executed by said computer system result in a process comprising:

said computer system storing a software in said memory; said computer system receiving licensing information as an input and using said licensing information in an algorithm to identify a watermark in said software.

5. The article of manufacture of claim 4, wherein said watermark encodes therein information defining an executable code providing a functionality of said software.

6. The article of manufacture of claim 4, wherein the watermark affects functionality of the watermarked software.

7. The article of manufacture of claim 5, wherein said instructions comprise decode instructions for said computer system to use said licensing information to generate a decode key for decoding said software.

8. The article of manufacture of claim 7, wherein said licensing information comprises a license key, and said decode instructions instruct said computer to determine said license key from said licensing information and to generate said decode key using said license key.

9. The article of manufacture of claim 4: wherein said watermark encodes a license key; said instructions include a prompt to enter licensing information; wherein said software provides a certain functionality after receipt of licensing information in response to said prompt only if said licensing information comprises a license key encoded in said watermark.

10. A computer-based system for modifying software, comprising: a computer having a processor and memory; wherein said computer is programmed to receive software that provides a specified functionality when installed on a computer system; wherein said computer is programmed to embed a watermark into said software; wherein said water-

17

mark encodes at least one first license code, thereby resulting in a first license code encoded watermarked software; and wherein said first license code encoded watermarked software is designed to prompt for entry of licensing information and only provides a certain functionality if licensing information entered in response to said prompt comprises at least one of said at least one first license code encoded in said watermark.

11. A method for licensed software use, the method comprising:

- loading a software product on a computer, said computer comprising a processor, memory, an input, and an output, so that said computer is programmed to execute said software product;
- said software product outputting a prompt for input of license information; and
- said software product using license information entered via said input in response to said prompt in a routine designed to decode a first license code encoded in said software product.

12. A method for encoding software code using a computer having a processor and memory, comprising: storing a software code in said memory; wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system; and encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key encoded software code; and wherein, when installed on a computer system, said first license key encoded software code will provide said specified underlying functionality only after receipt of said first license key.

18

13. A method for encoding software code using a computer having a processor and memory, comprising:

- storing a software code in said memory;
- wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system; and
- modifying, by said computer, using a first license key and an encoding algorithm, said software code, to form a modified software code; and
- wherein said modifying comprises encoding said first code resource to form an encoded first code resource;
- wherein said modified software code comprises said encoded first code resource, and a decode resource for decoding said encoded first code resource;
- wherein said decode resource is configured to decode said encoded first code resource upon receipt of said first license key.

14. A method for encoding software code using a computer having a processor and memory, comprising:

- storing a software code in said memory;
- wherein said software code defines software code interrelationships between code resources that result in a specified underlying functionality when installed on a computer system; and
- encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key encoded software code in which at least one of said software code interrelationships are encoded.

* * * * *

Exhibit 3



United States Patent [19]
Beetcher et al.

[11] **Patent Number:** 5,933,497
[45] **Date of Patent:** Aug. 3, 1999

[54] **APPARATUS AND METHOD FOR CONTROLLING ACCESS TO SOFTWARE** 4,932,054 6/1990 Chou et al. 380/4
4,959,861 9/1990 Howlette 380/4

[75] **Inventors:** Robert Carl Beetcher; Michael Joseph Corrigan; Francis Joseph Reardon, Jr., all of Rochester; James William Moran, Eyota, all of Minn.

Primary Examiner—Bernarr E. Gregory
Attorney, Agent, or Firm—Roy W. Truelson

[73] **Assignee:** International Business Machines Corporation, Armonk, N.Y.

[57] **ABSTRACT**

[21] **Appl. No.:** 08/011,042
[22] **Filed:** Jan. 29, 1993

Software is distributed without entitlement to run, while a separately distributed encrypted entitlement key enables execution of the software. The key includes the serial number of the computer for which the software is licensed, together with a plurality of entitlement bits indicating which software modules are entitled to run on the machine. A secure decryption mechanism contained on the computer fetches its serial number and uses it as a key to decrypt the entitlement information, which is then stored in a product lock table in memory. The distributed software contains a plurality of entitlement verification triggers. Each trigger is a single machine instruction in the object code, identifying a product number of the software module. When a trigger is encountered during execution, the computer checks the product lock table entry corresponding to the product number of the software. If the product is entitled to run, execution continues normally; otherwise execution is aborted. Because this verification involves only a single machine instruction, it can be done with virtually no impact to overall system performance. As a result, it is possible to place a substantial number of such entitlement verification triggers in the object code, making it virtually impossible for someone to alter the code by "patching" the triggers. The triggering instruction may alternatively perform some useful work in parallel with entitlement verification.

Related U.S. Application Data

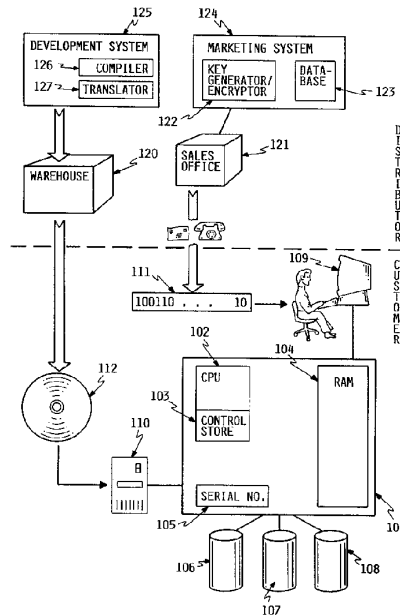
[63] Continuation of application No. 07/629,295, Dec. 14, 1990, abandoned.
[51] **Int. Cl.⁶** **H04L 9/00**
[52] **U.S. Cl.** **380/4; 380/9; 380/21; 380/23; 380/25; 380/49; 380/50**
[58] **Field of Search** 380/4, 9, 20, 21, 380/23, 25, 43, 44, 49, 50

[56] **References Cited**

U.S. PATENT DOCUMENTS

3,609,697	9/1971	Blevins et al.	364/200
4,433,207	2/1984	Best	380/4
4,471,163	9/1984	Donald	380/4
4,683,553	7/1987	Mollier	380/4
4,685,055	8/1987	Thomas	364/200
4,866,769	9/1989	Karp	380/4
4,903,296	2/1990	Chandra et al.	380/4

26 Claims, 10 Drawing Sheets



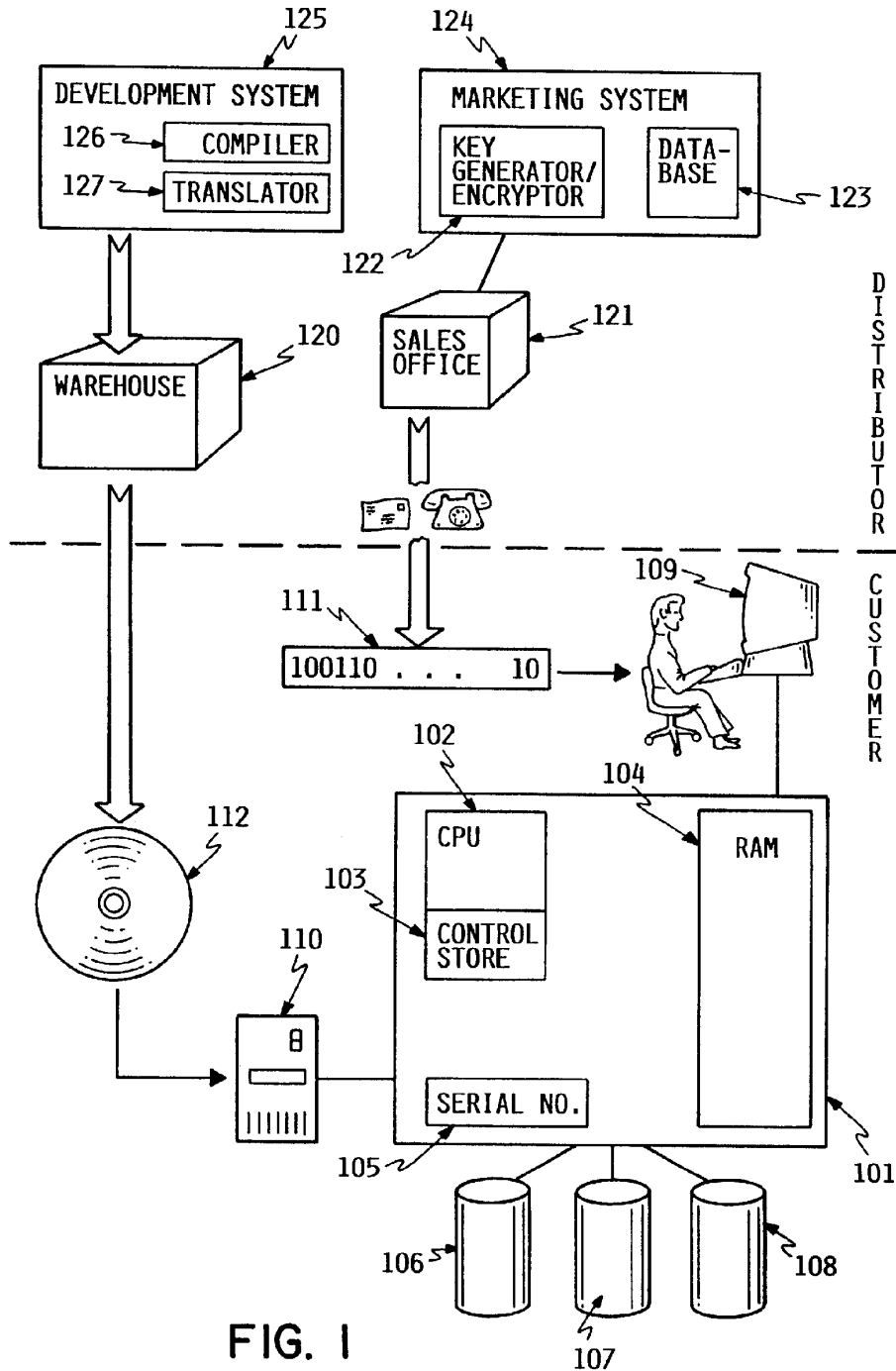


FIG. 1

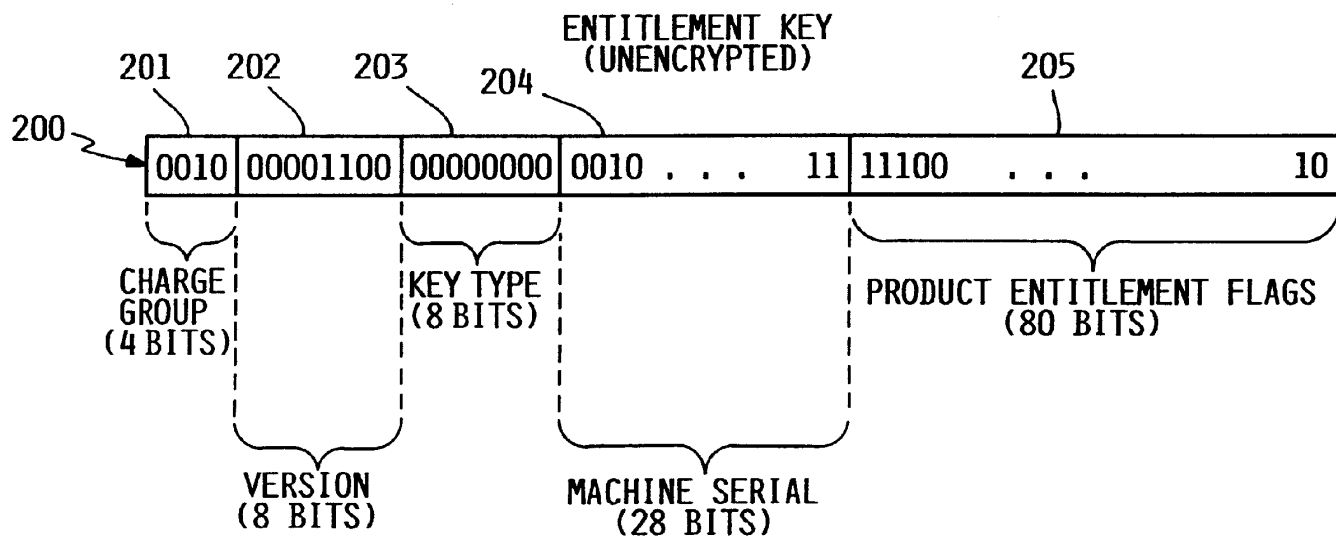
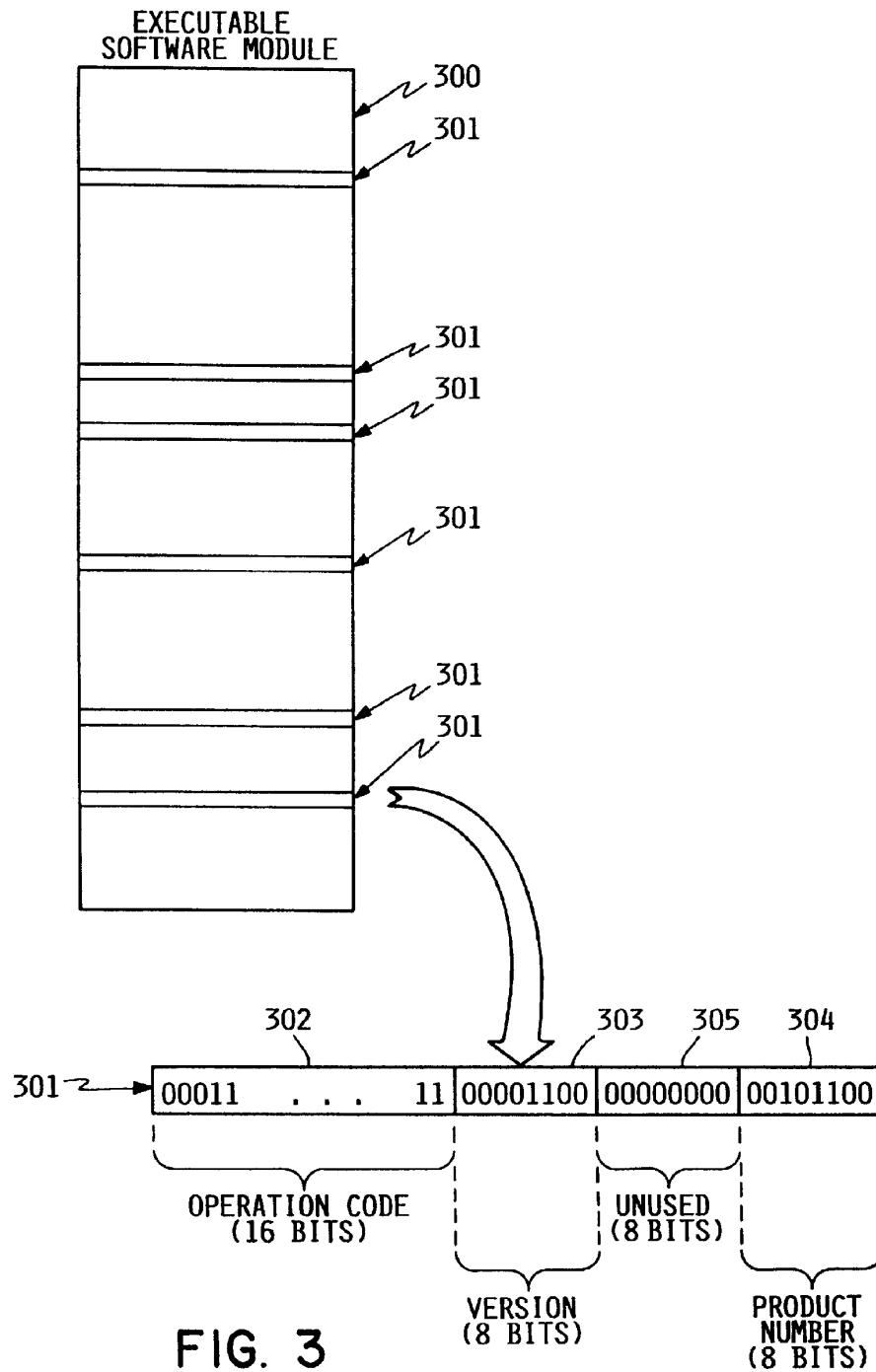


FIG. 2



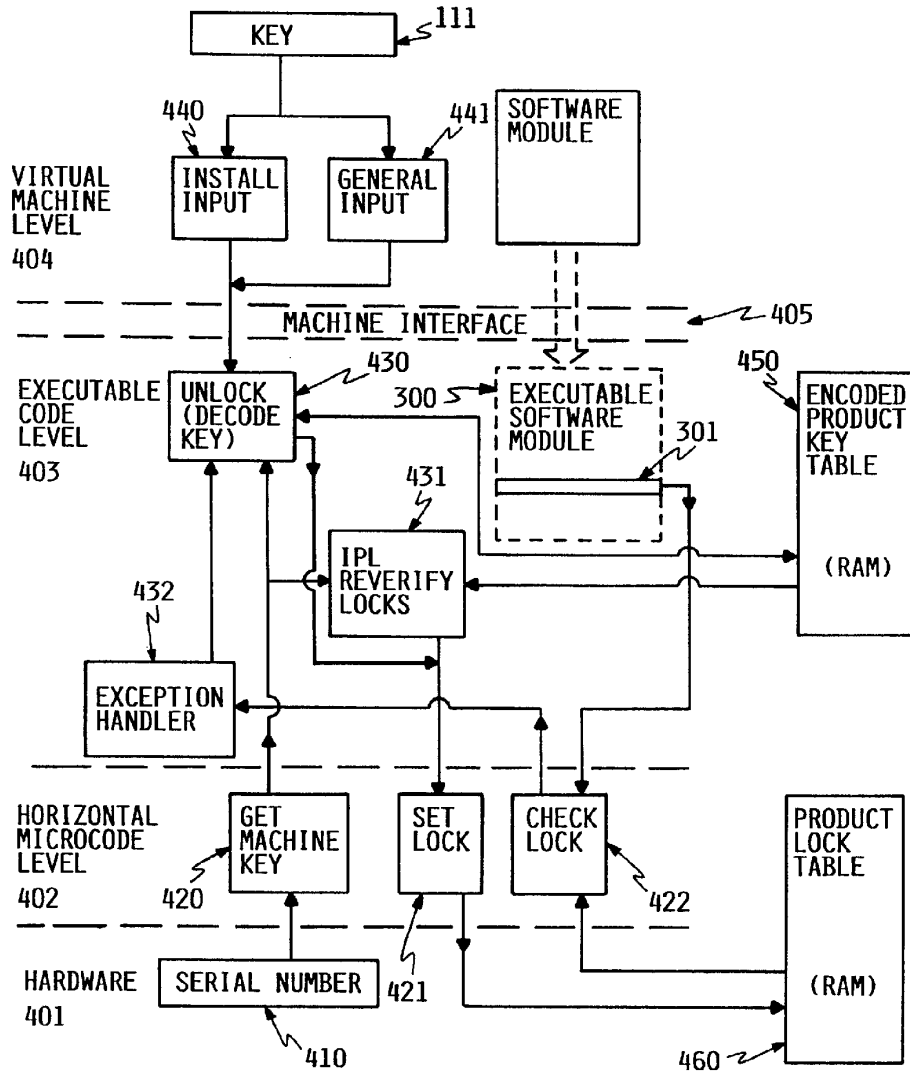


FIG. 4

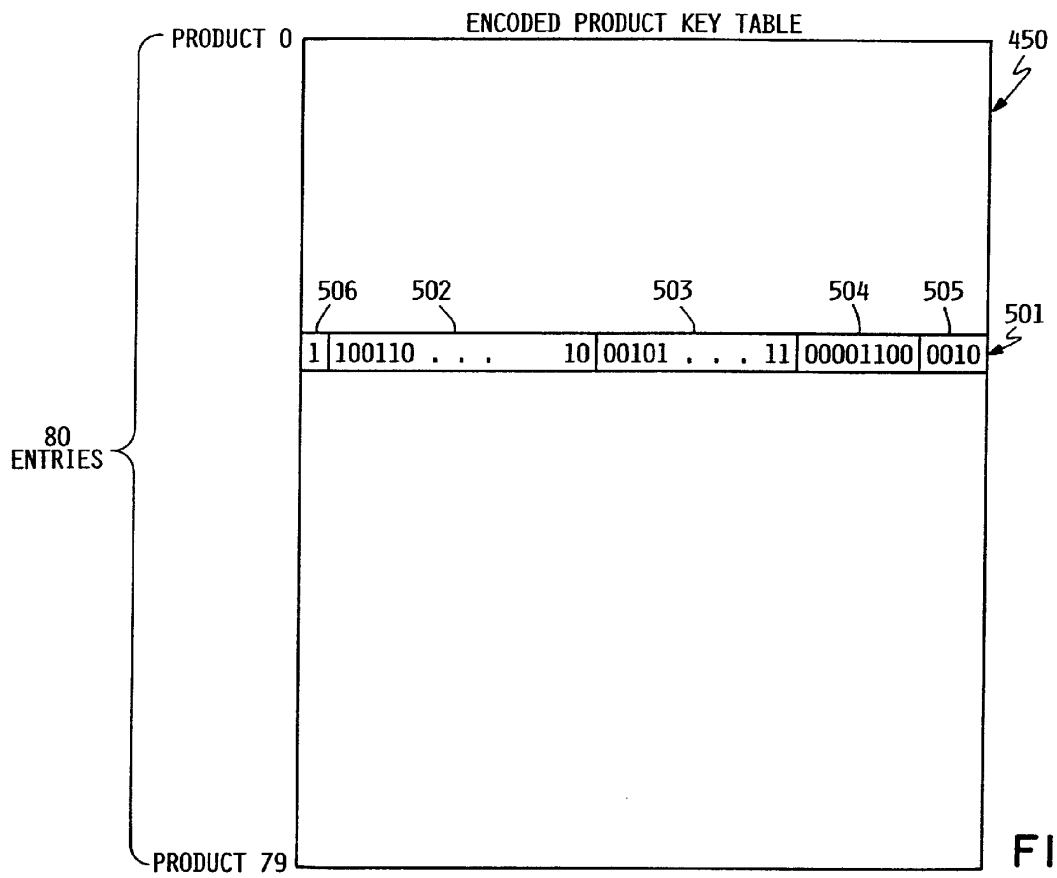


FIG. 5

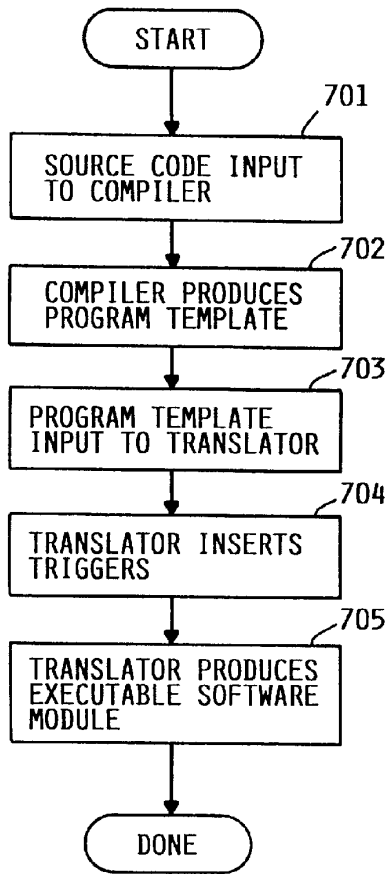


FIG. 7

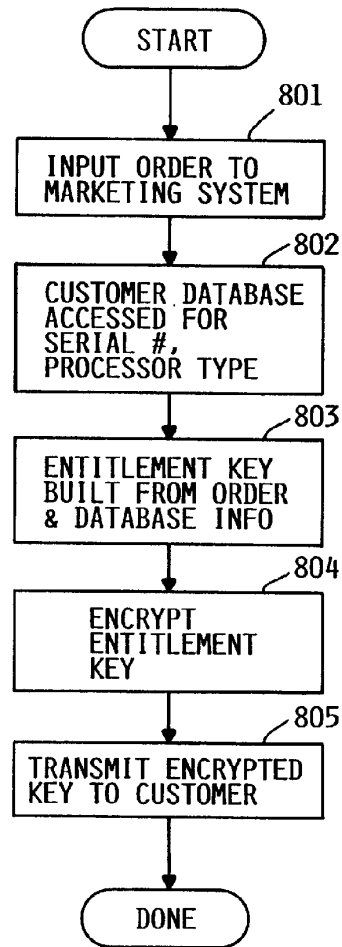


FIG. 8

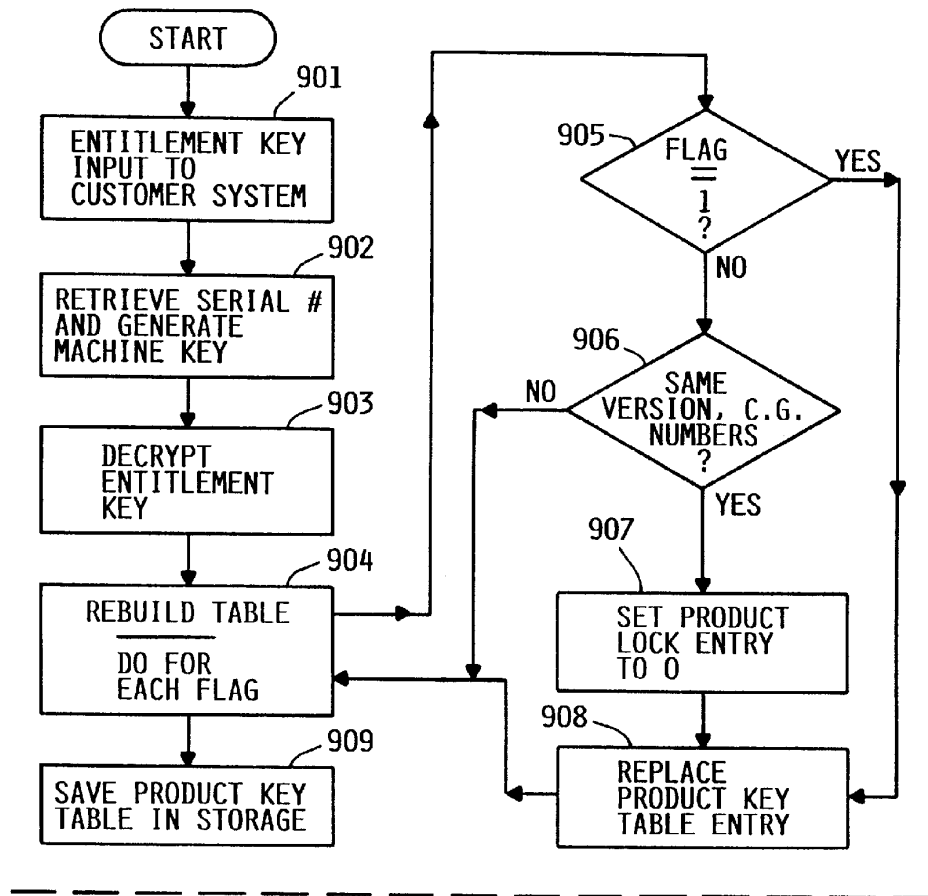


FIG. 9a

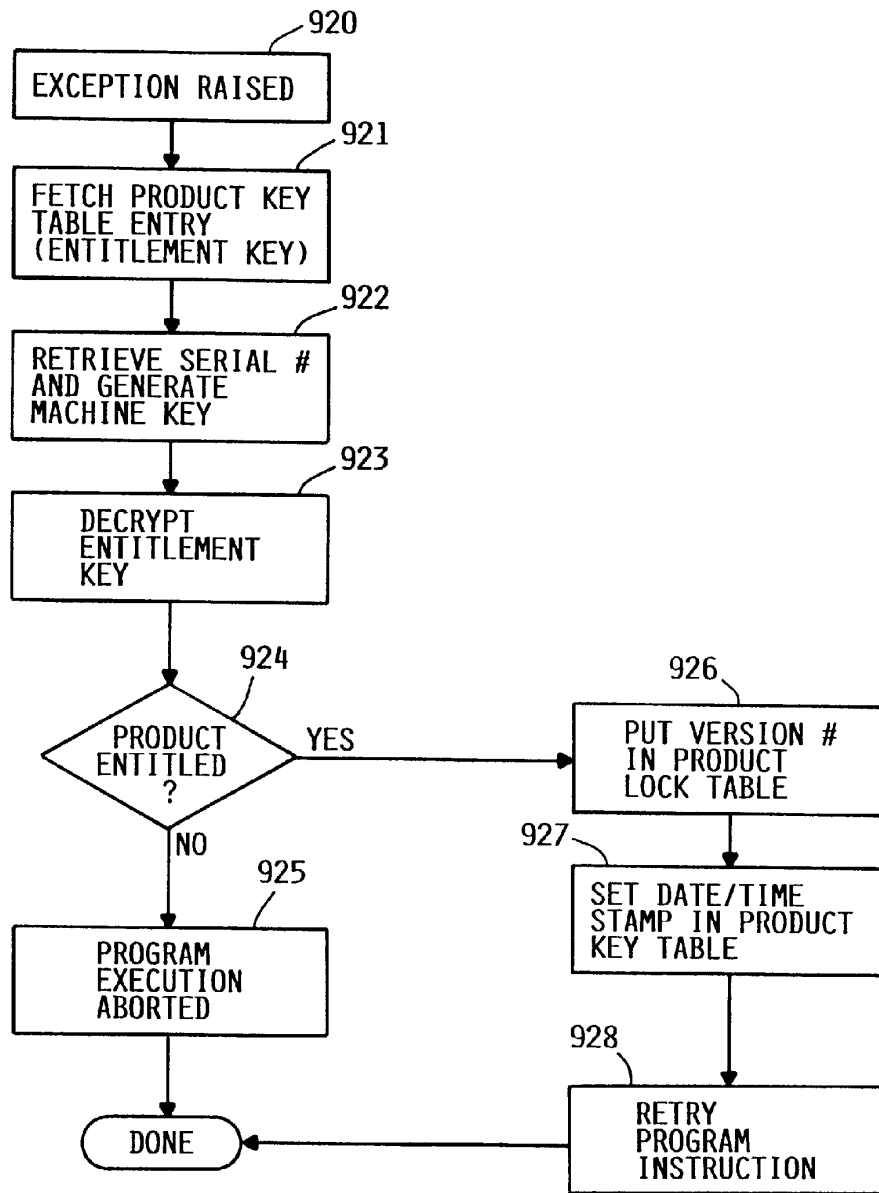


FIG. 9b

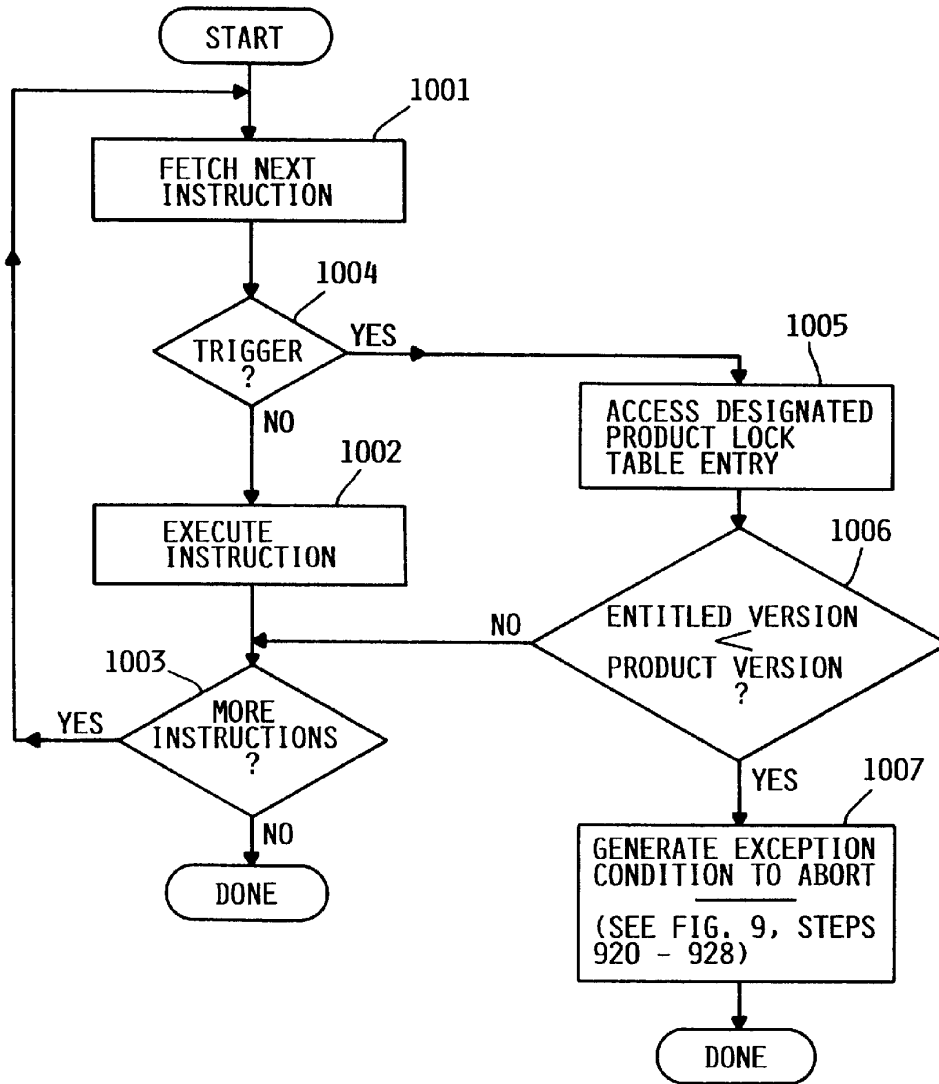


FIG. 10

APPARATUS AND METHOD FOR CONTROLLING ACCESS TO SOFTWARE

This application is a continuation of Ser. No. 07/629,295, filed Dec. 14, 1990, now abandoned.

FIELD OF THE INVENTION

The present invention relates to computer software usage, and in particular to restricting the ability of a computer user to use licensed software in a manner inconsistent with the license.

BACKGROUND OF THE INVENTION

A modern computer system is an enormously complex machine, incorporating the results of countless hours of engineering and programming skill in its design. A computer system contains many elements, but these may generally be broken down into hardware and software. Hardware is the tangible circuits, boards, cables, storage devices, enclosures, etc. that make up the system. But the hardware, in and of itself, can not solve problems in the real world, any more than a typewriter can compose a Pulitzer-winning play. The hardware requires instructions which tell it what to do. In its purest form, "software" refers to these instructions which make the hardware perform useful work (although it is sometimes loosely applied to the media on which software is stored and distributed). Like hardware, software is the product of human ingenuity. High quality software requires considerable creativity, training and intelligence on the part of the creator, also known as the programmer. Universities offer courses of instruction in "computer science" and similar disciplines, for the purpose of teaching people the art of creating software. An entire industry, encompassing thousands of careers, has grown up to create software which performs useful work. As a result of the extensive effort that can be expended in the development of software, it is not uncommon for the value of the software which is part of a computer system to exceed the value of the hardware.

One of the characteristics of a modern computer system is its ability to transmit and copy data with great speed and ease. In general, this is a necessary and beneficial capability. But it also means that software which may have taken years of creative effort to develop can be replicated in a fraction of a second on relatively inexpensive magnetic media, a capability which can be abused. Unauthorized persons can, with little cost or effort, replicate valuable software. This practice is generally known as software piracy. In recent years, various laws imposing criminal and civil liabilities have been enacted to curb software piracy. However, given the relative ease with which one may copy software, and the temptation to do so which arises from the value of the software, software piracy remains a problem.

In the case of inexpensive mass distributed software for small computers known as "personal computers", it is common to license the software for a fixed one-time charge which is the same for all customers. This is less practical for large mainframe computer systems. Software for such systems can involve millions of lines of code, requiring a very large investment to develop and maintain. A single fixed one-time charge sufficient for the developer to recoup this investment is likely to be prohibitively expensive for many smaller users. Therefore, it is common in the mainframe environment to charge license fees for software based on usage. On such method, known as "tiered pricing", involves charging according to a variable fee scale based on the capability of the customer's machine. The customer with a

faster machine, with more terminals attached, will pay a higher license fee for the same software than a customer with a less capable machine. Another common practice is to charge separately for maintenance upgrades of the software.

Given the level of expertise required to create quality software, and the amount of time and effort that must be expended, creation of such software requires money. If software developers are not paid proportionate to their training and effort, it will not be possible to find people to create software. It is not merely a practical imperative that the investment in software made by its developers be protected, but a moral one as well. Legitimate owners of software have therefore sought to develop ways to distribute software which will make it difficult to use the software in an unauthorized manner, and ensure that software developers are fairly compensated for their products.

Software may be distributed by the legitimate owner in a number of different ways. From the standpoint of preventing unauthorized use, these distribution methods may broadly be classified in three groups: unrestricted entitlement methods, restricted entitlement methods, and non-entitlement methods.

Unrestricted entitlement means that the software as distributed will run on any machine for which it was designed, without restriction. An owner distributing software with unrestricted entitlement must distribute to each user only the software which that user has paid for and is entitled to run. Apart from legal and contractual obligations, there is nothing to prevent the recipient of such software from copying or using it in an unauthorized manner. For inexpensive software which is licensed for a one-time charge, such as that used on most personal computers, this is the most common method of distribution.

Restricted entitlement means that the software contains some built-in restriction, limiting the ability of a user to copy and run it on any number of machines. There are several varieties of restricted entitlement methods. One of these is copy protection, which restricts the number of copies that can be made from the distributed original. While it achieves some level of protection from software piracy, copy protection has certain disadvantages. Like unrestricted entitlement, copy protection requires that the owner distribute to each user only the software that the user is entitled to run. It is not foolproof, and some programs exist which can make literal copies of copy protected software, defeating the protection mechanism. Finally, it interferes with the user's ability to make legitimate copies for backup purposes or to run the software from a fast storage device. Another restricted entitlement method is to encode user or machine specific information in the software itself. When the software is run, the machine will perform a check to make sure the software is authorized for that machine or user. This method achieves protection without interfering with the user's ability to make legitimate copies. However, it requires a very expensive distribution system, since each copy of the distributed software must be uniquely compiled, placed on distribution media, and shipped.

Non-entitlement means that the software as distributed is disabled, and requires a separately distributed authorization to be able to run. A non-entitlement method has the potential of avoiding customized software distribution entirely, although not all such methods have this capability. For example, the owner can distribute the same set of multiple software programs on a single generic medium to all its customers, and separately distribute individualized authorization keys to the each customer which allow the customer

to run only those programs he has paid for. While non-entitlement methods avoid many of the problems associated with other methods, current designs suffer a high exposure to fabricated entitlement or significant performance degradation. In most cases, the mechanism used to withhold entitlement to run the software requires that the provision for entitlement verification be centralized in the software module (either as data or instructions), to avoid performance degradation overhead of the verification. In some cases, this entitlement overhead is due to the size of the product identifier and the packaging of protection routines within the distributed software. In other cases, the overhead is due to the need to perform complicated decode procedures while running the software. As a result of this centralization of the entitlement check, it is relatively easy for an experienced programmer to nullify the protection mechanism by "patching", i.e., modifying a small, selected portion of the object code. In another case, the object code doesn't provide for entitlement verification, but a secure call path does, which identifies the module by producing a bit signature from it. While this avoids exposure to patching, it unavoidably causes severe performance degradation of the call mechanism.

The protection methods taught by prior art involve trade-offs between level of protection and ease of use. It is possible to obtain a relatively high level of protection by encoding machine specific information in the software, but at a cost of maintaining a very complex distribution system in which each distributed copy of the software is unique. Less costly distribution is possible, but at the expense of losing some of the protection. A need exists for a method which achieves a high level of protection, can be easily distributed using mass distribution techniques, and does not unduly interfere with system performance, the user's ability to make legitimate back-up copies, or other necessary functions. At the same time, there is a need for a method which supports tiered pricing, and separate licensing fees for different versions of a software product.

It is therefore an object of the present invention to provide an enhanced method and apparatus for controlling the use of software in a computer system.

Another object of this invention is to provide a greater level of protection against unauthorized use of software in a computer system.

Another object of this invention is to reduce the cost of protecting software against unauthorized use.

Another object of this invention is to increase the performance of a computer system running software which is protected against unauthorized use.

Another object of this invention is to simplify the distribution system of a distributor of software protected against unauthorized use.

Another object of this invention is to provide a method and apparatus of protecting software from unauthorized use which reduces the impact of such protection on legitimate uses of the software.

Another object of this invention is to provide an enhanced method and apparatus of protecting software from unauthorized use while allowing the user to make legitimate back-up copies of the software.

Another object of this invention is to make it more difficult to alter software in a manner that will enable unauthorized use.

Another object of this invention is to provide an enhanced method and apparatus for distributing tier-priced software.

SUMMARY OF THE INVENTION

Software is distributed according to the present invention without entitlement to run. A separately distributed encrypted entitlement key enables execution of the software. The key includes the serial number of the machine for which the software is licensed, together with a plurality of entitlement bits indicating which software modules are entitled to run on the machine. A secure decryption mechanism is contained on the machine. The decryption mechanism fetches the machine serial number and uses it as a key to decrypt the entitlement key. The entitlement information is then stored in a product lock table in memory.

The distributed software contains a plurality of entitlement verification triggers. In the preferred embodiment, each trigger is a single machine instruction in the object code, identifying a product number of the software module. When such a triggering instruction is encountered during execution of the software module, the machine checks the product lock table entry corresponding to the product number of the software module. If the product is entitled to run, execution continues normally; otherwise execution is aborted. Because this verification involves only a single machine instruction, it can be done with virtually no impact to overall system performance. As a result, it is possible to place a substantial number of such entitlement verification triggers in the object code, making it virtually impossible for someone to alter the code by "patching" the triggers. In an alternative embodiment, the triggering instruction also performs some useful work necessary for the software module to properly execute. This renders the software even more difficult to "patch", and further reduces the impact to performance of such verification triggers.

Because the software itself does not contain any entitlement, no restrictions on the distribution are necessary. In the preferred embodiment, the distributor of software may record multiple software modules on a single generic medium, and distribute the same recorded set of modules to all its customers. Each customer will receive a unique entitlement key, enabling the customer to run only those software modules to which he is licensed. It makes no difference that the customer is given some modules to which he is not licensed, as he will be unable to execute them without the appropriate key. The customer may freely load the software onto other storage devices in his system, or make any number of back-up copies of the software.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 shows the major components of a software protection mechanism according to the preferred embodiment of this invention;

FIG. 2 shows the unencrypted contents of an entitlement key according to the preferred embodiment of this invention;

FIG. 3 shows the contents of a typical executable software module according to the preferred embodiment;

FIG. 4 shows the hardware and software structures required on the customer's computer system to support the software protection mechanism according to the preferred embodiment;

FIG. 5 shows the format of the encoded product key table according to the preferred embodiment;

FIG. 6 shows the format of the product lock table according to the preferred embodiment;

FIG. 7 is a block diagram of the steps required to place authority verification triggers in a software module, according to the preferred embodiment;

5

FIG. 8 is a block diagram of the steps required to generate an encrypted entitlement key according to the preferred embodiment;

FIG. 9 is a block diagram of the steps required to decode an entitlement key and maintain a record of entitlement status on the customer's computer system, according to the preferred embodiment;

FIG. 10 is a block diagram of the steps required to verify entitlement during execution of a software module according to the preferred embodiment.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

A diagram of the major components of a software protection mechanism according to the preferred embodiment of the present invention is shown in FIG. 1. A customer's computer system 101 comprises a central processing unit (CPU) 102 tightly coupled to control store 103, a random access system memory 104, a non-erasable electronically readable unique identifier 105, and a plurality of storage devices 106, 107, 108. In the preferred embodiment, unique identifier 105 is a machine serial number. In the preferred embodiment, storage devices 106-108 are rotating magnetic disk drive storage units, although other storage technologies could be employed. Computer system 101 also comprises an operator console 109 through which it can receive input from an operator, and a unit for reading software media 110. While one operator console, one unit for reading software media, and three storage units are shown in FIG. 1, it should be understood that the actual number of such devices attached to system 101 is variable. It should also be understood that additional devices may be attached to system 101. In the preferred embodiment, computer system 101 is an IBM AS/400 computer system, although other computer systems could be used.

Software modules are distributed separately from the entitlement keys required to operate them. The software modules are originally created on development computer system 125, which contains compiler 126 and translator 127. The software modules are recorded on software recording media 112 and stored in warehouse 120, from which they are distributed to the customer. Encrypted entitlement key 111 is distributed from a sales office 121 of the distributor. Sales office 121 has access to marketing computer system 124, comprising an entitlement key generator/encrypter 122 and a database 123 containing customer information. In particular, database 123 contains machine serial numbers and processor types necessary for generation of the encrypted entitlement key. In the preferred embodiment, marketing system 124 is a centrally located computer communicating with a plurality of sales offices. However, system 124 could in the alternative be located at the sales office itself, accessing a local or centralized database. While two separate computer systems 124, 125 under the control of the distributor are shown, it should be understood that systems 124 and 125 could physically be a single computer system performing both functions.

Encrypted entitlement key 111 is sent from the software distributor to the customer by mail, telephone, or other appropriate means. While it is possible to transmit the key electronically or on magnetic media such as a diskette, the key is sufficiently brief that an operator can enter it into system 101 by typing the key on console 109.

In the preferred embodiment, multiple software modules are distributed on the same media 112. The distributor may independently grant entitlement to use each module via the

6

entitlement key. Warehouse 120 stocks a generic set of software modules on media 112. Each customer is shipped the same generic set of software modules, irrespective of which ones the customer is licensed to use. The separately distributed entitlement key contains information enabling system 101 to determine which software modules are entitled to execute on it.

In the preferred embodiment, software media 112 comprise one or more optical read/only disks, and unit 110 is an optical disk reader, it being understood that electronic distribution or other distribution media could be used. Upon receipt of software media 112, the customer will typically load the desired software modules from unit 110 into system 101, and store the software modules on storage devices 106-108. The customer may create one or more back-up copies of the software modules on any suitable media, and store these in an archive. Software media 112 contains no restriction on copying or loading into the system, and is freely copyable and loadable.

The contents of entitlement key 200 before encryption according to the preferred embodiment are shown in FIG. 2. The key contains charge group field 201, software version field 202, key type field 203, machine serial number field 204, and product entitlement flags 205. Charge group 201 specifies one of 16 possible machine tier values, and is used for supporting tier pricing of software. Software version 202 specifies the version level of the software which is entitled. It is anticipated that separate charges may be imposed for maintenance upgrades of software. The version 202 specified in the key 200 will entitle software at that version level and all previous (lower) levels. Key type field 203 is a reserved area for future changes to the key format, key chaining, or for an extension of the number of different product supported. Machine serial number field 204 contains the serial number of the machine for which the entitlement key is intended. Product entitlement flag 205 is an 80-bit field containing 80 separate product flags, each corresponding to a product number. The bit is set to '1' if the corresponding product number is entitled; otherwise it is set to '0'.

In the preferred embodiment, software modules are distributed as compiled object code. A typical software module 300 is shown in FIG. 3. The software module comprises a plurality of object code instructions capable of executing on computer system 101. According to this invention, a number of entitlement verification triggering instructions 301 are embedded in the object code. All triggering instructions 301 contained within a software module are identical. Triggering instruction 301 comprises operation code field 302, version field 303, and product number field 304. Field 305 is unused. Operation code 302 is the verb portion of the object code instruction, identifying the operation to be performed. Version 303 identifies the version level of the software module. Product number 304 identifies the product number associated with the software module. A single operation code is used for all triggering instructions, although the version and product number information may vary from module to module. In an alternative embodiment, the triggering instruction is also a direct instruction to perform some other useful work (from among those instructions which do not require that an operand for the action be specified in the instruction). In this alternative embodiment, execution of the triggering instruction causes system 101 to perform some other operation simultaneous with the entitlement verification.

Computer system 101 contains means for receiving and decoding encrypted entitlement key 111 as well as means for

verifying that the system has entitlement to execute a software module in response to a triggering instruction. In the preferred embodiment, these functions are divided among different levels of system hardware and software as shown in FIG. 4. In this embodiment, system 101 comprises four levels of hardware/software function: hardware level 401, horizontal microcode level 402, executable code level 403, and virtual machine level 404. Machine interface 405 separates the virtual machine level from all lower levels. Machine interface 405 is the lowest level interface defined to the customer, i.e., the instruction set at the virtual machine level is defined to the customer, but operations at lower levels are not. The customer therefore lacks capability to directly alter instructions below the machine interface level. A permanent, unalterable machine serial number 410 is contained at hardware level 401.

Horizontal microcode 402 contains microcode entries interpreting the executable instruction set. It is physically stored in control store 103, which in the preferred embodiment is a read-only memory (ROM) which is not capable of alteration by the customer. Entries in horizontal microcode support get machine key 420, set lock 421, and check lock 422 functions. Get machine key function 420 fetches a unique identifier, which in the preferred embodiment is based on the system serial number, from some permanent hardware location. Set lock function 421 accesses product lock table 460 and alters an entry in the table. The set lock function is the only microcode function capable of altering product lock table 460. Check lock function 422 accesses product lock table 460 and reads one of the entries to verify entitlement.

Executable code level 403 contains low level supervisory support and the support which implements instructions defined at the machine interface. It is physically stored in memory, but because the internal specifications are not defined to the customer, it is for all practical purposes not capable of being altered by the customer. Low level support includes unlock routine 430, IPL reverify locks routine 431, and exception handler 432. Unlock routine 430 uses the unique machine key to decode entitlement key 111, and stores encrypted entitlement key 111 in encoded product key table 450. IPL reverify locks routine 431 fetches the contents of encoded product key table 450 to rebuild product lock table 460 when the system is re-initialized. Exception handler 432 responds to exception conditions raised by functions in horizontal microcode 402. Executable code level 403 additionally contains the object code form of software module 300.

Virtual machine level 404 refers to software as it is represented in terms of machine instructions. The machine at this level acts as a virtual machine in the sense that the machine instructions are not directly executable. Because the machine level is the lowest level of interface available to a customer, a nontraditional compilation path is required to produce an object code form of a module which can be executed on the system. This compilation process is described in a later paragraph. Although, in the truest sense, software which is produced through this nontraditional compilation path must take form as executable object code to be executed, it is normally and more clearly discussed in terms of its representation in machine instructions at the virtual machine level as if the machine instructions were directly executable. With this established, we can then say that virtual machine level 404 contains compiled user software. It also includes the higher level operating system support for system 101. This operation system support at virtual machine level 404 contains two user interface rou-

ties needed to support input of the entitlement key. General input routine 441 is used to handle input during normal operations. In addition, special install input routine 440 is required to input the key during initial installation of the operating system. This is required because that part of the operating system above machine interface level 405 is treated for purposes of this invention as any other program product; it will have a product number and its object code will be infected with entitlement verification triggers. Install input routine 440 is the only part of the operating system which will not have entitlement verification triggers, thereby allowing the entitlement key to be input when the system is originally installed.

Software module 300 is part of a program product in compiled object code form which executes on system 101. It exists as an entity in virtual machine level 404 in the sense that it is accessible to other objects in this level. However, the actual executable code operates at executable code level 403, as shown by the box in broken lines. The executable code contains entitlement verification triggering instructions 301 (only one shown), which are executed by horizontal microcode check lock function 422.

Encoded product key table 450 is shown in FIG. 5. The table is contained in random access memory 104, and duplicated on a non-volatile storage device so it can be recovered if the system must be powered down or otherwise re-initialized. Table 450 contains 80 entries 501, one corresponding to each possible product number. Each entry 501 comprises a complete copy of the encrypted entitlement key 502 applicable to the product number of the entry, a date/time stamp 503 indicating when the key was first used, a version number 504 of the key, a charge group 505 of the key, and an entitlement bit 506 indicating whether the key unlocks the product. Date/time stamp 503 may be encrypted. Version number 504, charge group 505 and entitlement bit 506 repeat information contained in encrypted key 501. They are contained in the table to support queries. However, no entry in product lock table 460 can be altered, granting entitlement for a program to execute, without first verifying the information in encrypted key 502. Because each key 502 in product key table 450 is in its encrypted form, no special measures are required to prevent user access to the table.

Product lock table 460 is shown in FIG. 6. This table is contained in a special low address range of random access memory 104, which is under complete control of horizontal microcode. Table 460 contains 80 entries 601, one corresponding to each possible product number. Each entry contains a version number indicating the maximum version level of entitlement. A version number of 0 indicates no entitlement to any version of the product. The version number may be scrambled to add another degree of protection against alteration of table 460.

The operation of a software module on computer system 101 in accordance with the preferred embodiment of the present invention will now be described. There are four parts to this operation. In the first part, a plurality of entitlement verification triggering instructions are placed in the executable object code form of the software module. In the second part, an encrypted entitlement key 111 authorizing access to the software module is generated. In the third part, computer system 101 receives, decodes and stores entitlement key 111, and sets product lock table 460. In the fourth part, the software module executing on system 101 causes system 101 to verify entitlement upon encountering an entitlement verification instruction. The first two parts are performed under the control of the software distributor. The last two are performed on the customer's system 101.

When the software distributor compiles a software module, it must place the entitlement verification triggers in object code. A typical such process, which takes place on development system 125, is shown in FIG. 7. Source code is generated by a programmer in the normal fashion, without inclusion of entitlement verification triggers. The source code is input into compiler 126 at step 701 to produce a program template at 702. The program template comprises machine instructions at virtual machine level 404 (i.e. above machine interface 405). The program template serves as input to translator 127 at step 704, along with its product number and version number identification. Translator 127 automatically generates a substantial number of entitlement verification triggers, inserts them in random locations in the object code, and resolves references after the triggers have been inserted. The resultant executable object code form of the software module which is output at 705 contains the embedded triggers. This executable form of the module comprises object code instructions at executable code level 403.

The process for generating an encrypted entitlement key is shown in FIG. 8. An order for licenses to one or more program products at some version level generated by a sales representative is input to marketing computer system 124 at step 801. In theory, the customer could order products at multiple version levels, although there is generally little reason for him to do so. However, since each entitlement key operates at only a specified version level, a separate key would have to be generated for each different version level ordered by the customer. Upon receipt of the customer's order, key generator/encryptor 122 executing on system 124 would access database 123 containing information about the customer, particularly the serial number and processor type of his machine, at step 802. This information is used to generate charge group field 201 and machine serial number field 204 of the unencrypted entitlement key 200. The remaining fields are generated by reference to the customer order and a database of possible product number offerings, building the complete unencrypted key at step 803. Key generator/encryptor 122 then encrypts the key, using any of a number of encryption techniques known in the art, at step 804. The resultant encrypted entitlement key 111 is then transmitted to the customer at step 805. Although key 111 is shown in FIG. 1 as a plurality of binary bits, it may be presented to the customer in some other form, such as hexadecimal digits or alphanumeric equivalents of groups of binary bits, in order to simplify the task of entering the key from a keyboard.

The process for receiving entitlement key 111 and maintaining product lock table 460 on computer system 101 is shown in FIG. 9. A customer enters entitlement key 111 into computer system 101 via console 109 at step 901. If this is an initial installation, install input routine 440 interacts with the operator to receive the input; otherwise general input routine 441 receives the input. The entitlement key is passed to unlock routine 430, which handles the decoding process. Unlock routine 430 causes get machine key function 420 to retrieve the machine serial number and generate the machine key at 902. Unlock routine 430 then uses the machine key to decode the entitlement key 111 at step 903. Unlock routine then rebuilds encoded product key table 450 at step 904, as described below. The decoded entitlement key takes the form shown in FIG. 2. It includes an 80-bit product entitlement flag array 205 indicating, for each product number, whether the product is unlocked under the new entitlement key. The new entitlement key is viewed as a replacement key for all products it unlocks. Unlock routine

430 scans each product entitlement flag 205 in the decoded key (step 904). If the product entitlement flag is set to '1' (indicating entitlement) at step 905, the corresponding entry in product key table 450 is replaced with the new entitlement key, version number, and charge group value at step 908. The entitlement bit field 506 is set to '1', and the date/time stamp field 503 is set to a zero value to indicate that the entitlement key has not yet been used. If the product entitlement flag of the new key is '0' the new entitlement key has no effect unless the version number and charge group number are the same as those stored in product key table 450. If the version and charge group number are the same (step 906), the entitlement key has the effect of locking the product. Unlock routine 430 will therefore invoke set lock function 421 to set the version number in product lock table 460 to '0' at step 907, and replace the corresponding entry in product key table 450 with the new entitlement key values at step 908. When table 450 has been rebuilt, its contents are saved in storage at step 909.

Unless a previously entitled product is unentitled, the rebuilding of table 450 has no immediate effect on product lock table 460. Products are unlocked on demand. Upon first execution of a previously unentitled software product, an exception is generated by the system when a triggering instruction is encountered. Exception handling routine 432 then calls unlock routine 430 to attempt to unlock the product at step 920. Unlock routine 430 then fetches the encrypted entitlement key from the appropriate entry in encoded product key table 450 at step 921, obtains the machine key at step 922, and decodes the entitlement key at step 923. If entitlement is indicated (step 924), set lock function 421 is invoked to set the version number in the entry 601 in product lock table 460 corresponding to the product number of the software product at step 926. At the same time, unlock routine 430 records the date and time of first use in date/time stamp field 503 at step 927. The triggering instruction is then retried and program execution continues at step 928. If entitlement is not indicated at step 924, program execution aborts at step 925.

Product lock table 460, being stored in RAM, will not survive system re-initialization. During re-initialization ("IPL"), IPL reverify locks routine 431 is called, to rebuild the product lock table. This routine will get the machine key as described above, and systematically decode each entitlement key entry in encoded product key table 450 to verify entitlement and rebuild the corresponding entry in product lock table 460.

The process for executing a software module according to the preferred embodiment is shown in FIG. 10. System 101 executes the module by fetching (step 1001) and executing (step 1002) object code instructions until done (step 1003). If any instruction is an entitlement verification triggering instruction 301 (step 1004) check lock function 422 is invoked. Check lock function 422 accesses the product lock table entry 601 corresponding to the product number contained in the triggering instruction at step 1005. If the version number in product lock table 460 is equal to or greater than the version number 303 contained in triggering instruction 301, the software is entitled to execute (step 1006). In this case, check lock function 422 takes no further action, and the system proceeds to execute the next object code instruction in the software module. If the software is not entitled, check lock function 422 generates an exception condition, causing control to pass to exception handler 432, which will terminate program execution (step 1007). The system does not save the results of an entitlement check which shows that the software is entitled. Therefore, when

a triggering instruction is again encountered in the software module, the system again verifies entitlement as described above.

In alternative embodiments, additional sophistication could be defined for the entitlement verification triggering instructions to enable their being injected into the object code through a traditional compilation path. This would be one where the machine interface available to customers and compiler writers is that same set of executable instructions in which the object code form of a module is executed by the system. For support of such a traditional compilation path where the object code format is known by customers, additional barriers to patching of the object code to nullify or alter the entitlement triggering instructions may be appropriate. One such additional barrier would be to define the entitlement triggering instruction to simultaneously perform some other function. In this case, it is critical that the alternative function performed by the triggering instruction can not be performed by any other simple instruction. The alternative function must be so selected that any compiled software module will be reasonably certain of containing a number of instructions performing the function. If these criteria are met, the compiler can automatically generate the object code to perform the alternative function (and simultaneously, the entitlement verification trigger) as part of its normal compilation procedure. This definition would provide a significant barrier to patching of the object code to nullify the entitlement triggering instructions. Another alternative embodiment is to define that the entitlement triggering instruction must be positioned in the object code with an addressability alignment that has a simple relationship to the product number that it identifies. It should be a relatively simple matter for a compiler to inject the instructions at the proper code alignment as part of the normal compilation process and a simple matter for the instruction implementation to perform this additional verification. This definition would provide a significant barrier to patching of the object code to alter the identity of the product number supplied in the entitlement triggering instructions.

In the preferred embodiment, provision is made for 80 independent product numbers. It should be understood that the actual number of product numbers supported according to this invention is variable. It is anticipated in the preferred embodiment that the number of separately compilable software modules distributed by the distributor may well exceed 80 by a large factor. The number of product numbers corresponds to the number of separately priced software packages offered by the distributor's marketing organization. While each software module has only one product number, there may be many software modules sharing that product number. For example, the distributor may offer a word processing package, which includes separate software modules to handle screen editing, spell checking, document formatting, etc. If these software modules are always licensed as part of the word processing package, they would have a common product number. In an alternative embodiment, it would be possible to have a separate number for each software module.

In the preferred embodiment, software is licensed for a one-time charge, although additional charges can be made for maintenance upgrades. In an alternative embodiment, software can be licensed for a period of time. In this alternative embodiment, the entitlement key will contain an additional field indicating a length of time for which software is licensed. IPL reverify locks routine 431 will be called periodically to compare the date/time stamp associated with a product number in encoded product key table

450 with the length of time of the license, to determine whether the license has expired. If a license has expired, the corresponding unlock bit in product lock table 460 will be set to '0', preventing further execution of the software until additional entitlement is obtained.

Although a specific embodiment of the invention has been disclosed along with certain alternatives, it will be recognized by those skilled in the art that additional variations in form and detail may be made within the scope of the following claims.

What is claimed is:

1. An apparatus for controlling the use of a software module executing on a computer system, said computer system comprising:

means for granting entitlement for said computer system to execute said software module, said software module being a program unit that is discrete and identifiable with respect to compiling, combining with other units, and loading;

a plurality of independent triggering means in said software module for triggering entitlement verification;

entitlement verification means, responsive to each of said plurality of independent triggering means, for verifying that said computer system has entitlement to execute said software module; and

means, responsive to said entitlement verification means, for aborting execution of said software module if said entitlement verification means determines that said computer system lacks entitlement to execute said software module.

2. The apparatus for controlling the use of a software module of claim 1, wherein each of said plurality of independent triggering means is a single object code instruction which triggers said entitlement verification means.

3. The apparatus for controlling the use of a software module of claim 2, wherein said object code instruction which triggers said entitlement verification means also causes said computer system to perform an additional function required for proper execution of said software module concurrently with verifying that said computer system has entitlement to execute said software module, whereby if said software module is modified by removing said object code instruction, said additional function required for proper execution of said software module will not be performed.

4. The apparatus for controlling the use of a software module of claim 2, wherein said object code instruction which triggers said entitlement verification means contains a product number for said software module.

5. The apparatus for controlling the use of a software module of claim 4, wherein said entitlement verification means comprises:

a product lock table in said computer system, said product lock table comprising a plurality of entries containing entitlement information, wherein each of said plurality of entries is associated with a product number; and

means, responsive to said object code instruction, for accessing entitlement information contained in an entry in said product lock table associated with said product number contained in said object code instruction.

6. The apparatus for controlling the use of a software module of claim 1, wherein said means for granting entitlement for said computer system to run said software module comprises:

means for generating an entitlement key consisting of data; and

means for inputting said entitlement key into said computer system.

13

7. The apparatus for controlling the use of a software module of claim 6, wherein said computer system contains a unique identifier, wherein said means for generating an entitlement key uses said unique identifier to generate said entitlement key, and wherein said entitlement key grants entitlement to run software only on a computer system containing the same unique identifier.

8. The apparatus for controlling the use of a software module of claim 7, further comprising:

means for encrypting said entitlement key using said unique identifier, wherein the resultant encrypted entitlement key may be decrypted only on a computer system having the same unique identifier;

means in said computer system for accessing the unique identifier of said computer system; and

means in said computer system, responsive to said means for accessing said unique identifier, for decrypting said encrypted entitlement key.

9. A method for controlling the use of a software module executing on a computer system, said software module comprising a plurality of object code instructions, said method comprising the steps of:

granting entitlement for said computer system to execute said software module, said software module being a program unit that is discrete and identifiable with respect to compiling, combining with other units, and loading;

placing in said software module a plurality of independent triggering means for triggering entitlement verification; executing object code instructions contained in said software module;

triggering an entitlement verification in said computer system to verify that said computer system has entitlement to execute said software module whenever one of said plurality of independent triggering means is encountered during execution of said object code instructions contained in said software module;

aborting execution of said software module if said entitlement verification determines that said computer system lacks entitlement to execute said software module.

10. The method for controlling the use of a software module of claim 9, wherein said step of placing in said software module a plurality of independent triggering means comprising placing, at each of a plurality of separate locations in said software module, a single object code instruction which triggers said entitlement verification.

11. The method for controlling the use of a software module of claim 10, wherein said object code instruction which triggers said entitlement verification also causes said computer system to perform an additional function required for proper execution of said software module concurrently with verifying that said computer system has entitlement to execute said software module, whereby if said software module is modified by removing said object code instruction, said additional function required for proper execution of said software module will not be performed.

12. The method for controlling the use of a software module of claim 10, wherein said object code instruction which triggers said entitlement verification contains a product number for said software module.

13. The method for controlling the use of a software module of claim 12, further comprising the step of:

maintaining a product lock table in said computer system, said product lock table comprising a plurality of entries containing entitlement information, wherein each of said plurality of entries is associated with a product number; and

14

wherein said step of triggering an entitlement verification comprises accessing entitlement information contained in an entry in said product lock table associated with said product number contained in said object code instruction.

14. The method for controlling the use of a software module of claim 9, further comprising the steps of:

generating an entitlement key consisting of a plurality of data bits, said entitlement key providing information enabling said computer system to determine whether or not it has entitlement to execute said software module; and

inputting said entitlement key into said computer system.

15. The method for controlling the use of a software module of claim 14, wherein said computer system contains a unique identifier, wherein said step of generating an entitlement key uses said unique identifier to generate said entitlement key, and wherein said entitlement key grants entitlement to run software only on a computer system containing the same unique identifier.

16. A program product apparatus for controlling entitlement, wherein said program product apparatus executes on a computer system having means for receiving entitlement to execute a software module, and having entitlement verification means responsive to triggering means in said software module for verifying that said computer system has entitlement to execute said software module, said program product apparatus comprising:

at least one software module recorded on recording media, said software module being a program unit that is discrete and identifiable with respect to compiling, combining with other units, and loading; and

a plurality of independent triggering means in said software module for triggering said entitlement verification means on said computer system.

17. The program product apparatus for controlling entitlement of claim 16, wherein each of said plurality of independent triggering means is a single object code instruction which triggers said entitlement verification means.

18. The program product apparatus for controlling entitlement of claim 17, wherein said object code instruction which triggers said entitlement verification means also causes said computer system to perform an additional function required for proper execution of said software module concurrently with verifying that said computer system has entitlement to execute said software module, whereby if said software module is modified by removing said object code instruction, said additional function required for proper execution of said software module will not be performed.

19. The program product apparatus for controlling entitlement of claim 17, wherein:

said object code instruction which triggers said entitlement verification means contains a product number for said software module;

said entitlement verification means on said computer system comprises a product lock table in said computer system, said product lock table comprising a plurality of entries containing entitlement information, wherein each of said plurality of entries is associated with a product number; and

said entitlement verification means on said computer system further comprises means, responsive to said object code instruction, for accessing entitlement information contained in an entry in said product lock table associated with said product number contained in said object code instruction.

15

20. A method for distributing a software module, wherein said software module is capable of executing on any one of a plurality of computer systems, each of said computer systems having means for receiving entitlement to execute said software module, and having entitlement verification means responsive to triggering means in said software module for verifying that said computer system has entitlement to execute said software module, said method comprising the steps of:

- placing in said software module a plurality of independent triggering means for triggering said entitlement verification means, said software module being a program unit that is discrete and identifiable with respect to compiling, combining with other units, and loading;
- distributing a copy of said software module to each of said computer systems; and
- granting entitlement for at least one of said computer systems to execute said software module.

21. The method for distributing a software module of claim 20, wherein said step of placing in said software module a plurality of independent triggering means comprises placing, at each of a plurality of separate locations in said software module, a single object code instruction which triggers said entitlement verification means.

22. The method for distributing a software module of claim 21, wherein said object code instruction which triggers said entitlement verification also causes said computer system to perform an additional function required for proper execution of said software module concurrently with verifying that said computer system has entitlement to execute said software module, whereby if said software module is modified by removing said object code instruction, said additional function required for proper execution of said software module will not be performed.

23. The method for distributing a software module of claim 21, wherein:

- said object code instruction which triggers said entitlement verification means contains a product number for said software module;
- said entitlement verification means on said computer system comprises a product lock table in said computer

16

system, said product lock table comprising a plurality of entries containing entitlement information, wherein each of said plurality of entries is associated with a product number; and

said entitlement verification means on said computer system further comprises means, responsive to said object code instruction, for accessing entitlement information contained in an entry in said product lock table associated with said product number contained in said object code instruction.

24. The method for distributing a software module of claim 20, wherein said step of granting entitlement for said computer system to execute said software module comprises the steps of:

- generating an entitlement key consisting of a plurality of data bits, said entitlement key providing information enabling said computer system to determine whether or not it has entitlement to execute said software module; and
- transmitting said entitlement key to said computer system.

25. The method for controlling the use of a software module of claim 24, wherein said computer system contains a unique identifier, wherein said step of generating an entitlement key uses said unique identifier to generate said entitlement key, and wherein said entitlement key grants entitlement to run software only on a computer system containing the same unique identifier.

26. The method for controlling the use of a software module of claim 20, wherein:

- said step of distributing said software module comprises distributing a plurality of software modules, each having a plurality of independent triggering means for triggering entitlement verification, on a single recording medium; and
- said step of granting entitlement for said computer system to execute said software module comprises granting separate entitlement to at least two of said plurality of software modules on said single recording medium.

* * * * *

Exhibit 4

(19)日本国特許庁(J P)

(12) 公開特許公報(A)

(11)特許出願公開番号

特開平5-334072

(43)公開日 平成5年(1993)12月17日

(51)Int.Cl.⁵

識別記号 庁内整理番号

F I

技術表示箇所

G 0 6 F 9/06

4 5 0 C 7232-5B

審査請求 有 請求項の数26(全 20 頁)

(21)出願番号 特願平3-308350

(22)出願日 平成3年(1991)10月29日

(31)優先権主張番号 6 2 9 2 9 5

(32)優先日 1990年12月14日

(33)優先権主張国 米国(U S)

(71)出願人 390009531
 インターナショナル・ビジネス・マシーンズ・コーポレーション
 INTERNATIONAL BUSINESS MACHINES CORPORATION
 アメリカ合衆国10504、ニューヨーク州アーモンク(番地なし)

(72)発明者 ロバート・カール・ビーチャー
 アメリカ合衆国55904、ミネソタ州ロチェスター、17番ストリート、サウス・イースト 910番地

(74)代理人 弁理士 頼宮 孝一 (外4名)
 最終頁に続く

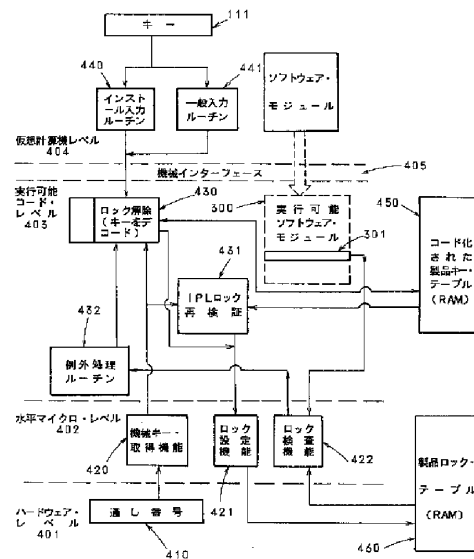
(54)【発明の名称】 ソフトウェアの使用を管理するための装置及び方法

(57)【要約】

【目的】本発明の目的は、コンピュータ・システムにおけるソフトウェアの使用を管理する改善された方法及び装置を提供することにある。

【構成】ソフトウェアは、実行する資格付与なしで配布され、別途配布される暗号化された資格付与キーによって、はじめてそのソフトウェアの実行が可能になる。このキーは、ソフトウェアのライセンスが与えられるコンピュータの通し番号と、どのソフトウェア・モジュールが機械上で走行する資格を付与されているかを示す、複数の資格付与ビットを含んでいる。配布されたソフトウェアは、複数の資格検証トリガを含む。各トリガは、ソフトウェア・モジュールの製品番号を識別する、目的コード形式の単一の機械語命令である。

【効果】本発明によって、コンピュータ・システムにおけるソフトウェアの使用を管理する、改善された方法及び装置が提供される。



【特許請求の範囲】

【請求項1】コンピュータ・システムにソフトウェア・モジュールを実行する資格を付与する手段と、上記ソフトウェア・モジュール内に配置された、資格の検証をトリガする複数の独立したトリガ手段と、上記の複数の独立したトリガ手段のそれぞれに 대응して、上記コンピュータ・システムが上記ソフトウェア・モジュールを実行する資格をもつことを検証する資格検証手段と、

上記の資格検証手段に 대응して、上記コンピュータ・システムが上記のソフトウェア・モジュールを実行する資格をもたないと上記資格検証手段が判定した場合に、上記ソフトウェア・モジュールの実行を打ち切る手段とを備える、コンピュータ・システムで実行されるソフトウェア・モジュールの使用を管理するための装置。

【請求項2】上記の複数の独立したトリガ手段が、上記資格検証手段をトリガする単一の目的コード命令である、請求項1に記載の、ソフトウェア・モジュールの使用を管理するための装置。

【請求項3】上記資格検証手段をトリガする上記目的コード命令が、上記ソフトウェア・モジュールの適切な実行に必要な追加ステップをも実行し、その結果、上記目的コード命令を除去することによって上記ソフトウェア・モジュールが修正される場合は、上記の追加ステップが実行されず、上記ソフトウェア・モジュールが適切に実行されなくなる、請求項2に記載の、ソフトウェア・モジュールの使用を管理するための装置。

【請求項4】上記資格検証手段をトリガする上記目的コード命令が、上記のソフトウェア・モジュールの製品番号を含む、請求項2に記載の、ソフトウェア・モジュールの使用を管理するための装置。

【請求項5】上記資格検証手段が、資格付与情報を含みかつそれぞれが製品番号に関連する複数のエントリを備える、上記コンピュータ・システム内の製品ロック・テーブルと、上記目的コード命令に 対応して、上記目的コード命令に含まれる上記製品番号に関連する上記製品ロック・テーブル中のエントリに含まれる上記資格付与情報にアクセスする手段とを含む、請求項4に記載の、ソフトウェア・モジュールの使用を管理するための装置。

【請求項6】上記のコンピュータ・システムに上記のソフトウェア・モジュールを実行する資格を付与する上記手段が、

データから構成される資格付与キーを生成する手段と、上記の資格付与キーを上記コンピュータ・システムに入力する手段とを備える、請求項1に記載の、ソフトウェア・モジュールの使用を管理するための装置。

【請求項7】上記コンピュータ・システムが独自の識別子を含み、

資格付与キーを生成する上記手段が、上記の独自の識別

子を用いて上記資格付与キーを生成し、

上記資格付与キーが、同じ独自の識別子を含むコンピュータ・システム上でのみソフトウェアを実行する資格を与える、

請求項6に記載の、ソフトウェア・モジュールの使用を管理するための装置。

【請求項8】上記の独自の識別子を用いて上記資格付与キーを暗号化し、その結果得られる暗号化された資格付与キーが同じ独自の識別子をもつコンピュータ・システム上でしか解読できないようにする手段と、

上記コンピュータ・システムの独自の識別子にアクセスする、コンピュータ・システム内の手段と、

上記の独自の識別子にアクセスする上記手段に 対応して、上記の暗号化された資格付与キーを解読する、上記のコンピュータ・システム内の手段とを備える、請求項7に記載の、ソフトウェア・モジュールの使用を管理するための装置。

【請求項9】資格の検証をトリガする複数の独立したトリガ手段をソフトウェア・モジュール内に配置する段階と、

上記ソフトウェア・モジュールを実行する段階と、

上記ソフトウェア・モジュールの実行中に上記複数の独立したトリガ手段の一つに出会ったとき、コンピュータ・システムがそのソフトウェア・モジュールを実行する資格をもつことを検証する資格検証動作を上記コンピュータ・システム中でトリガする段階と、

上記資格検証で、上記コンピュータ・システムには上記ソフトウェア・モジュールを実行する資格がないと判定された場合、上記ソフトウェア・モジュールの実行を打ち切る段階とを含む、コンピュータ・システム上で実行されるソフトウェア・モジュールの使用を管理するための方法。

【請求項10】複数の独立したトリガ手段を上記ソフトウェア・モジュール内に配置する上記段階が、上記ソフトウェア・モジュール内の複数の別々の位置に、上記資格検証動作をトリガする単一の目的コード命令を配置することを 含む、請求項9に記載の、ソフトウェア・モジュールの使用を管理するための方法。

【請求項11】上記資格検証動作をトリガする上記目的コード命令が、上記ソフトウェア・モジュールの適切な実行に必要な追加ステップをも実行し、その結果、目的コード命令を除去することによって上記ソフトウェア・モジュールが修正される場合は、上記の追加ステップが実行されず、上記ソフトウェア・モジュールが適切に実行されなくなる、請求項10に記載の、ソフトウェア・モジュールの使用を管理するための方法。

【請求項12】上記資格検証動作をトリガする上記目的コード命令が、上記ソフトウェア・モジュールの製品番号を含む、請求項10に記載の、ソフトウェア・モジュールの使用を管理するための方法。

【請求項13】資格付与情報を含みかつそれぞれが製品番号に関連する複数のエントリを備えた製品ロック・テーブルを、上記のコンピュータ・システム内で維持する段階を含み、資格検証動作をトリガする上記段階が、上記目的コード命令中に含まれる製品番号に関連する製品ロック・テーブル中のエントリに含まれる上記資格付与情報にアクセスすることを含む、

請求項12に記載の、ソフトウェア・モジュールの使用を管理するための方法。

【請求項14】複数のデータ・ビットから構成され、上記コンピュータ・システムが上記ソフトウェア・モジュールを実行する資格をもつかどうかを判定できるようにする情報を提供する資格付与キーを生成する段階と、上記資格付与キーを上記コンピュータ・システムに入力する段階とを含む、請求項9に記載の、ソフトウェア・モジュールの使用を管理するための方法。

【請求項15】上記コンピュータ・システムが独自の識別子を含み、

資格付与キーを生成する上記段階が、上記の独自の識別子を用いて上記資格付与キーを生成し、

上記資格付与キーが、同じ独自の識別子を含むコンピュータ・システム上でのみソフトウェアを実行する資格を与える、

請求項14に記載の、ソフトウェア・モジュールの使用を管理するための方法。

【請求項16】ソフトウェア・モジュールを実行する資格を受け取る手段を有し、かつ上記ソフトウェア・モジュール内のトリガ手段にตอบสนองして、上記コンピュータ・システムが上記ソフトウェア・モジュールを実行する資格をもつことを検証する資格検証手段を有するコンピュータ・システム上で、上記ソフトウェア・モジュールが実行される、資格の付与を管理するためのプログラム製品であって、記録媒体上に記録された少なくとも一個のソフトウェア・モジュールと、

コンピュータ・システム上で資格の検証をトリガする、上記ソフトウェア・モジュール中の複数の独立したトリガ手段とを備える、プログラム製品。

【請求項17】上記複数の独立したトリガ手段のそれぞれが、上記資格の検証をトリガする単一の目的コード命令である、請求項16に記載の、資格付与を管理するためのプログラム製品。

【請求項18】上記資格検証手段をトリガする上記目的コード命令が、上記ソフトウェア・モジュールの適切な実行に必要な追加ステップをも実行し、その結果、上記目的コード命令を除去することによって上記ソフトウェア・モジュールが修正される場合は、上記追加ステップが実行されず、ソフトウェア・モジュールが適切に実行されなくなる、請求項17に記載の、資格付与を管理するためのプログラム製品。

【請求項19】上記資格検証手段をトリガする上記目的コード命令が、上記ソフトウェア・モジュールの製品番号を含み、

上記コンピュータ・システム上の上記資格検証手段が、資格付与情報を含みかつそれぞれが製品番号に関連する複数のエントリを備えた製品ロック・テーブルをコンピュータ・システム内に有する、

請求項17に記載の、資格付与を管理するためのプログラム製品。

【請求項20】ソフトウェア・モジュールを実行する資格を受け取る手段を有し、かつ上記ソフトウェア・モジュール内のトリガ手段にตอบสนองして、上記コンピュータ・システムが上記ソフトウェア・モジュールを実行する資格をもつことを検証する資格検証手段を有するコンピュータ・システム上で、上記ソフトウェア・モジュールが実行される、ソフトウェア・モジュールを配布する方法であって、

資格の検証をトリガする複数の独立したトリガ手段を上記ソフトウェア・モジュール内に配置する段階と、

上記ソフトウェア・モジュールを上記コンピュータ・システムに配布する段階と、

上記コンピュータ・システムに上記ソフトウェア・モジュールを実行する資格を付与する段階とを含む、ソフトウェア・モジュールを配布する方法。

【請求項21】上記複数の独立したトリガ手段を上記ソフトウェア・モジュール内に配置する上記段階が、上記資格の検証をトリガする単一の目的コード命令を、上記ソフトウェア・モジュール中の複数の別々の位置に配置することを含む、請求項20に記載の、ソフトウェア・モジュールを配布する方法。

【請求項22】上記資格検証をトリガする上記目的コード命令が、上記ソフトウェア・モジュールの適切な実行に必要な追加ステップをも実行し、その結果、上記目的コード命令を除去することによって上記ソフトウェア・モジュールが修正される場合は、上記の追加ステップが実施されず、ソフトウェア・モジュールが適切に実行されなくなる、請求項21に記載の、ソフトウェア・モジュールを配布する方法。

【請求項23】上記資格検証をトリガする上記目的コード命令が、上記ソフトウェア・モジュールの製品番号を含み、

コンピュータ・システム上の上記資格検証手段が、資格付与情報を含みかつそれぞれが製品番号に関連する複数のエントリを備えた製品ロック・テーブルを、コンピュータ・システム内に有する、

請求項21に記載の、ソフトウェア・モジュールを配布する方法。

【請求項24】上記コンピュータ・システムに上記ソフトウェア・モジュールを実行する資格を付与する上記段階が、

複数のデータ・ビットから構成され、上記コンピュータ・システムが上記ソフトウェア・モジュールを実行する資格をもつかどうかを判定できるようにする情報を提供する、資格付与キーを生成する段階と、

上記資格付与キーをコンピュータ・システムに伝送する段階とを含む、請求項20に記載の、ソフトウェア・モジュールを配布する方法。

【請求項25】上記コンピュータ・システムが独自の識別子を含み、

資格付与キーを生成する上記段階が、上記の独自の識別子を用いて上記の資格付与キーを生成し、

上記資格付与キーが、同じ独自の識別子を含むコンピュータ・システム上でのみソフトウェアを実行する資格を与える、

請求項24に記載の、ソフトウェア・モジュールを配布する方法。

【請求項26】上記ソフトウェア・モジュールを配布する上記段階が、それぞれ単一の記憶媒体上に資格検証をトリガする複数の独立したトリガ手段を有する、複数のソフトウェア・モジュールを配布することを含み、

上記コンピュータ・システムに上記ソフトウェア・モジュールを実行する資格を付与する上記段階が、上記の単一の記憶媒体上の上記複数のソフトウェア・モジュールのうち少なくとも2つに別々の資格を与えることを含む、

請求項20に記載の、ソフトウェア・モジュールを配布する方法。

【発明の詳細な説明】

【0001】

【産業上の利用分野】本発明は、コンピュータ・ソフトウェアの使用に関し、より具体的には、コンピュータ・ユーザがライセンス・ソフトウェアをライセンスに従わない形で使用できるのを制限することに関する。

【0002】

【従来の技術】現代のコンピュータ・システムは、数えきれない時間にわたる技術上及びプログラミング上の熟練の成果をその設計に取り入れた、極めて複雑な機械である。コンピュータ・システムは多くの構成要素を含んでいるが、それらはハードウェアとソフトウェアに大別できる。ハードウェアは、システムを構成する、有形の回路、ボード、ケーブル、記憶装置、格納装置などである。しかし、タイプライタでピューリツァー賞を受賞できる脚本を書けないのと同様、ハードウェアは、それ自体で、現実の世界の問題を解決することはできない。ハードウェアは、何をすべきかを告げる命令を必要とする。「ソフトウェア」とは、そのもっとも純粋な形では、ハードウェアに有用な仕事を実行させる命令を言う。(ただし漠然と、ソフトウェアが記憶され配布されている媒体に適用されることもある)。ハードウェアと同様に、ソフトウェアも人間の創意工夫の産物である。

高品質のソフトウェアは、プログラマとも呼ばれる作成者の側の、かなりの創造性、訓練、知能を必要とする。多くの大学が、人々にソフトウェアを作成する技術を教える目的で、「コンピュータ科学」及び類似の学科の教育課程を設けている。この業界全体が、何千人ものキャリアを抱え、有用な仕事をするソフトウェアを作成するまでに成長して来た。ソフトウェアの開発に莫大な労力が費やされる結果、コンピュータ・システムの一部分であるソフトウェアの価値が、ハードウェアの価値を上回ることも珍しくはない。

【0003】現代のコンピュータ・システムの特徴の一つは、非常に高速度で容易にデータを伝送し複製できることである。一般的に、これは必要かつ有益な能力である。しかし、このことはまた、何年も労力をかけて作成したソフトウェアが、比較的安価な磁気媒体で1秒の何分の一かで複製でき、濫用される可能性があることを意味している。許可を得ていない人々が、僅かな費用と労力で、価値のあるソフトウェアを複製することができ、この慣行は、一般に、ソフトウェア著作権侵害と呼ばれている。近年、ソフトウェア著作権侵害を抑制するため、刑法及び民法上の義務を課する様々な法律が制定されてきている。しかしながら、ソフトウェアが比較的容易に複製でき、かつソフトウェアが高価なためにそうしたくなる誘惑にかられるため、ソフトウェアの著作権侵害は依然として問題となっている。

【0004】「パーソナル・コンピュータ」と呼ばれる小型コンピュータ用の大量配布される安価なソフトウェアの場合、全顧客に対して同額の固定した一括払い料金で、ソフトウェアの使用ライセンスを認めるのが普通である。これは、大型のメインフレーム・コンピュータ・システムでは、それほど行われていない。このような大型システム用のソフトウェアは、数百万行のコードを必要とすることがあり、開発及び維持に非常に大量の投資を必要とする。開発者がこの投資を回収するのに十分な単一の固定した一括払い料金を設定すると、多くの小口ユーザにとって、使用できないほど高価になりがちである。したがって、メインフレームの場合は、使用量に基づくソフトウェアのライセンス料を請求するのが普通である。「段階的価格設定」と呼ばれるこのような一つの方法は、可変料金体系に従って顧客の機械の性能に基づいた料金を請求するものである。より多くの端末が接続されたより高速の機械を使用する顧客ほど、同じソフトウェアに対し、性能の劣る機械の顧客よりも高いライセンス料を支払う。もう一つの通常の慣行は、ソフトウェアの保守グレードアップに対して別途料金を請求するものである。

【0005】高品位のソフトウェアを作成するのに必要な専門知識のレベルと、それにかかる時間及び労力の量を考えると、このようなソフトウェアの作成には金がかかる。ソフトウェアの開発者がその訓練と努力に比して

報われない場合は、ソフトウェアを作成する人がいなくなる。開発者が行ったソフトウェアへの投資を保護することは、実際上の要請であるだけでなく、道徳上の要請でもある。したがって、ソフトウェアの正当な所有者は、ソフトウェアを許可なしに使用することが難しく、ソフトウェアの開発者がその製品に対して正当に報われる、ソフトウェア配布方法の開発を求めて来た。

【0006】ソフトウェアは、正当な所有者から多数の異なる方法で配布される。無許可の使用を防止する観点から、これらの配布方法は、無制限資格付与方法、制限資格付与方法、非資格付与方法の3つのグループに大別できる。

【0007】無制限資格付与方法とは、配布されたソフトウェアが、その設計の対象であったどの機械でも、制限なしに走行するという意味である。無制限資格付与方法のソフトウェアを配布する所有者は、各ユーザに、ユーザがそれに対して対価を支払いそれを実行する資格を有するソフトウェアだけを配布しなければならない。このようなソフトウェアの受取側がこれを許可されない形で複製したり使用したりするのを妨げるものは、法的及び契約上の義務だけである。大部分のパーソナル・コンピュータで使用されているものなど、一括払い料金でライセンスが与えられる安価なソフトウェアでは、これがもっとも普通の配布方法である。

【0008】制限資格付与方法とは、ソフトウェアが、ユーザがそれを複写していくらでも多くの機械上で実行できるのを制限する、ある種の制限を内蔵していることを意味する。いくつかの異なる制限資格付与方法がある。その一つは、配布されたオリジナルから作成できるコピーの数を制限する、複写制限である。複写制限は、ソフトウェア著作権侵害に対してあるレベルの保護を実現するが、若干の欠点をもつ。無制限資格付与方法と同様に、複写制限でも、所有者が各ユーザに、そのユーザが実行する資格を有するソフトウェアだけを配布することが必要となる。これはフルブープではなく、保護機構を打ちやぶって、複写保護されたソフトウェアの文字通りのコピーをとることのできるプログラムも存在する。最後に、これは、ユーザがバックアップの目的で正当なコピーをとれる、またはソフトウェアを高速記憶装置から走行できるのを妨げる。もう一つの制限資格付与方法は、ユーザまたは機械に特有の情報をソフトウェア自体の中にコード化するものである。ソフトウェアを実行する際、機械は、ソフトウェアがその機械またはユーザに許可されていることを確かめるために検査を行う。この方法は、ユーザが正当なコピーをとれるのを妨げずに、保護を実現する。ただし、これは、配布されたソフトウェアの各コピーをそれぞれ独自にコンパイルし、配布媒体に載せ、発送しなければならないので、非常に高価な配布システムが必要となる。

【0009】非資格付与方法は、配布されたソフトウェア

が使用禁止にされていて、実行するには別途配布される資格付与方法を必要とすることを意味する。ある非資格付与方法は、特注ソフトウェアの配布を完全に避ける可能性がある。ただし、必ずしもこのような方法の全てがこうした能力をもっているわけでない。たとえば、所有者は、同じ1組の多数のソフトウェア・プログラムを単一の総称媒体上でその全ての顧客に配布し、顧客が支払いを済ませたプログラムだけを実行することを許す、個別化された許可キーを各顧客に別途配布することができる。非資格付与方法は、他の方法に付随する多くの問題を回避するが、現在の設計は、資格付与方法の偽造や著しい性能劣化を受ける恐れが大きい。大抵の場合、ソフトウェアを実行するための資格付与方法を差し止めるために用いられる機構は、性能劣化という検証上のオーバーヘッドを避けるため、資格検証手段をソフトウェア・モジュール内に（データあるいは命令として）集中することを必要とする。場合によっては、この資格付与方法のオーバーヘッドが、製品識別子のサイズ、及び配布されたソフトウェア内の保護ルーチンの実装によることがある。あるいはまた、そのオーバーヘッドがソフトウェアを走行させながら複雑な復号手段を実行する必要によることもある。こうした資格検査の集中の結果、経験あるプログラマが、「パッチング」すなわち目的コードの選択された大部分を無効にすることによって、保護機構を無効にするのは、比較的容易である。別の場合、目的コードは資格検証を行わず、モジュールを、それからビット署名を作成することによって識別する、安全呼出経路がそれを行う。これはパッチングを受け難くするが、不可避免的に呼出し機構の重大な性能劣化をひき起こす。

【0010】従来技術で教示されている保護方法は、保護レベルと使いやすさとの折合いをつけるものである。機械特有の情報をソフトウェア中にコード化することによって比較的高レベルの保護を得ることが可能であるが、配布される各ソフトウェア・コピーが独自のものとなる、非常に複雑な配布システムを維持するという犠牲を払わなければならない。より安価な配布も可能であるが、保護が一部失われるという犠牲を払わなければならない。高レベルの保護を実現し、大量配布技法を用いて容易に配布でき、かつシステムの性能、ユーザが正当にバックアップ複写用コピーをとる可能性、その他の必要な機能を不当に妨げない方法が求められている。同時に、段階的価格設定、及び異なるソフトウェアの異なるバージョンごとに別々のライセンス料をサポートする方法も求められている。

【0011】

【発明が解決しようとする課題】本発明の目的は、コンピュータ・システムにおけるソフトウェアの使用を管理する改善された方法及び装置を提供することにある。

【0012】本発明の他の目的は、コンピュータ・システムにおける無許可のソフトウェア使用に対するより高

レベルの保護を提供することにある。

【0013】本発明の他の目的は、ソフトウェアを無許可使用に対して保護する費用を削減することにある。

【0014】本発明の他の目的は、無許可使用に対して保護されたソフトウェアを実行するコンピュータ・システムの性能を増強することにある。

【0015】本発明の他の目的は、無許可使用に対して保護されたソフトウェアの配布者の配布システムを簡単にすることにある。

【0016】本発明の他の目的は、このような保護がソフトウェアの正当な使用に及ぼす影響を削減する、ソフトウェアを無許可使用から保護する方法及び装置を提供することにある。

【0017】本発明の他の目的は、ユーザがソフトウェアの正当なバックアップ用コピーをとるのを許しながら、ソフトウェアを無許可使用から保護する改善された方法及び装置を提供することにある。

【0018】本発明の他の目的は、ソフトウェアを無許可使用の使用が可能となるように改変するのを難しくすることにある。

【0019】本発明の他の目的は、段階的に価格設定したソフトウェアを配布する、改善された方法及び装置を提供することにある。

【0020】

【課題を解決するための手段】本発明によれば、ソフトウェアは、実行するための資格付与なしに配布される。別々に配布される暗号化された資格付与キーにより、ソフトウェアの実行が可能となる。この資格付与キーは、それに対してソフトウェアのライセンスが与えられている機械の通し番号と、どのソフトウェア・モジュールがその機械で走行する資格をもつかを指示する複数の資格付与ビットを含んでいる。安全解読機構が機械に内蔵されている。安全解読機構が機械の通し番号を取り出し、これを資格付与キーを解読するためのキーとして使用する。次いで、資格付与信息が、メモリ内の製品ロック・テーブルに記憶される。

【0021】配布されたソフトウェアは、複数の資格検証トリガを含んでいる。好ましい実施例では、各トリガは、ソフトウェア・モジュールの製品番号を識別する、目的コード中の単一機械語命令である。ソフトウェア・モジュールの実行中にこのような資格検証トリガに出会うと、機械は、ソフトウェア・モジュールの製品番号に対応する製品ロック・テーブルのエントリを検査する。製品が走行する資格をもつ場合、正常に実行が継続され、そうでない場合は、実行が打ち切られる。この検証はただ1個の機械語命令しか要しないので、システム全体の性能にほとんど影響を与えずに実行できる。その結果、かなりの数のこのような資格検証トリガを目的コード中に配置し、誰かがこのトリガを「パッチング」することによってコードを改変することを事実上不可能にす

ることができる。別の実施例では、資格検証トリガが、ソフトウェア・モジュールの適正な実行に必要な何らかの有用な仕事をも行う。これによって、ソフトウェアを「パッチング」するのが一層難しくなり、このような検証トリガの性能に対する影響がさらに軽減される。

【0022】ソフトウェア自身はどのような資格付与も含んでいないので、配布の制限は必要でない。好ましい実施例では、ソフトウェアの配布者は、複数のソフトウェア・モジュールを単一の総称媒体に記録し、記録された同じ1組のモジュールをその全ての顧客に配布することができる。各顧客は、自分がライセンスを与えられているソフトウェア・モジュールだけを実行できる、独自の資格付与キーを受け取る。顧客に、ライセンスが与えられていない何らかのモジュールが与えられても、適切なキーなしにはそれが実行できないので、大きな問題にはならない。顧客は、自由にソフトウェアを自分のシステムのその他の記憶装置にロードし、あるいはソフトウェアのバックアップ用コピーをいくらかでも作成することができる。

【0023】

【実施例】本発明の好ましい実施例にもとづくソフトウェア保護機構の主要要素の説明図を図1に示す。顧客のコンピュータ・システム101は、制御記憶機構103に緊密に結合された中央演算処理装置(CPU)102、ランダム・アクセス・システム・メモリ104、消去不能で電子的に読取り可能な独自の識別子105、複数の記憶装置106、107、108を含んでいる。好ましい実施例では、独自の識別子105は機械の通し番号である。好ましい実施例では、記憶装置106~108は、回転式磁気ディスク記憶装置であるが、他の記憶技術を使用することも可能である。コンピュータ・システム101はまた、オペレータからの入力を受け取る操作卓109及びソフトウェア媒体読取装置110をも含んでいる。図1には1台の操作卓、1個のソフトウェア媒体読取装置、3個の記憶装置が示されているが、システム101に取り付けられるこのような装置の実際の数は可変であることを理解されたい。また、追加の装置をシステム101に取り付けられることを理解されたい。好ましい実施例では、コンピュータ・システム101は、IBM社のAS/400コンピュータ・システムであるが、他のコンピュータ・システムも使用できる。

【0024】ソフトウェア・モジュールは、それを動作させるのに必要な資格付与キー(entitlement key)とは別に配布される。本来、ソフトウェア・モジュールは、開発用コンピュータ・システム125上で作成される。開発用コンピュータ・システム125は、コンパイラ126及びトランスレータ127を含んでいる。ソフトウェア・モジュールはソフトウェア記録媒体112上に記録され、倉庫120に格納され、そこから顧客に配布される。配布者の販売事務所121か

ら、暗号化された資格付与キー111が配布される。販売事務所121は、マーケティング用コンピュータ・システム124にアクセスすることができる。マーケティング用コンピュータ・システム124は、資格付与キーの生成/暗号化プログラム122と、顧客情報を含むデータ・ベース123とを含んでいる。具体的には、データ・ベース123は、暗号化された資格付与キーの生成に必要な機械の通し番号とプロセッサの種類の情報を含んでいる。好ましい実施例では、マーケティング用コンピュータ・システム124は、中央に位置して複数の販売事務所と通信するコンピュータである。ただし、マーケティング用コンピュータ・システム124は、販売事務所自体に位置し、局所または中央のデータ・ベースにアクセスすることもできる。図1には、配布者の制御下にある別々の2つのコンピュータ・システム124、125が示されているが、両方の機能を実行する物理的に単一のコンピュータ・システムでもよいことを理解されたい。

【0025】ソフトウェアの配布者から顧客に、暗号化された資格付与キー111が郵送、電話またはその他の適当な手段で送られる。資格付与キーを電子的にまたはフロッピー・ディスクなどの磁気媒体上で伝送することが可能であるが、キーは、オペレータが操作卓109上でタイプして、これをシステム101中に入力できるほど十分簡潔なものである。

【0026】好ましい実施例では、多数のソフトウェア・モジュールが同一媒体112上で配布される。配布者は、資格付与キーを介して各モジュールを使用する資格を独立に与えることができる。倉庫120は、媒体112上に、総称的に1組のソフトウェア・モジュールを格納する。各顧客に、どのモジュールを使用するライセンスが与えられているかにかかわらず、同じ総称的な1組のソフトウェア・モジュールが発送される。別途に配布される資格付与キーは、どのソフトウェア・モジュールをその上で実行する資格があるかをシステム101が決定するための情報を含んでいる。

【0027】好ましい実施例では、ソフトウェア媒体112は、一枚または複数の読取り専用光ディスクから構成され、媒体読取装置110は光ディスク読取装置である。ただし、電子式配布媒体やその他の配布媒体も使用できることを理解されたい。ソフトウェア媒体112を受け取ると、通常、顧客は、所望のソフトウェア・モジュールを媒体読取装置110からシステム101にロードし、そのソフトウェア・モジュールを記憶装置106~108に記憶する。顧客は、ソフトウェア・モジュールの一つまたは複数のバックアップ用コピーを任意の適当な媒体上に生成し、これらをアーカイブに記憶することができる。ソフトウェア媒体112は、システムへの複写やロードに対する制限を含んでおらず、自由に複写可能かつロード可能である。

【0028】好ましい実施例による暗号化前の資格付与キー200の内容が図2に示されている。このキーは、料金グループ・フィールド201、ソフトウェア・バージョン・フィールド202、キー形式フィールド203、機械通し番号フィールド204及び製品資格付与フラグ205を含んでいる。料金グループ・フィールド201は、16通りの機械段階値から、1つを指定し、ソフトウェアの段階的価格設定をサポートするのに使用される。ソフトウェア・バージョン・フィールド202は、資格が与えられるソフトウェアのバージョン・レベルを指定する。ソフトウェアのグレードアップを維持するための別途料金が課せられることが予想されている。資格付与キー200中で指定されるバージョンは、そのバージョン・レベルより以前(下位)の全レベルのソフトウェアに資格を与える。キー形式フィールド203はキー・フォーマット、キーの関連性の将来変更のため、あるいはサポートされる異なる製品の番号拡張のために保留された領域である。機械通し番号フィールド204は、資格付与キーが対象としている機械の通し番号を含んでいる。製品資格付与フラグ205は、それぞれが製品番号に対応する80本の別々の製品フラグを含む、80ビットのフィールドである。対応する製品番号に資格が与えられている場合、このビットは“1”にセットされ、そうでない場合は、“0”にセットされる。

【0029】好ましい実施例では、ソフトウェア・モジュールは、コンパイルされた目的コードとして配布される。典型的なソフトウェア・モジュール300を図3に示す。このソフトウェア・モジュールは、コンピュータ・システム101上で実行できる複数の目的コード命令を含んでいる。本発明によれば、いくつかの資格検証トリガ命令(以下、「資格検証トリガ」と表記)301が目的コードに組み込まれている。ソフトウェア・モジュール内に含まれる全ての資格検証トリガ301は、同一である。資格検証トリガ301は、命令コード・フィールド302、バージョン・フィールド303、製品番号フィールド304を含んでいる。フィールド305は使用されていない。命令コード・フィールド302は、実行すべき操作を識別する、目的コード命令の動詞部分である。バージョン・フィールド303は、ソフトウェア・モジュールのバージョン・レベルを識別する。製品番号フィールド304は、そのソフトウェア・モジュールに関連する製品番号を識別する。バージョン及び製品番号の情報はモジュールごとに変わるが、全ての資格検証トリガには単一の命令コードが使用される。別の実施例では、資格検証トリガは、他の何らかの有用な仕事を実行する直接命令(とりわけ、その処理用のオペランドを命令中で指定しておく必要のない命令)でもある。このような別の実施例では、トリガ命令が実行されると、システム101は資格検証と同時に他の何らかの操作を実行する。

【0030】コンピュータ・システム101は、暗号化された資格付与キー111を受け取ってデコードする手段、ならびに資格検証トリガにตอบสนองしてシステムがソフトウェア・モジュールを実行する資格をもつことを検証する手段を含んでいる。好ましい実施例では、図4に示されているように、これらの機能は、異なるレベルのシステム・ハードウェア及びシステム・ソフトウェアの間で分割されている。この実施例では、システム101は、ハードウェア・レベル401、水平マイクロコード・レベル402、実行可能コード・レベル403、仮想計算機レベル404という、4レベルのハードウェア/ソフトウェア機能を含んでいる。機械インターフェース405が、仮想計算機レベルを下位の全てのレベルから分離している。機械インターフェース405は、顧客に対して定義されている最も下位レベルにあるインターフェースである。すなわち、仮想計算機レベルの命令ビットは顧客に対して定義されるが、それより下のレベルの操作は定義されない。したがって、顧客は、機械インターフェース・レベルより下のレベルの命令を直接変更する能力がない。永久的な、変更不能な機械通し番号410が、ハードウェア・レベル401に格納されている。

【0031】水平マイクロコード402は、実行可能な命令セットを解釈するマイクロコード・エントリを含んでいる。これは、制御記憶機構103、すなわち好ましい実施例では、顧客による変更が不可能な読み専用メモリ（ROM）に、物理的に記憶されている。水平マイクロコード中のエントリは、機械キー取得機能420、ロック設定機能421、ロック検査機能422をサポートしている。機械キー取得機能420は、好ましい実施例ではシステム通し番号に基づく、独自の識別子を、ある永久的なハードウェア位置から取り出す。ロック設定機能421は、製品ロック・テーブル460にアクセスし、テーブル中のエントリを変更する。ロック設定機能は、製品ロック・テーブル460を変更できる唯一のマイクロコード機能である。ロック検査機能422は、製品ロック・テーブル460にアクセスし、エントリの一つを読み取って資格が付与されているかどうかを検証する。

【0032】実行可能コード・レベル403は、下位レベルの監視サポート、及び機械インターフェースで定義された命令を実施するサポートを含んでいる。これはメモリに物理的に記憶されるが、その内部仕様が顧客に対して定義されていないので、実際上は顧客によって変更できない。下位レベルのサポートには、ロック解除ルーチン430、初期プログラムロード（IPL）ロック再検証ルーチン431、例外処理ルーチン432が含まれる。アンロック・ルーチン430は、固有の機械キーを用いて資格付与キー111をデコードし、暗号化された資格付与キー111を製品キー・テーブル450に記憶する。初期プログラムロードロック再検証ルーチン43

1は、システムが再び初期設定されると、コード化された製品キー・テーブル450の内容を取り出して、製品ロック・テーブル460を再構築する。例外処理ルーチン432は、水平マイクロコード・レベル402の機能によって生成される例外条件にตอบสนองする。実行可能コード・レベル403は、目的コード形式のソフトウェア・モジュール300を含んでいる。

【0033】仮想計算機レベル404は、機械語命令によって表されているものとしてソフトウェアを参照する。このレベルの計算機は、機械語命令が直接実行可能ではないという意味で、仮想計算機として動作する。仮想計算機レベル404は顧客に利用できる最も下位レベルのインターフェースなので、システム上で実行可能な目的コード形式のモジュールを作成するのに、非従来型のコンパイル経路が必要である。このコンパイル処理については、後に説明する。厳密には、この非従来型のコンパイル経路を介して作成されるソフトウェアは、実行しようとする実行可能な目的コードの形式をとらなければならないが、機械語命令が直接実行可能であるかのように、仮想計算機レベルの機械語命令で表して議論するのが普通であり、よりわかりやすい。これが確立されると、仮想計算機レベル404がコンパイルしたユーザのソフトウェアを含んでいるとすることができる。これはまた、システム101に対するより上位レベルのオペレーティング・システムのサポートも含んでいる。仮想計算機レベル404でこのオペレーション・システムのサポートは、資格付与キーの入力をサポートするのに必要な2個のユーザ・インターフェース・ルーチンを含んでいる。一般入力ルーチン441は、通常の操作中に入力を処理するのに使用する。さらに、オペレーティング・システムの初期導入中に資格付与キーを入力するには特別なインストール入力ルーチン440が必要である。これが必要なのは、機械インターフェース・レベル405より上位のオペレーティング・システムの部分が、本発明では他のプログラム製品として扱われるためである。すなわち、このような部分は製品番号をもち、その目的コードは資格検証トリガの影響を受ける。インストール入力ルーチン440は、オペレーティング・システム、資格検証トリガをもたない唯一の部分であり、従って、システムを最初に導入する際に資格付与キーが入力できるようになる。

【0034】ソフトウェア・モジュール300は、システム101上で実行されるコンパイルされた目的コード形式のプログラム製品の一部である。これは、仮想計算機レベルの他の対象にアクセス可能であるという意味で、仮想計算機レベル404のエンティティとして存在する。ただし、実際に実行可能なコードは、破線の枠で示されているように、実行可能コード・レベル403で動作する。実行可能コードは、水平マイクロコードのロック検査機能422によって実行される、資格検証トリ

が301(一つだけが図に示されている)を含んでいる。

【0035】コード化された製品キー・テーブル450が図5に示されている。製品キー・テーブル450はランダム・アクセス・メモリ104に入っており、不揮発性記憶装置に複製されているため、システムの電力を遮断したり、その他の理由で再初期設定しなければならない場合に、回復することが可能である。テーブル450は、それぞれが各製品番号に対応し得る80個のエントリ501を含んでいる。各エントリ501は、そのエントリの製品番号に適用される暗号化された資格付与キー502と、そのキーが最初にいつ使用されたかを示す日時スタンプ503と、そのキーのバージョン番号504と、そのキーの料金グループ505と、そのキーが製品をアンロックするかどうかを示す資格付与ビット506との完全なコピーを含んでいる。日時スタンプ503は暗号化することができる。バージョン番号504、料金グループ505及び資格付与ビット506は、暗号化された資格付与キー502に含まれる情報を繰り返す。それらは、照合をサポートするためにこのテーブルに含まれている。ただし、最初に暗号化された資格付与キー502中の情報を検証してからでないと、製品ロック・テーブル460のエントリを改変して、プログラム実行の資格を与えることはできない。製品キー・テーブル450中の各資格付与キー502は暗号化された形なので、ユーザーのこのテーブルへのアクセスを防止するために特別な措置は必要でない。

【0036】図6に、製品ロック・テーブル460が示されている。このテーブルは、水平マイクロコードの完全な制御下にある、ランダム・アクセス・メモリ104の特別な下位アドレス範囲に入っている。製品ロック・テーブル460は、それぞれが各製品番号に対応し得る80個のエントリ601を含んでいる。各エントリは、資格付与の最大バージョンのレベルを示すバージョン番号を含んでいる。0のバージョン番号は、製品のどのバージョンにも資格付与がないことを示している。製品ロック・テーブル460の改変に対する保護の度合をさらに強化するために、バージョン番号をスクランブルすることができる。

【0037】次に、本発明の好ましい実施例によるコンピュータ・システム101上でのソフトウェア・モジュールの動作について述べる。この動作には4つの部分がある。第1の動作は、複数の資格検証トリガを実行可能な目的コード形式のソフトウェア・モジュール中に配置する。第2の動作は、ソフトウェア・モジュールへのアクセスを許可する暗号化された資格付与キー111を生成する。第3の動作は、コンピュータ・システム101が資格付与キー111を受け取り、デコードして、記憶し、製品ロック・テーブル460を設定する。第4の動作は、ソフトウェア・モジュールをシステム101上で

実行して、資格検証トリガに出会ったときには、システム101に資格を検証させる。始めの2つの動作は、ソフトウェア配布者の管理下で実施される。後の2つの動作は、顧客のシステム101上で実施される。

【0038】ソフトウェア配布者は、ソフトウェア・モジュールをコンパイルするとき、資格検証トリガを目的コード中に配置しなければならない。開発用コンピュータ・システム125で起こる典型的なこの過程を図7に示す。原始コードは、資格検証トリガを含めずに、通常通りプログラマによって生成される。ステップ701で、原始コードがコンパイラ126に入力され、ステップ702で、プログラム・テンプレートを作成する。プログラム・テンプレートは、仮想計算機レベル404(すなわち機械インターフェース405より上のレベル)の機械語命令を含んでいる。ステップ704で、プログラム・テンプレートは、その製品番号及びバージョン番号を識別すると共に、トランスレータ127への入力として働く。トランスレータ127は、自動的に、かなりの数の資格検証トリガを生成し、これを目的コード中のランダムな位置に挿入し、資格検証トリガの挿入後に参照を解決する。ステップ705で出力された結果得られる実行可能な目的コード形式のソフトウェア・モジュールには、資格検証トリガが含まれている。この実行可能な形式のソフトウェア・モジュールは、実行可能コード・レベル403の目的コード命令を含んでいる。

【0039】暗号化された資格付与キーを生成する過程を図8に示す。ステップ801で、販売担当員によって生成されたあるバージョン・レベルの1個または複数のプログラム製品に対するライセンスの注文が、マーケティング用コンピュータ・システム124に入力される。理論上、顧客が多数のバージョン・レベルの製品を注文することは可能であるが、一般的にはそうする理由はほとんどない。ただし、各資格付与キーは指定されたバージョン・レベルでしか動作しないので、顧客が異なるバージョン・レベルを注文すると、別々の資格付与キーを生成しなければならない。顧客の注文を受け取ると、ステップ802で、システム124で実行中のキー生成/暗号化プログラム122が、顧客に関する情報、具体的には機械の通し番号及びプロセッサ形式の情報を含むデータベース123にアクセスする。この情報を使って、暗号化されていない資格付与キー200の料金グループ・フィールド201及び機械通し番号フィールド204が生成される。ステップ803で、顧客の注文及び可能な製品番号提供のデータベースへの参照によって残りのフィールドが生成されて、完全な非暗号形式の資格付与キーが構築される。次いで、ステップ804で、キー生成/暗号化プログラム122が、当技術分野で既知のいくつかの暗号化技法のうちのどれかを使って、資格付与キーを暗号化する。次いで、ステップ805で、その結果得られた暗号化された資格付与キー111が、顧客に

伝送される。図1ではキー111は複数の2進ビットとして示されているが、資格付与キーを鍵盤から入力する仕事を簡単にするため、16進数字や英数字による等価物など2進ビットをグループ化した何らかの形式で顧客に提示してもよい。

【0040】コンピュータ・システム101上で資格付与キー111を受け取り、製品ロック・テーブル460を維持する過程を図9a及び図9bに示す。ステップ901で、顧客は、操作卓109を介して、資格付与キー111をコンピュータ・システム101に入力する。これが初期導入である場合には、インストール入力ルーチン440がオペレータと対話して入力を受け取る。そうでない場合は、一般入力ルーチン441が入力を受け取る。資格付与キーは、デコード過程を処理するロック解除ルーチン430に渡される。ステップ902で、ロック解除ルーチン430が、機械キー取得機能420に、機械通し番号を検索させ、機械キーを生成させる。次いで、ステップ903で、ロック解除ルーチン430が、機械キーを用いて資格付与キー111をデコードする。次いで、ステップ904で、以下に述べるように、ロック解除ルーチン430がコード化された製品キー・テーブル450を再構築する。デコードされた資格付与キーは、図2に示した形をとる。これは、各製品番号ごとに、その製品が新しい資格付与キーの下でロックを解除されているかどうかを示す、80ビットの製品資格付与フラグ・アレイ205を含んでいる。新しい資格付与キーは、それがロックを解除する全ての製品に対する置換されたキーと見なされる。ロック解除ルーチン430は、デコードされた資格付与キー中の各製品資格付与フラグ205を走査する(ステップ904)。ステップ905で、製品資格付与フラグが“1”(資格付与ありを指す)にセットされている場合、ステップ908で、製品キー・テーブル450中の対応するエントリが、新しい資格付与キー、バージョン番号及び料金グループ値で置換される。資格付与ビット・フィールド506が“1”にセットされ、日/時スタンプ・ビット・フィールド503が、資格付与キーがまだ使用されていないことを示すゼロの値にセットされる。新しいキーの製品資格付与フラグが“0”の場合には、バージョン番号と料金グループ番号が製品キー・テーブル450に記憶されているものと同じでない限り、新しい資格付与キーは効果がない。バージョン番号及び料金グループ番号が同じ場合(ステップ906)、資格付与キーは製品をロックする効果がある。したがって、ロック解除ルーチン430は、ステップ907で、製品ロック・テーブル460中のバージョン番号を“0”にセットするために、ロック設定機能421を呼び出し、ステップ908で、製品キー・テーブル450中の対応するエントリを新しい資格付与キーの値で置換する。製品キー・テーブル450が再構築されると、ステップ909で、その内容が記憶

装置にセーブされる。

【0041】前に資格を付与された製品が資格を失わない限り、製品キー・テーブル450の再構築は製品ロック・テーブル460に直接影響を与えない。製品は、要求に応じて、ロックを解除される。前に資格を付与されていないソフトウェア製品を最初に行うとき、資格検証トリガに出会うと、システムによって例外が生成される。次いで、ステップ920で、例外処理ルーチン432が、製品のロック解除を試みるため、ロック解除ルーチン430を呼び出す。次に、ステップ921で、ロック解除ルーチン430が、コード化された製品キー・テーブル450中の適当なエントリから暗号化された資格付与キーを取り出し、ステップ922で機械キーを得て、ステップ923で資格付与キーを解読する。資格が付与されている(ステップ924)場合は、ステップ926で、ソフトウェア製品の製品番号に対応する、製品ロック・テーブル460のエントリ601中でバージョン番号を設定するために、ロック設定機能421が呼び出される。同時に、ステップ927で、ロック解除ルーチン430が、第1回使用の日時を日/時スタンプ・フィールド503に記録する。次いで、ステップ928で、資格検証トリガが再試行され、プログラムの実行が続行される。ステップ924で資格付与が示されない場合は、ステップ925で、プログラムの実行が打ち切られる。

【0042】製品ロック・テーブル460は、RAMに記憶されており、システムの再初期設定後は残らない。再初期設定(「IPL」)中に、製品ロック・テーブル430を再構築するため、IPLロック再検証ルーチン431が呼び出される。このルーチンは、上記のように、資格を検証し、製品ロック・テーブル460中の対応するエントリを再構築するために機械キーを得て、コード化された製品キー・テーブル450中の各資格付与キー・エントリを系統的にデコードする。

【0043】好ましい実施例による、ソフトウェア・モジュールを実行する過程を図10に示す。システム101によるソフトウェア・モジュールの実行は、モジュールの目的コード命令が終了するまで(ステップ1003)、これを取り出して(ステップ1001)実行する(ステップ1002)ことによってなされる。命令が資格検証トリガ301である(ステップ1004)場合は、ロック検査機能422が呼び出される。ステップ1005で、ロック検査機能422が、資格検証トリガに含まれる製品番号に対応する、製品ロック・テーブル・エントリ601にアクセスする。製品ロック・テーブル460中のバージョン番号が資格検証トリガ301に含まれるバージョン番号303に等しいかまたはそれより大きい場合には、ソフトウェアは実行する資格を付与される(ステップ1006)。この場合、ロック検査機能422はそれ以上の処置を行わず、システムはソフトウ

ウェア・モジュール内の次の目的コード命令の実行に進む。ソフトウェアが資格を付与されていない場合は、ロック検査機能が例外条件を生成して、制御を例外処理ルーチン432に渡し、例外処理ルーチン432がプログラムの実行を終了させる（ステップ1007）。システムは、ソフトウェアが資格を付与されていることを示す資格検査の結果をセーブしない。したがって、ソフトウェア・モジュール中で資格検証トリガに再び出会うと、上記のように、システムは再度資格を検証する。

【0044】別の実施例では、資格検証トリガを、従来型のコンパイル経路を介して目的コード中に注入できるように、追加の定義をすることが可能である。これは、顧客及びコンパイラ作成者が利用できる機械インターフェースが、目的コード形式のモジュールがシステムによって実行されるのと同じ実行可能な命令のビットとなっているものとなるはずである。目的コードのフォーマットが顧客に知られている従来型のコンパイル経路をサポートするため、資格検証トリガを無効にするかまたは変更するような目的コードの「パッチング」に対する障壁を追加することが適切になる場合がある。このような追加の障壁の一つは、同時に他の何らかの機能を果たすように、資格検証トリガを定義することである。この場合、資格検証トリガによって実施される代替機能が、他の単純命令で実施できないことが肝要である。この代替機能は、コンパイルされたどんなソフトウェア・モジュールもかなり確実にその機能を果たすいくつかの命令を含むように、選定しなければならない。これらの判定規準に合致している場合、コンパイラは、自動的に、その通常のコンパイル手順の一部として、その代替機能を（それと同時に資格検証トリガも）果たす目的コードを生成することができる。この定義は、資格検証トリガを無効にするような目的コードの「パッチング」に対する重要な障壁をもたらすはずである。他の実施例は、資格検証トリガを、目的コード中で、それが識別する製品番号に対して単純な関係をもつアドレス可能位置に、配置しなければならないと定義することである。コンパイラが、通常のコンパイル過程の一部としてこれらの命令を適切なコード位置に注入するのは比較的簡単ではなく、命令実施態様での追加的検証を行うのは簡単ではなく、この定義は、資格検証トリガ中で供給される製品番号の識別を変更する、目的コードのパッチングに対する重要な障壁となるはずである。

【0045】好ましい実施例は、80個の独立な製品番号に対応している。本発明によってサポートされる製品番号の実際の数は可変であることを理解されたい。好ましい実施例では、配布者から配布される、別々にコンパイル可能なソフトウェア・モジュールの数が80を大幅に越えることも予想されている。製品番号の数は、配布者のマーケティング組織によって提供される別々に価格設定されたソフトウェア・パッケージの数に対応する。

各ソフトウェア・モジュールは一つの製品番号しかもないが、この製品番号を共有するソフトウェア・モジュールは多数存在し得る。たとえば、配布者は、画面編集、スペル・チェック、文書フォーマット化などを扱う別々のソフトウェア・モジュールを含む、ワード・プロセッシング・パッケージを提供する。こうしたソフトウェア・モジュールが常にワード・プロセッシング・パッケージの一部としてライセンスを与えられる場合には、それらのモジュールは共通の製品番号をもつことになる。別の実施例では、各ソフトウェア・モジュールごとに、別々の番号をもつことも可能である。

【0046】好ましい実施例では、ソフトウェアは一括払い料金でライセンスが与えられるが、グレードアップを維持するための追加料金が請求されることが可能である。別の実施例では、ある期間の間ソフトウェアのライセンスを許諾することができる。このような別の実施例においては、資格付与キーが、ソフトウェアのライセンスが与えられる期間の長さを示す追加のフィールドを含むことになる。コード化された製品キー・テーブル450中の製品番号に関連する日/時スタンプをライセンス許諾の期間の長さと比較して、ライセンスが満了したかどうかを判定するために、IPLロック再検証ルーチン431が、定期的に呼び出される。このとき、ライセンスが満了した場合には、製品ロック・テーブル460中の対応するエントリにロック解除ビット（図示せず）を設けておきこれを“0”にセットすることによって、新たな資格付与が得られるまで、ソフトウェア・モジュールのそれ以上の実行を防止するようにすることもできる。

【発明の効果】本発明によって、コンピュータ・システムにおけるソフトウェアの使用を管理する、改善された方法及び装置が提供される。

【図面の簡単な説明】

【図1】本発明の好ましい実施例によるソフトウェア保護機構の主要要素を示す図である。

【図2】本発明の好ましい実施例による資格付与キーの暗号化されていない内容を示す図である。

【図3】好ましい実施例による典型的な実行可能ソフトウェア・モジュールの内容を示す図である。

【図4】好ましい実施例によるソフトウェア保護機構をサポートするために顧客のコンピュータ・システム上で必要なハードウェア及びソフトウェアの構造を示す図である。

【図5】好ましい実施例によるコード化された製品キー・テーブルのフォーマットを示す図である。

【図6】好ましい実施例による製品ロック・テーブルのフォーマットを示す図である。

【図7】好ましい実施例による、資格検証トリガをソフトウェア・モジュールに配置するのに必要なステップのブロック図である。

【図8】好ましい実施例により、暗号化された資格付与キーを生成するのに必要なステップのブロック図である。

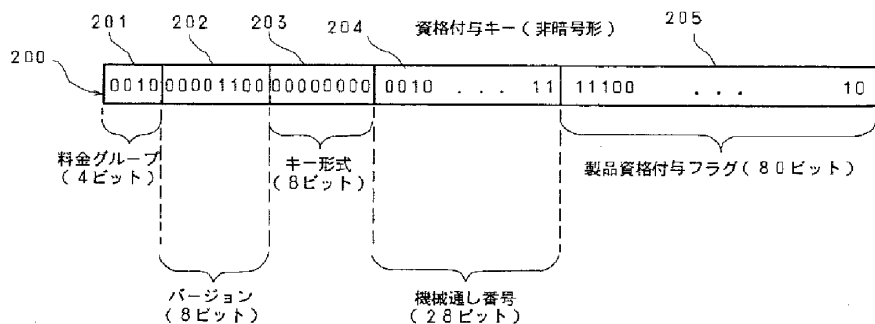
【図9】好ましい実施例による、顧客のコンピュータ・システム上で資格付与キーをデコードし、資格付与状況の記録を維持するのに必要なステップのブロック図である。

【図10】好ましい実施例による、ソフトウェア・モジュールの実行中に資格を検証するのに必要なステップのブロック図である。

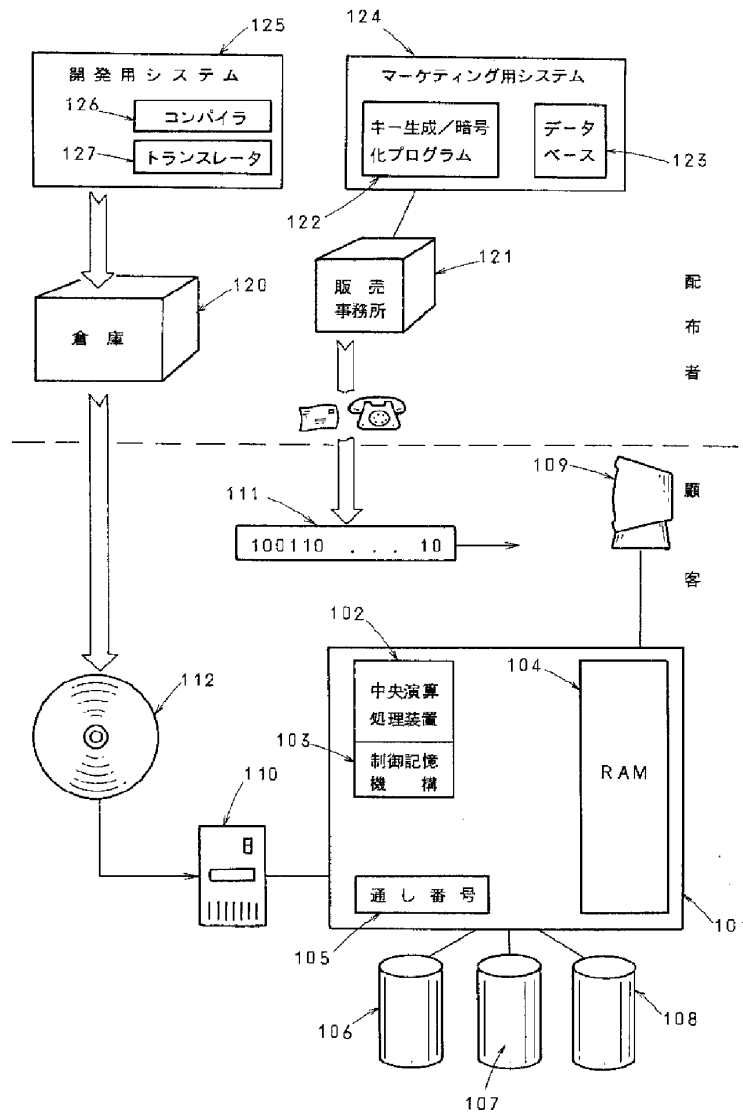
【符号の説明】

- 101 顧客側コンピュータ・システム
- 102 中央演算処理装置 (CPU)
- 103 制御記憶機構
- 104 ランダム・アクセス・メモリ (RAM)
- 106 記憶装置
- 107 記憶装置
- 108 記憶装置
- 109 操作卓
- 110 媒体読取装置
- 112 ソフトウェア記録媒体
- 120 倉庫
- 121 販売事務所
- 123 データ・ベース
- 124 マーケティング用コンピュータ・システム
- 125 開発用コンピュータ・システム
- 126 コンパイラ
- 127 トランスレータ

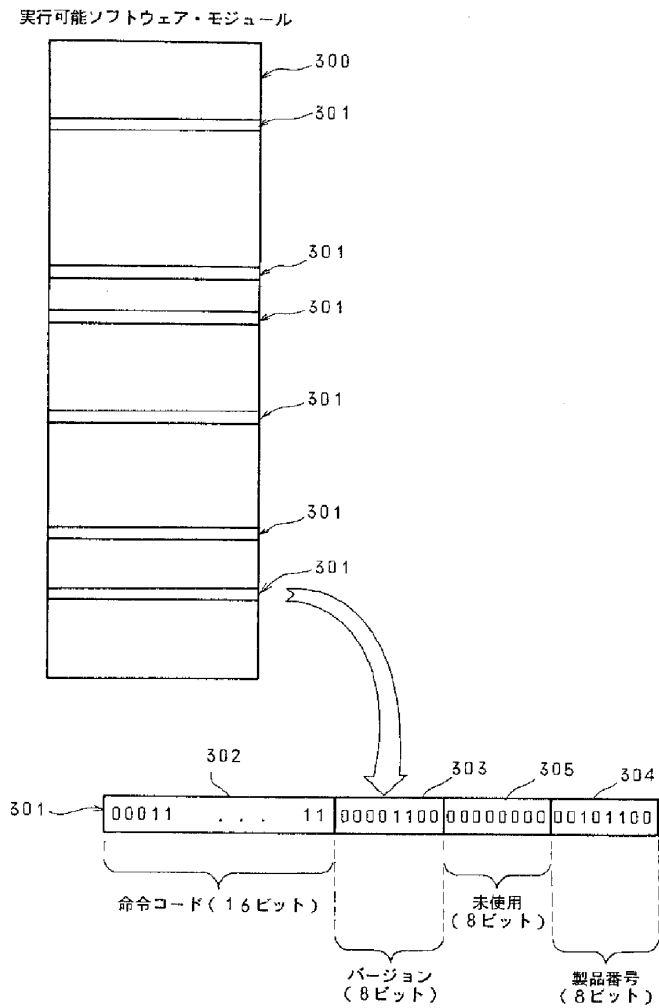
【図2】



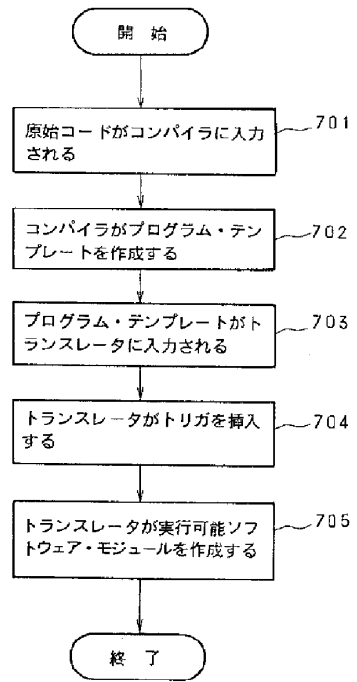
【図1】



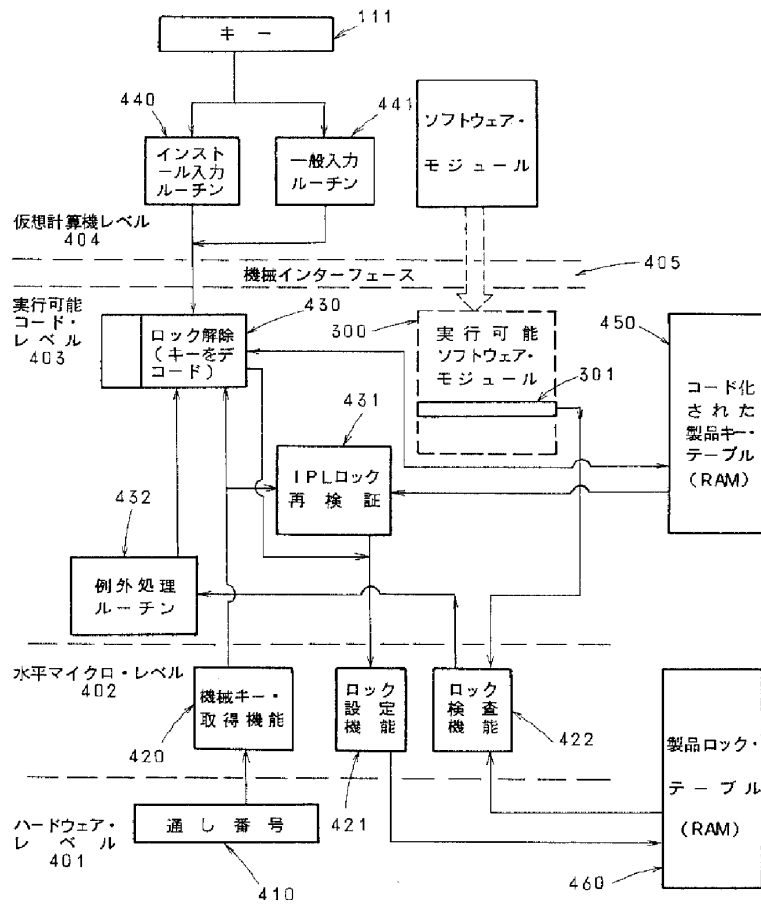
【図3】



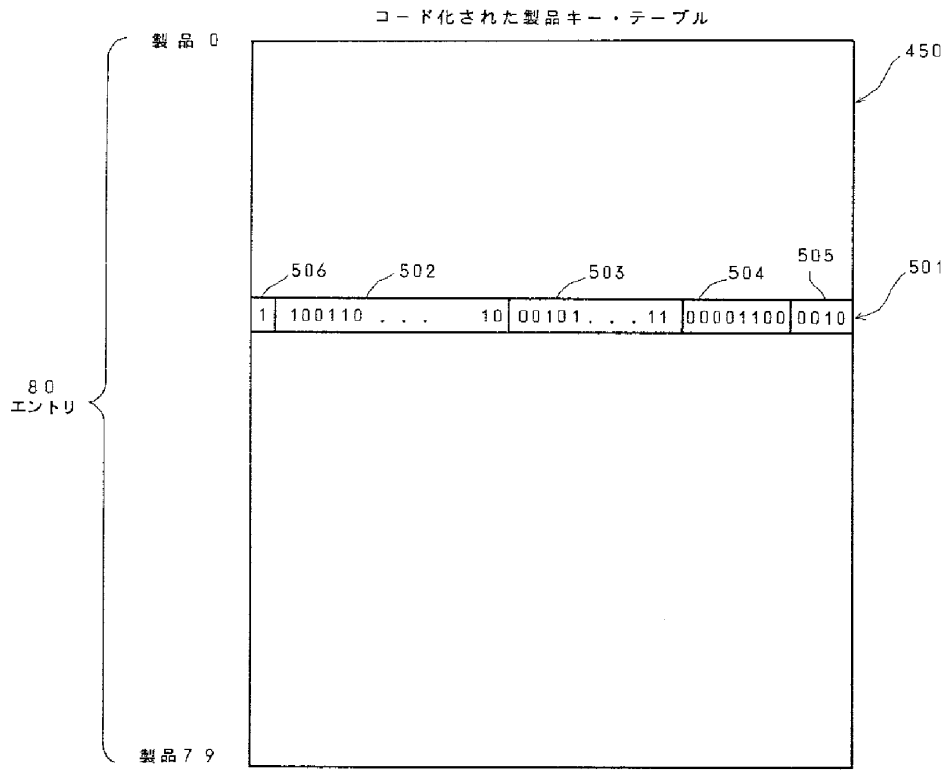
【図7】



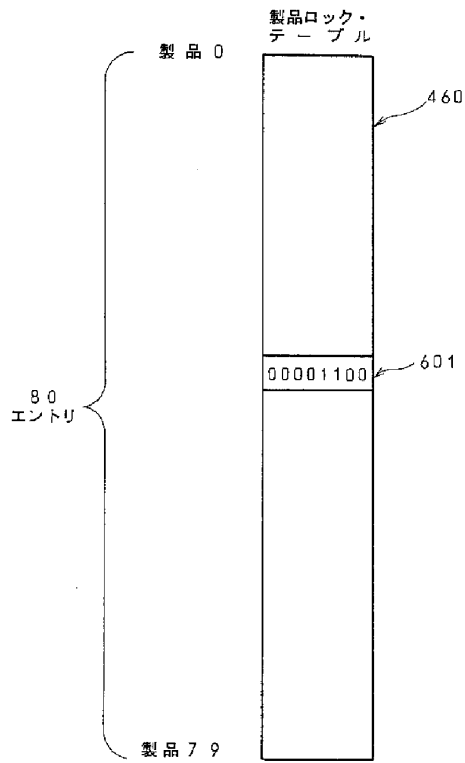
【図4】



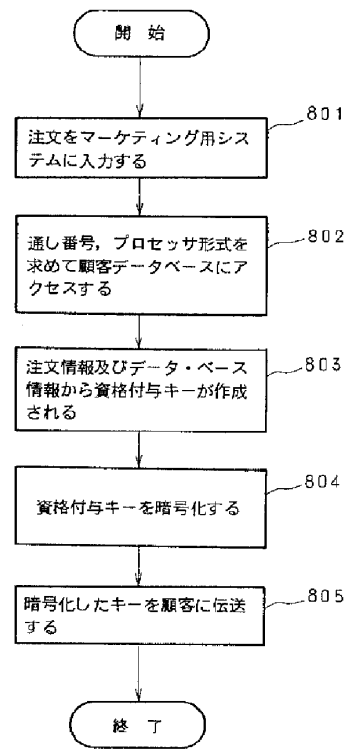
【図5】



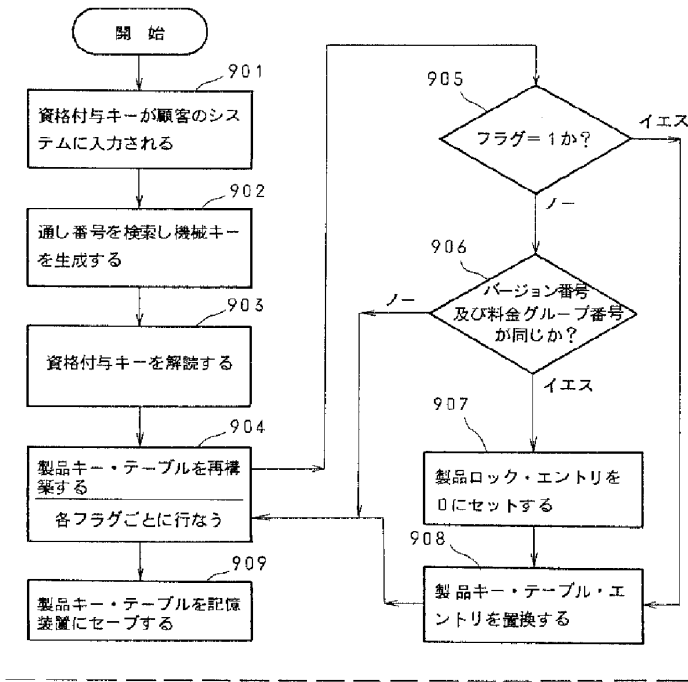
【図6】



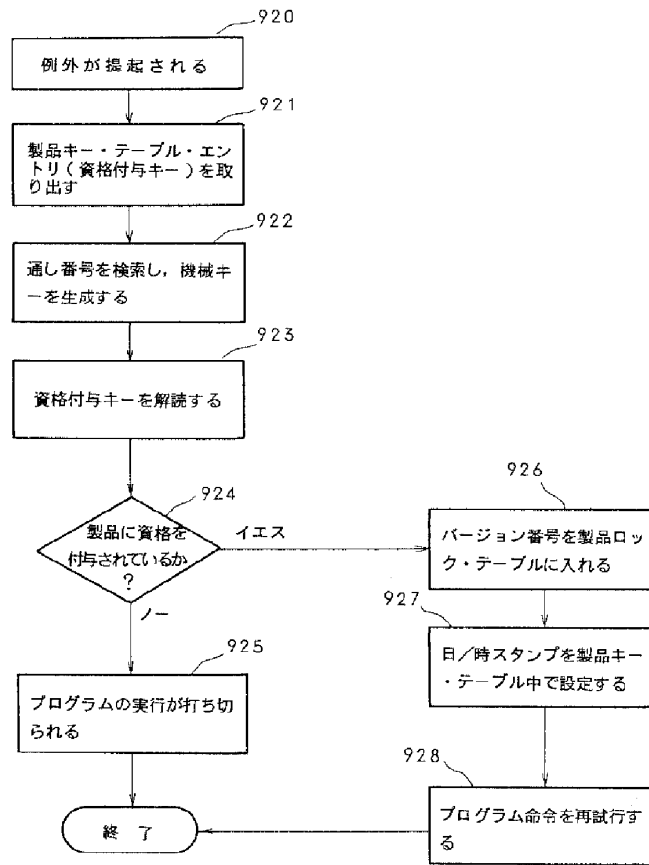
【図8】



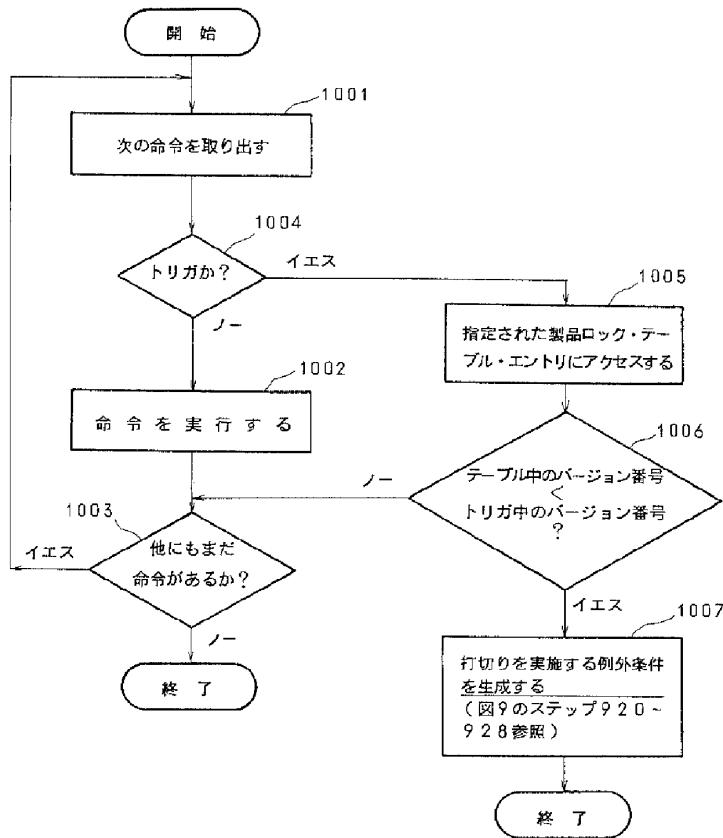
【図9a】



【図9b】



【図10】



フロントページの続き

(72)発明者 マイケル・ジョーゼフ・コリガン
アメリカ合衆国55906、ミネソタ州ロチェ
スター、20番アベニュー、ノース・イース
ト 1510番地

(72)発明者 フランシス・ジョーゼフ・リアドン・ジュ
ニア
アメリカ合衆国55901、ミネソタ州ロチェ
スター、シャトー・ロード、ノース・ウェ
スト 5685番地

(72)発明者 ジェームズ・ウィリアム・モラン
アメリカ合衆国55934、ミネソタ州エヨタ
チェスター・ロード、サウス・イースト
3221番地

Exhibit 5

(11)Publication number	05-334072
(43)Date of publication of application	17.12.1993
(51)Int.Cl.	G06F 9/06
(21)Application number	03-308350
(22)Date of filing	29.10.1991
(71)Applicant	INTERNATL BUSINESS MACH CORP <IBM>
(72)Inventor	ROBERT CARL BEECHER MICHAEL JOSEPH CORRIGAN FRANCIS JOSEPH RIADON JR JAMES WILLIAM MORAN

(30)Priority

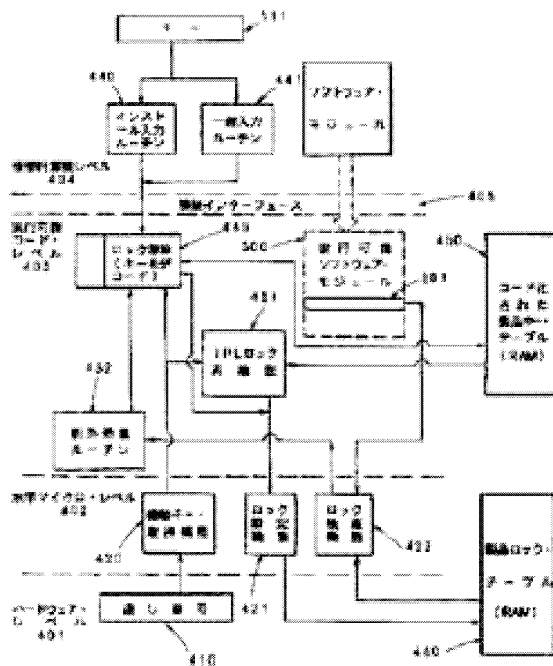
Priority number : 90 629295 Priority date : 14.12.1990 Priority country : US

(54)DEVICE AND METHOD FOR MANAGING USE OF SOFTWARE

(57)Abstract

PURPOSE: To provide an improved method and device for managing the use of software in a computer system.

CONSTITUTION: Software is distributed with no execution qualification and can only be executed with a separately distributed enciphered qualified key 111. The key 111 contains the serial number 410 of a computer to which the license of the software is given and a plurality of qualifying bits indicating the software module qualified to be run on a machine. The distributed software contains a plurality of qualification verifying triggers. Each trigger is an object code type of single machine word instruction which discriminates the product number of a software module. Therefore, an improved method and device for managing the use of software in a computer system can be provided.



[Claim(s)]

[Claim 1] Equipment characterized by comprising the following for managing use of a software module performed with computer systems.

A means to give qualification for performing a software module to computer systems.

An independent trigger means of plurality which carries out the trigger of the verification of qualification arranged in the above-mentioned software module.

A qualification verifying means which verifies that respond to independent each of a trigger means of the above-mentioned plurality, and the above-mentioned computer systems have the qualification for performing the above-mentioned software module.

A means to close execution of the above-mentioned software module when it responds to the above-mentioned qualification verifying means, the above-mentioned computer systems did not have the qualification for performing the above-mentioned software module and the above-mentioned qualification verifying means judges.

[Claim 2] Equipment for managing use of a software module of a description to Claim 1 whose independent trigger means of the above-mentioned plurality is the single target code command which carries out the trigger of the above-mentioned qualification verifying means.

[Claim 3] An addition step which needs for suitable execution of the above-mentioned software module the above-mentioned target code command which carries out the trigger of the above-

mentioned qualification verifying means is also performed, As a result, when the above-mentioned software module is corrected by removing the above-mentioned target code command, Equipment for managing use of a software module of a description to Claim 2 with which the above-mentioned addition step is not performed, but the above-mentioned software module is no longer performed appropriately.

[Claim 4]Equipment for managing use of a software module of a description to Claim 2 with which the above-mentioned target code command which carries out the trigger of the above-mentioned qualification verifying means includes product number of the above-mentioned software module.

[Claim 5]Equipment characterized by comprising the following for managing use of a software module of a description to Claim 4.

A product locking table in the above-mentioned computer systems for which the above-mentioned qualification verifying means is provided with a plurality of entries with which each is related to product number including qualification grant information.

A means to access the above-mentioned qualification grant information included in an entry in the above-mentioned product locking table relevant to the above-mentioned product number which responds to the above-mentioned target code command, and is included in the above-mentioned target code command.

[Claim 6]Equipment characterized by comprising the following for managing use of a software module of a description to Claim 1.

A means by which an above-mentioned means to give qualification for performing the above-mentioned software module to the above-mentioned computer systems generates a qualification grant key which comprises data.

A means to input the above-mentioned qualification grant key into the above-mentioned computer systems.

[Claim 7]An above-mentioned means by which the above-mentioned computer systems generate a qualification grant key including an original identifier, Equipment for managing use of a software module of a description to Claim 6 which generates the above-mentioned qualification grant key using the above-mentioned original identifier, and gives qualification for performing software only on computer systems with which the above-mentioned qualification grant key contains the same original identifier.

[Claim 8]Equipment characterized by comprising the following for managing use of a software module of a description to Claim 7.

A means prevent from decoding only on computer systems in which an enciphered qualification grant key which enciphers the above-mentioned qualification grant key using the above-mentioned original identifier, and is obtained as a result has the same original identifier.

A means in computer systems to access an original identifier of the above-mentioned computer systems.

A means in the above-mentioned computer systems to respond to an above-mentioned means to access the above-mentioned original identifier, and to decode the enciphered above-mentioned qualification grant key.

[Claim 9]A method characterized by comprising the following for managing use of a software module performed on computer systems.

A stage of arranging an independent trigger means of plurality which carries out the trigger of the verification of qualification in a software module.

A stage of performing the above-mentioned software module.

A stage of carrying out the trigger of the qualification verification operation which verifies computer systems having the qualification for performing the software module when it meets with one of the trigger means plurality became [above-mentioned] independent of during execution of the above-mentioned software module in the above-mentioned computer systems.

A stage of closing execution of the above-mentioned software module when judged with there being no qualification for performing the above-mentioned software module in the above-mentioned computer systems by the above-mentioned qualification verification.

[Claim 10]The above-mentioned stage of arranging a plurality of independent trigger means in the above-mentioned software module, A method for managing use of a software module of a description to Claim 9 including arranging a single target code command which carries out the trigger of the above-mentioned qualification verification operation to a plurality of separate positions in the above-mentioned software module.

[Claim 11]An addition step which needs for suitable execution of the above-mentioned software module the above-mentioned target code command which carries out the trigger of the above-mentioned qualification verification operation is also performed, As a result, when the above-mentioned software module is corrected by removing a target code command, A method for managing use of a software module of a description to Claim 10 that the above-mentioned addition step is not performed but the above-mentioned software module is no longer performed appropriately.

[Claim 12]A way for managing use of a software module of a description to Claim 10 the above-mentioned target code command which carries out the trigger of the above-mentioned qualification verification operation includes product number of the above-mentioned software module.

[Claim 13]Each a product locking table provided with a plurality of entries in relation to product number, including qualification grant information, The above-mentioned stage of carrying out the trigger of the qualification verification operation including a stage maintained within the above-mentioned computer systems, A method for managing use of a software module of a description to Claim 12 including accessing the above-mentioned qualification grant information included in an entry in a product locking table relevant to product number included during the above-mentioned target code command.

[Claim 14]A method characterized by comprising the following for managing use of a software module of a description to Claim 9.

A stage of generating a qualification grant key which provides information which enables it to judge whether it comprising a plurality of data bits, and the above-mentioned computer systems having the qualification for performing the above-mentioned software module.

A stage of inputting the above-mentioned qualification grant key into the above-mentioned computer systems.

[Claim 15]The above-mentioned stage where the above-mentioned computer systems generate a qualification grant key including an original identifier, A method for managing use of a software module of a description to Claim 14 of generating the above-mentioned qualification grant key using the above-mentioned original identifier, and giving qualification for performing software

only on computer systems with which the above-mentioned qualification grant key contains the same original identifier.

[Claim 16]Have a means to receive qualification for performing a software module, and it responds to a trigger means in the above-mentioned software module, In a program product for managing qualification in which the above-mentioned software module is performed on computer systems which have a qualification verifying means which verifies that the above-mentioned computer systems have the qualification for performing the above-mentioned software module,

A program product comprising:

At least one software module recorded on a recording medium.

An independent trigger means of plurality in the above-mentioned software module which carries out the trigger of the verification of qualification on computer systems.

[Claim 17]A program product for managing qualification grant of a description to Claim 16 whose independent each of a trigger means of the above-mentioned plurality is the single target code command which carries out the trigger of the verification of the above-mentioned qualification.

[Claim 18]An addition step which needs for suitable execution of the above-mentioned software module the above-mentioned target code command which carries out the trigger of the above-mentioned qualification verifying means is also performed, As a result, a program product for managing qualification grant of a description to Claim 17 in which the above-mentioned addition step is not performed, but a software module is no longer appropriately performed when the above-mentioned software module is corrected by removing the above-mentioned target code command.

[Claim 19]The above-mentioned target code command which carries out the trigger of the above-mentioned qualification verifying means includes product number of the above-mentioned software module, A program product for managing qualification grant of a description to Claim 17 which has the product locking table in which the above-mentioned qualification verifying means on the above-mentioned computer systems was provided with a plurality of entries for which each relates to product number including qualification grant information in computer systems.

[Claim 20]Have a means to receive qualification for performing a software module, and it responds to a trigger means in the above-mentioned software module, In a method of distributing a software module that the above-mentioned software module is performed on computer systems which have a qualification verifying means which verifies that the above-mentioned computer systems have the qualification for performing the above-mentioned software module,

A method characterized by comprising the following of distributing a software module.

A stage of arranging an independent trigger means of plurality which carries out the trigger of the verification of qualification in the above-mentioned software module.

A stage of distributing the above-mentioned software module to the above-mentioned computer systems.

A stage of giving qualification for performing the above-mentioned software module to the above-mentioned computer systems.

[Claim 21]The above-mentioned stage of arranging an independent trigger means of the above-mentioned plurality in the above-mentioned software module, A method of distributing a

software module of a description to Claim 20 including arranging a single target code command which carries out the trigger of the verification of the above-mentioned qualification in a plurality of separate positions in the above-mentioned software module.

[Claim 22]An addition step which needs for suitable execution of the above-mentioned software module the above-mentioned target code command which carries out the trigger of the above-mentioned qualification verification is also performed, As a result, a method of distributing a software module of a description to Claim 21 that the above-mentioned addition step is not carried out but a software module is no longer appropriately performed when the above-mentioned software module is corrected by removing the above-mentioned target code command.

[Claim 23]The above-mentioned target code command which carries out the trigger of the above-mentioned qualification verification includes product number of the above-mentioned software module, A method of distributing a software module of a description to Claim 21 of having the product locking table in which the above-mentioned qualification verifying means on computer systems was provided with a plurality of entries for which each relates to product number including qualification grant information in computer systems.

[Claim 24]A method characterized by comprising the following of distributing a software module of a description to Claim 20.

The above-mentioned stage of giving qualification for performing the above-mentioned software module to the above-mentioned computer systems, A stage of comprising a plurality of data bits and providing information which enables it to judge whether the above-mentioned computer systems having the qualification for performing the above-mentioned software module and of generating a qualification grant key.

A stage of transmitting the above-mentioned qualification grant key to computer systems.

[Claim 25]The above-mentioned stage where the above-mentioned computer systems generate a qualification grant key including an original identifier, A method of distributing a software module of a description to Claim 24 of generating the above-mentioned qualification grant key using the above-mentioned original identifier, and giving qualification for performing software only on computer systems with which the above-mentioned qualification grant key contains the same original identifier.

[Claim 26]The above-mentioned stage of distributing the above-mentioned software module has the independent trigger means of plurality which carries out the trigger of the qualification verification on a respectively single storage medium, The above-mentioned stage of giving qualification for performing the above-mentioned software module to the above-mentioned computer systems including distributing a plurality of software modules, A method of distributing a software module of a description to Claim 20 including giving separate qualification to at least two of a plurality of above-mentioned software modules on the above-mentioned single storage medium.

[Detailed Description of the Invention]

[0001]

[Industrial Application]The present invention relates to use of computer software, and relates to a computer user restricting more specifically that license software can be used in the form where a license is not followed.

[0002]

[Description of the Prior Art]present-day computer systems are continued for the time which it cannot finish counting -- technically -- and it is the very complicated machine which adopted the result of skill on programming to the design. They can be divided roughly into hardware and software although computer systems include many components. Hardware is a material circuit, a board, a cable, memory storage, enclosure, etc. which constitute a system. However, the same with the ability of the scenario which can win the Pulitzer prize with a typewriter not to be written, hardware is itself and cannot solve the problem in the actual world. Hardware needs the command which tells what you should do. "Software" means the command which makes hardware perform useful work in the reasonable pure form. (However, software may be vaguely applied to the medium memorized and distributed). Software as well as hardware is a product of human being's originality and creativity. High quality software needs the remarkable creativity by the side of the maker called a programmer, training, and intelligence. The curriculum of "computer science" and a similar subject of study is provided in order for many universities to teach people the technology which creates software. This whole industry held thousands of persons' carrier, and by the time it creates the software which does useful work, it will have grown up. As a result of spending an immense labor on development of software, the worth of software which is some computer systems of exceeding worth of hardware is not new, either.

[0003]One of the characteristics of present-day computer systems is being able to transmit and copy data easily at high speed dramatically. Generally, this is required and useful capability. However, this means that the software which applied to which and created the labor for many years can reproduce with what [for 1 second / 1/], and may be abused with comparatively inexpensive magnetic media again. People who have not got permission can reproduce worthy software by few expenses and labors. This custom is generally called software copyright infringement. In order to inhibit software copyright infringement in recent years, various laws which impose the duty on criminal law and Civil Code have been enacted. However, since it is cut by temptation [software can copy comparatively easily, and] to come to do so since software is expensive, the copyright infringement of software still poses a problem.

[0004]In the case of the inexpensive software for the small computers called a "personal computer" by which extensive distribution is carried out, usually, the license to use of software is accepted at the lump sum payment charge which the same amount fixed to all the customers. This is not performed so much in a large-sized mainframe computer system. Such software for large-sized systems may require the code of millions of lines, and needs very a lot of investment for development and maintenance. If fixed sufficient single lump sum payment charges for a developer to recoup this investment are set up, it tends to become so expensive for many petty users that it cannot be used. Therefore, in the case of a mainframe, usually, the license fee of the software based on the amount used is charged. Such one method called "gradual price setting" charges the charge based on the performance of a customer's machine according to the variable tariff structure. A license fee higher than the customer of the machine which is inferior in performance is paid to the same software as the customer who uses a high-speed machine rather than more terminals were connected. Another usual custom charges a charge separately to maintenance upgrade of software.

[0005]Considering the level of a know how required to create high-definition software, and the quantity of the time concerning it, and a labor, money is required for creation of such software. When the developer of software is not rewarded as compared with the training and efforts, those who create software stop there being. It is not only actually the upper request to protect the investment in the software which the developer performed, but it is also a moral request.

Therefore, it is difficult for the proper owner of software to use it without permission of software, and he has asked for development of the software distribution method that the developer of software is properly rewarded to the product.

[0006]Software is distributed by the way a large number differ from a proper owner. These distribution methods can be divided roughly into three groups, the unrestricted qualification giving method, the restriction qualification giving method, and the non-qualification giving method, from a viewpoint of preventing unauthorized use.

[0007]Unrestricted qualification grant means that the distributed software runs without restriction by every machine which was an object of the design. The owner who distributes the software of unrestricted qualification grant has to distribute only the software which has the qualification for a user paying it a remuneration and performing it to each user. What bars reproducing in the form where the receiving area of such software is not permitted this, or using it is legal, and only a contractual obligation. In the inexpensive software to which it is licensed at a lump sum payment charge, it is the distribution method with this most ordinary which is used with most personal computers.

[0008]Restriction qualification grant means that software builds in a certain kind of restriction which restricts that a user copies it and can perform on without limit many machines. There are some different restriction qualification giving methods. One of them is copy restrictions which restrict the number of the copies which can be created from the distributed original copy. Although copy restrictions realize protection of a certain level against software copyright infringement, they have some defect. It is necessary to distribute only the software which has the qualification for the user also performing copy restrictions to each user in an owner like unrestricted qualification grant. This strikes not a full proof but a protection feature, and the program which can make the literal copy of the software with which copy protection of the bush was carried out also exists. Finally, this prevents him from the ability for a user to make a proper copy or run software from a high speed storage for the object of backup. Another restriction qualification giving method codes information peculiar to a user or a machine in the software itself. When performing software, a machine inspects in order to confirm that software is permitted to the machine or user. This method realizes protection, without preventing a user from the ability of a proper copy to be made. However, this compiles each copy of the distributed software uniquely, respectively, and puts it on a distribution medium, and since it must ship, a very expensive distribution system is needed.

[0009]The non-qualification giving method means that distributed software is made into the disable and needs the qualification grant separately distributed for performing. A certain non-qualification giving method may avoid distribution of custom software completely. However, such not all methods necessarily have such capability. For example, the owner can distribute the many software program of the same set to all those customers on a single general term medium, and can distribute separately the individualized permission key for which it allows running only the program for which the customer substituted payment to each customer. Although the non-qualification giving method avoids many problems which accompany other methods, the present design has a large possibility of receiving counterfeit and the remarkable performance degradation of qualification grant. When the most, the mechanism used in order to prohibit the qualification grant for performing software needs to concentrate a qualification verifying means in a software module (as data or a command), in order to avoid the overhead on verification called performance degradation. Depending on the case, the overhead of this qualification grant may be based on mounting of a product identification child's size, and the protection routine

within the distributed software. Or while the overhead makes it run software, it may be based on the necessity of performing a complicated decoding means again. As a result of concentration of a such qualification inspection, when an experienced programmer invalidates the small part as which "patching, i.e., a target code," was chosen, it is comparatively easy to invalidate a protection feature. When another, a target code does not perform qualification verification but the safety call course of identifying a module by creating a bit signature from it performs it. Although patching is made hard to receive, this is called unescapable and causes the serious performance degradation of a mechanism.

[0010]The protection method currently taught by the prior art makes a compromise of a protecting level and facility. Although it is possible to obtain protection of a high level comparatively by coding information peculiar to a machine in software, each software copy distributed must pay the sacrifice used as an original thing of maintaining a very complicated distribution system. Although more inexpensive distribution is also possible, the sacrifice that a part of protection is lost must be paid. A possibility that protection of a high level will be realized, and it can distribute easily using extensive distribution technique, and system performance and a user will make the copy for a backup copy properly, and the method which does not bar other required functions unfairly are called for. Simultaneously, gradual price setting and the method of supporting a separate license fee for every version that different software differs are also called for.

[0011]

[Problem to be solved by the invention]There is the object of this invention in providing the method and equipment which manage use of the software in computer systems and which have been improved.

[0012]Other objects of the present invention are providing protection of a high level to the unauthorized software using in computer systems rather than.

[0013]Other objects of the present invention are to reduce the expense which protects software against unauthorized use.

[0014]Other objects of the present invention are to reinforce the performance of the computer systems which perform software protected against unauthorized use.

[0015]Other objects of the present invention are to simplify the distribution system of the distribution person of software protected against unauthorized use.

[0016]Other objects of the present invention have such protection in providing the method and equipment which reduce the influences which it has on proper use of software and which protect software from unauthorized use.

[0017]Other objects of the present invention are to provide the method and equipment which protect software from unauthorized use and which have been improved, while a user allows making the proper copy for backup of software.

[0018]Other objects of the present invention are to make it difficult to change software so that unauthorized use may be attained.

[0019]Other objects of the present invention are to provide the method and equipment which distribute the software which carried out price setting gradually and which have been improved.

[0020]

[Means for solving problem]According to the present invention, software is distributed without the qualification grant for performing. Execution of software is attained by the enciphered qualification grant key which is distributed independently. This qualification grant key contains a plurality of qualification grant bits which instruct the consecutive numbers of the machine with

which software is licensed to it, and which software module has the qualification it runs by that machine. The safe decipherment mechanism is built in the machine. A safe decipherment mechanism takes out mechanical consecutive numbers, and it uses this as a key for decoding a qualification grant key. Subsequently, qualification grant information is memorized by the product locking table in a memory.

[0021]The distributed software contains a plurality of qualification verification triggers. In a preferable Example, each trigger is a simple-machine word command in a target code which identifies the product number of a software module. If meeting it with such a qualification verification trigger during execution of a software module, a machine will inspect the entry of the product locking table in which it corresponds to the product number of a software module. When it has the qualification a product runs, execution is continued normally, and execution is closed when that is not right. Since this verification requires only one machine language instruction, it can be performed without hardly affecting the performance of the whole system. As a result, most number of such qualification verification triggers can be arranged in a target code, and when someone does "patching" of this trigger, it can make it impossible to change a code as a matter of fact. In another Example, a qualification verification trigger also performs a certain useful work required for proper execution of a software module. By this, it becomes much more difficult to carry out "patching" of the software, and influence on the performance of such a verification trigger is further reduced.

[0022]Since the software itself does not include any qualification grants, restriction of distribution is not required. In a preferable Example, the distribution person of software can record a plurality of software modules on a single general term medium, and can distribute 1 set of recorded same modules to all those customers. Each customer receives the original qualification grant key to which he can perform only the software module to which it is licensed. Since it cannot be performed without a suitable key even if a certain module in which it is not licensed to a customer is given, it does not become a big problem. The customer can load software to the memory storage of others of his own system freely, or can create the copy for backup of software without limit.

[0023]

[Working example]The explanatory view of the main elements of a protection-of-software mechanism based on the preferable Example of the present invention is shown in Fig. 1. A customer's computer systems 101, The original identifier 105 which can be read, and a plurality of memory storage 106, 107, and 108 are included that the central processing unit (CPU) 102 closely combined with the control storage 103, the random access system memory 104, and elimination are impossible, and electronically. In a preferable Example, the original identifiers 105 are mechanical consecutive numbers. Although the memory storage 106-108 is a revolving magnetic disc memory in a preferable Example, it is also possible to use other storing technology. The computer systems 101 also contain again the console 109 and the software-media reader 110 which receive the input from an operator. Although one set of a console and one software-media reader and three memory storage are shown in Fig. 1, please understand that the actual number of such equipment attached to the system 101 is variable. Please understand that additional equipment is attached by the system 101. In a preferable Example, although the computer systems 101 are AS/400 computer systems of IBM, other computer systems can be used.

[0024]A software module is distributed separately from a qualification grant key (entitlement key) required to operate it. Originally, a software module is created on the computer systems 125

for development. The computer systems 125 for development contain the compiler 126 and the translator 127. A software module is recorded on the software recording medium 112, is stored in the warehouse 120, and is distributed to a customer from there. The enciphered qualification grant key 111 is distributed from a distribution person's sales office 121. The sales office 121 can access the computer systems 124 for marketing. The computer systems 124 for marketing include generation / enciphered program 122 of a qualification grant key, and the data base 123 containing customer data. Specifically, the data base 123 includes the information on the consecutive numbers of a machine required for generation of the enciphered qualification grant key, and the kind of processor. In a preferable Example, the computer systems 124 for marketing are computers which are placed in the center and communicate with a plurality of sales offices. However, the computer systems 124 for marketing can be placed at the sales office itself, and can also access the data base of a part or a center. Although the two separate computer systems 124 and 125 under a distribution person's control are shown in Fig.1, please understand that that single computer systems may be physically used performs both of functions.

[0025]The enciphered qualification grant key 111 is sent to a customer by mailing, a telephone, or other suitable means from the distribution person of software. Although it is possible to transmit a qualification grant key on magnetic media, such as a floppy disk, electronically, a key is so sufficiently [that an operator types on the console 109 and can input this into the system 101] brief.

[0026]In a preferable Example, a many software module is distributed on the same medium 112. The distribution person can give independently the qualification for using each module via a qualification grant key. The warehouse 120 stores 1 set of software modules generically on the medium 112. 1 set of same generic software modules are shipped irrespective of whether the license which uses which module for each customer is given. The qualification grant key distributed separately includes information for the system 101 to determine whether there is any qualification for performing which software module on it.

[0027]In a preferable Example, the software media 112 comprise one sheet or a plurality of read-only optical discs, and the medium reader 110 is an optical disc reader. However, please understand that an electronic distribution medium and other distribution media can also be used. If the software media 112 are received, a customer will load a desired software module to the system 101 from the medium reader 110, and will usually memorize the software module to the memory storage 106-108. The customer can generate one or more copies for backup of a software module on any suitable medium, and can memorize these to an archive. The software media 112 can be freely copied excluding the restriction to the copy and loading to a system, and loading is possible for them.

[0028]The contents of the qualification grant key 200 before encryption by a preferable Example are shown in Fig.2. This key contains the charge group field 201, the software version field 202, the key type field 203, the machine consecutive-numbers field 204, and the product qualification grant flag 205. The charge group field 201 is used for specifying one and supporting the gradual price setting of software from 16 kinds of machine stage values. The software version field 202 specifies the version level of the software with which qualification is given. The thing for which a charge is imposed separately in order to maintain upgrade of software is expected. The version specified in the qualification grant key 200 gives qualification to the software of all the former (lower level) levels from the version level. The key type field 203 is the region suspended for number extension of a different product of a key format and future change of the relevance of a key sake [a product] or supported. The machine consecutive-numbers field 204 contains the

consecutive numbers of the target machine [key / qualification grant]. The product qualification grant flag 205 is the 80-bit field where each contains 80 separate product flags with which it corresponds to product number. When qualification is given to the product number to which it corresponds, this bit is set to "1", and it is set to "0" when that is not right.

[0029]In a preferable Example, a software module is distributed as a compiled target code. The typical software module 300 is shown in Fig.3. This software module includes a plurality of target code commands which can be executed on the computer systems 101. According to the present invention, some qualification verification trigger commands (the following, a "qualification verification trigger", and the notation) 301 are included in the target code. All the qualification verification triggers 301 contained in a software module are the same. The qualification verification trigger 301 includes the operation code field 302, the version field 303, and the product number field 304. The field 305 is not used. The operation code field 302 is a verb portion of the target code command which identifies the operation which should be performed. The version field 303 identifies the version level of a software module. The product number field 304 identifies the product number relevant to the software module. Although the information on a version and product number changes for every module, a single instruction code is used for all the qualification verification triggers. In another Example, a qualification verification trigger is also the direct instruction (command which does not need to divide, does not need to be ordering the operand for the processing and does not need to be specified) which performs other useful work of a certain. In such an another Example, if a trigger command is executed, the system 101 will perform other operations of a certain simultaneously with qualification verification.

[0030]The computer systems 101 contain a means to receive and decode the enciphered qualification grant key 111, and a means to verify having the qualification for responding to a qualification verification trigger and a system performing a software module. In the preferable Example, these functions are divided between the system hardware of a different level, and the system software as shown in Fig.4. In this Example, the system 101 includes the hardware/software function of four levels called the hardware level 401, the horizontal-microcode level 402, the executable code level 403, and the virtual-machine level 404. The machine interface 405 has separated the virtual-machine level from all the lower level levels. The machine interface 405 is an interface which has at least the bottom defined to the customer in a level. That is, although the order bit of a virtual-machine level is defined to a customer, operation of the level below it is not defined. Therefore, a customer does not have the capability to change the command of the level below a machine interface level directly. The machine consecutive numbers 410 unchangeable [eternal] are stored in the hardware level 401.

[0031]The horizontal microcode 402 includes the microcode entry which interprets the instruction set which can be performed. This is physically memorized by the control storage 103, i.e., a preferable Example, by the read-only memory (ROM) in which change by a customer is impossible. The entry in a horizontal microcode is supporting the machine-key acquisition function 420, the lock setting up function 421, and the lock checking feature 422. The machine-key acquisition function 420 takes out the original identifier based on system consecutive numbers from a certain eternal hardware position in the preferable Example. The lock setting up function 421 accesses the product locking table 460, and changes the entry in a table. A lock setting up function is the only microcode function in which the product locking table 460 can be changed. The lock checking feature 422 verifies whether the product locking table 460 is accessed, one of the entries is read, and qualification is given.

[0032]The executable code level 403 includes the monitoring support of a lower level, and the support which carries out the command defined by the machine interface. Although this is physically memorized by the memory, since the internal specifications are not defined to the customer, it is unchangeable by a customer in practice. The lock release routine 430, the initial program load (IPL) lock re-verification routine 431, and the exception-handling routine 432 are contained in the support of a lower level. The unlocking routine 430 decodes the qualification grant key 111 using a peculiar machine key, and memorizes the enciphered qualification grant key 111 to the product key table 450. If initial setting of the system is carried out again, the initial-program load-locks re-verification routine 431 will take out the contents of the coded product key table 450, and will reconstruct the product locking table 460. The exception-handling routine 432 responds to the exception condition generated by the function of the horizontal-microcode level 402. The executable code level 403 contains the software module 300 of target code form.

[0033]Refer to the software for the virtual-machine level 404 as what is expressed by the machine language instruction. The computer of this level operates as a virtual machine in the meaning that immediate execution is not possible for a machine language instruction. Since at least the bottom which can be used for a customer is an interface of a level, although the module of the target code form which can be performed on a system is created, the compile course of a non-conventional type is required for the virtual-machine level 404. About this compile process, it describes behind. Strictly the software created via the compile course of this non-conventional type, Although the form of the executable target code which it is going to execute must be taken, usually the machine language instruction of a virtual-machine level expresses and discusses [that] like with immediate execution possible for a machine language instruction, and it is more intelligible. If this is established, it can be said that a user's software which the virtual-machine level 404 compiled is included. This also includes the support of the operating system of an upper level again rather than receiving the system 101. The support of this operation system contains two user interface routines required to support the input of a qualification grant key on the virtual-machine level 404. The general input routine 441 is used for processing an input in normal operation. The installation input routine 440 special to inputting a qualification grant key is required during the initial introduction of an operating system. The thing which needs this is because the portion of an upper level operating system is treated as other program products by the present invention from the machine interface level 405. Namely, such a portion has product number and the target code is subject to the influence of a qualification verification trigger. The installation input routine 440 is the only portion without a qualification verification trigger of an operating system.

Therefore, when introducing a system first, a qualification grant key can be input.

[0034]The software modules 300 are some program products of the compiled target code form which is performed on the system 101. This means being accessible for other objects of a virtual-machine level, and exists in them as an entity of the virtual-machine level 404. However, the code which can actually be executed operates on the executable code level 403 as shown by the frame of the broken lines. The executable code contains the qualification verification trigger 301 (only one is shown in the figure) performed by the lock checking feature 422 of a horizontal microcode.

[0035]The coded product key table 450 is shown in Fig.5. Recovering is possible, when it must intercept the electric power of a system or reinitialization must be carried out for other Reasons,

since the product key table 450 is contained in the random access memory 104 and is reproduced by the nonvolatile storage. The table 450 includes the 80 entries 501 to which each can correspond to each product number. The enciphered qualification grant key 502 by which each entry 501 is applied to the product number of the entry, The perfect copy of the time stamp 503 in which it is shown when the key was used first, the version number 504 and the charge group 505 of a key of the key, and the qualification grant bit 506 that shows whether the key unlocks a product is included. The time stamp 503 can be enciphered. The version number 504, the charge group 505, and the qualification grant bit 506 repeat the information included in the enciphered qualification grant key 502. They are contained on this table, in order to support checking. However, unless it is after verifying the information in the qualification grant key 502 enciphered first, the entry of the product locking table 460 cannot be changed and qualification of program execution cannot be given. Since each qualification grant key 502 in the product key table 450 is the enciphered form, in order to prevent access to this table of a user, it does not need a special measure.

[0036]The product locking table 460 is shown in Fig.6. This table is contained in the special lower address range of the random access memory 104 under perfect control of a horizontal microcode. The product locking table 460 includes the 80 entries 601 to which each can correspond to each product number. Each entry contains the version number which shows the level of the maximum version of qualification grant. It is shown that the version number of 0 does not have qualification grant in every version of a product. In order to further strengthen the degree of protection to change of the product locking table 460, the scramble of the version number can be carried out.

[0037]Next, operation of the software module on the computer systems 101 by the preferable Example of the present invention is described. There are four portions in this operation. First operation is arranged in the software module of the target code form that a plurality of qualification verification triggers can be performed. Second operation generates the enciphered qualification grant key 111 which permits access to a software module. The computer systems 101 receive, decode and memorize the qualification grant key 111, and the 3rd operation sets up the product locking table 460. The 4th operation makes the system 101 verify qualification, when a software module is performed on the system 101 and it meets with a qualification verification trigger. Two operations to begin are carried out under a software distribution person's management. Two next operations are carried out on a customer's system 101.

[0038]The software distribution person has to arrange a qualification verification trigger in a target code, when compiling a software module. This typical process in which it happens with the computer systems 125 for development is shown in Fig.7. Without including a qualification verification trigger, it usually passes along a source code and it is generated by the programmer. A source code is input into the compiler 126 at Step 701, and a program template is created at Step 702. The program template includes the machine language instruction of the virtual-machine level 404 (namely, level above the machine interface 405). At Step 704, a program template identifies the product number and version number, and it works as an input to the translator 127. Automatically, the translator 127 generates most number of qualification verification triggers, inserts this in the random position in a target code, and solves reference after insertion of a qualification verification trigger. The qualification verification trigger is contained in the software module of the target code form which is acquired as a result of being output at Step 705 and which can be performed. The software module of the form in which this execution is possible includes the target code command of the executable code level 403.

[0039]The process in which the enciphered qualification grant key is generated is shown in Fig.8. At Step 801, the order of the license over one piece or a plurality of program products of a certain version level generated by the sale charge member is input into the computer systems 124 for marketing. Theoretically, although a customer is able to order the product of a many version level, there is almost no Reason for generally doing so. However, if the version level with which customers differ is ordered since each qualification grant key operates only on the specified version level, it must generate a separate qualification grant key. If a customer's order is received, the key generation / enciphered program 122 under execution will access the database 123 including the consecutive numbers of the information about a customer, specifically a machine, and the information on processor form by the system 124 at Step 802. The charge group field 201 and the machine consecutive-numbers field 204 of the qualification grant key 200 which are not enciphered are generated using this information. At Step 803, the remaining fields are generated by a customer's order and the reference to the database of possible product number offer, and the qualification grant key of a perfect non-code form is built by them. Subsequently, key generation / enciphered program 122 enciphers a qualification grant key at Step 804 using one of some known encryption techniques by this technical field. Subsequently, the enciphered qualification grant key 111 which is Step 805 and was obtained as a result is transmitted to a customer. Although the key 111 is shown by Fig.1 as a plurality of binary bits, in order to simplify work which inputs a qualification grant key from a keyboard, binary bits, such as an equivalent by the hexadecimal digit or an alphanumeric character, may be shown to a customer in a certain form which carried out grouping.

[0040]The qualification grant key 111 is received on the computer systems 101, and the process in which the product locking table 460 is maintained is shown in Fig.9 a and Fig.9 b. At Step 901, a customer inputs the qualification grant key 111 into the computer systems 101 via the console 109. When this is initial introduction, the installation input routine 440 has a dialog with an operator, and receives an input. When that is not right, the general input routine 441 receives an input. A qualification grant key is passed to the lock release routine 430 which processes a decoding process. The lock release routine 430 makes the machine-key acquisition function 420 search machine consecutive numbers with Step 902, and makes it generate a machine key at it. Subsequently, the lock release routine 430 decodes the qualification grant key 111 at Step 903 using a machine key. Subsequently, at Step 904, the product key table 450 in which the lock release routine 430 was coded is reconstructed so that it may state below. The decoded qualification grant key takes the form shown in Fig.2. This contains 80-bit product qualification grant Flagg Alley 205 who shows whether the lock is released under the qualification grant key with the new product for every product number. It is considered that a new qualification grant key is the replaced key to all the products in which it releases a lock. The lock release routine 430 scans each product qualification grant Flagg 205 in the decoded qualification grant key (Step 904). When product qualification grant Flagg is set to "1" (those with qualification grant are pointed out) at Step 905, the entry to which it corresponds in the product key table 450 at Step 908 is replaced with a new qualification grant key, a version number, and a charge group value. The qualification grant bit field 506 are set to "1", and the stamp bit field 503 are set to the value of the zero which show that the qualification grant key has not been used yet at the time of the day /. When product qualification grant Flagg of a new key is "0", unless a version number and the charge group number are the same as what is memorized by the product key table 450, a new qualification grant key is ineffective. When a version number and the charge group number are the same (Step 906), a qualification grant key has an effect which locks a product. Therefore, in

order for the lock release routine 430 to be Step 907 and to set the version number in the product locking table 460 to "0". The lock setting up function 421 is called and the entry to which it corresponds in the product key table 450 at Step 908 is replaced with the value of a new qualification grant key. When the product key table 450 is reconstructed, it is Step 909 and the contents are saved to memory storage.

[0041]Unless the product given qualification before loses qualification, reconstruction of the product key table 450 does not have direct influence on the product locking table 460. A product is released in a lock according to a demand. An exception will be generated by the system, if it meets with a qualification verification trigger when performing the software product to which qualification is not given in front first. Subsequently, at Step 920, in order that the exception-handling routine 432 may try the lock release of a product, the lock release routine 430 is called. Next, the qualification grant key enciphered from the suitable entry in the product key table 450 in which the lock release routine 430 was coded at Step 921 is taken out, a machine key is obtained at Step 922, and a qualification grant key is decoded at Step 923. When qualification is given (Step 924), in order to set up a version number in the entry 601 of the product locking table 460 in which it is Step 926 and corresponds to the product number of a software product, the lock setting up function 421 is called. Simultaneously, the lock release routine 430 records the time of the 1st use on the stamp field 503 at Step 927 at the time of the day /. Subsequently, at Step 928, a qualification verification trigger is retried and execution of a program is continued. When qualification grant is not shown by Step 924, it is Step 925 and execution of a program is closed.

[0042]The product locking table 460 is memorized by RAM.

After reinitialization of a system does not remain.

During reinitialization ("IPL"), in order to reconstruct the product locking table 430, the IPL lock re-verification routine 431 is called. As mentioned above, this routine verifies qualification, and in order to reconstruct the entry in the product locking table 460 to which it corresponds, it obtains a machine key, and it decodes systematically each qualification grant key entry in the coded product key table 450.

[0043]The process by a preferable Example in which a software module is performed is shown in Fig.10. Execution of the software module by the system 101 is made by what this is taken out and performed for (Step 1002) (Step 1001) until a modular target code command is completed (Step 1003). When a command is the qualification verification trigger 301 (Step 1004), the lock checking feature 422 is called. At Step 1005, the lock checking feature 422 accesses the product locking table entry 601 to which it corresponds to the product number included in a qualification verification trigger. The qualification for the version number in the product locking table 460 being equal to the version number 303 contained in the qualification verification trigger 301, or performing software, in being larger than it is given (Step 1006). In this case, the lock checking feature 422 does not perform treatment beyond it, but a system proceeds to execution of the next target code command in a software module. When software is not given qualification, a lock checking feature generates an exception condition, control is passed to the exception-handling routine 432, and the exception-handling routine 432 terminates execution of a program (Step 1007). A system does not save the result of the qualification inspection which shows that software is given qualification. Therefore, if it meets with a qualification verification trigger again in a software module, a system will verify qualification again as mentioned above.

[0044]It is possible for another Example to define an addition so that a qualification verification trigger can be injected into a target code via the compile course of a conventional type. This

should serve as a bit of the command in which the execution as the module of target code form being performed by a system with same machine interface that a customer and a compiler maker can use is possible. or [invalidating a qualification verification trigger, in order that the format of a target code may support the compile course of the conventional type known by the customer] - - or it may become suitable to add the barrier to "patching" of a target code which is changed. One of such the additional barriers is defining a qualification verification trigger, as other functions of a certain are performed simultaneously. In this case, it is important that the alternate function carried out by the qualification verification trigger cannot carry out with other simple commands. This alternate function must be selected so that any compiled software modules may include some commands which perform that function quite reliably. When having coincided in these criteria, the compiler can generate automatically the target code which performs the alternate function (it is also a qualification verification trigger simultaneously with it) as a part of the usual compilation order. This definition should bring about the important barrier to "patching" of a target code which invalidates a qualification verification trigger. It is defining it as having to arrange other Examples to the addressable location which has a simple relation to the product number from which it discriminates a qualification verification trigger in a target code. It must be comparatively simple for a compiler to inject these commands into a suitable code position as a part of usual compile process, and it must be easy to perform this additional verification in the command embodiment. This definition should serve as an important barrier to patching of a target code which changes identification of the product number supplied in a qualification verification trigger.

[0045]The preferable Example is corresponding to 80 independent product number. Please understand that the actual number of the product number supported by the present invention is variable. In the preferable Example, it is also expected that the number of the software modules which can be compiled independently distributed by the distribution person exceeds 80 substantially. It corresponds to the number of the software packages in which the number of product number is provided by a distribution person's marketing organization and by which price setting was carried out independently. Although each software module has only one product number, many software modules which share this product number may exist. For example, a distribution person provides the word processing package containing the separate software module treating editing on screen, spell checking, document formatting, etc. When such a software module can always license as some word processing packages, those modules will have common product number. It is also possible to have a separate number for every software module in another Example.

[0046]A surcharge for software to maintain upgrade in a preferable Example, although licensed at a lump sum payment charge is able to be charged. In another Example, the license of the software between a certain periods is permissible. In such an another Example, a qualification grant key will include the field of the addition which shows the length of a period to which software is licensed. In order to judge whether the license completed the stamp as compared with the length of the period of a grant of license at the time of the day /relevant to the product number in the coded product key table 450, the IPL lock re-verification routine 431 is called periodically. Until new qualification grant is obtained by providing the lock release bit (not shown) for the entry in the product locking table 460 to which it corresponds, and setting this to "0", when the license expires at this time, Execution beyond it of a software module can be prevented.

[Effect of the Invention]The method and equipment which manage use of the software in computer systems by the present invention and which have been improved are provided.

[Brief Description of the Drawings]

A [FIG. 1] It is the figure showing the main elements of the protection-of-software mechanism by the preferable Example of the present invention.

A [FIG. 2] It is the figure showing the contents as which the qualification grant key by the preferable Example of the present invention is not enciphered.

A [FIG. 3] It is the figure showing the contents of the typical software module by a preferable Example which can be performed.

A [FIG. 4] It is the figure showing the structure of hardware required on a customer's computer systems in order to support the protection-of-software mechanism by a preferable Example, and software.

A [FIG. 5] It is the figure showing the format of the coded product key table by a preferable Example.

A [FIG. 6] It is the figure showing the format of the product locking table by a preferable Example.

A [FIG. 7] It is a block diagram of a step required to arrange the qualification verification trigger by a preferable Example to a software module.

A [FIG. 8] It is a block diagram of a step required to generate the enciphered qualification grant key according to a preferable Example.

A [FIG. 9] It is a block diagram of a step required to decode a qualification grant key on a customer's computer systems by a preferable Example, and maintain record of a qualification grant situation.

A [FIG. 10] It is a block diagram of a step required to verify qualification during the execution of a software module by a preferable Example.

[Explanations of letters or numerals]

101 Customer side computer system

102 Central processing unit (CPU)

103 Control storage

104 Random access memory (RAM)

106 Memory storage

107 Memory storage

108 Memory storage

109 Console

110 Medium reader

112 Software recording medium

120 Warehouse

121 Sales office

123 Data base

124 Computer systems for marketing

125 Computer systems for development

126 Compiler

127 Translator

Exhibit 6



INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification⁶ : H04L 9/00	A1	(11) International Publication Number: WO 97/26732 (43) International Publication Date: 24 July 1997 (24.07.97)
(21) International Application Number: PCT/US97/00651 (22) International Filing Date: 16 January 1997 (16.01.97) (30) Priority Data: 08/587,943 17 January 1996 (17.01.96) US (71) Applicant: THE DICE COMPANY [US/US]; 20191 E. Country Club Drive, Townhouse 4, Aventura, FL 33180 (US). (72) Inventors: MOSKOWITZ, Scott, A.; 20191 E. Country Club Drive, Townhouse 4, Aventura, FL 33180 (US). COOPERMAN, Marc; 2929 Ramona, Palo Alto, CA 94306 (US). (74) Agents: ALTMILLER, John, C. et al.; Kenyon & Kenyon, 1025 Connecticut Avenue, N.W., Washington, DC 20036 (US).	(81) Designated States: AL, AU, BA, BB, BG, BR, CA, CN, CU, CZ, EE, GE, HU, IL, IS, JP, KP, KR, LC, LK, LR, LT, LV, MG, MK, MN, MX, NO, NZ, PL, RO, SG, SI, SK, TR, TT, UA, UZ, VN, ARIPO patent (KE, LS, MW, SD, SZ, UG), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG). Published <i>With international search report.</i>	
(54) Title: METHOD FOR STEGA-CIPHER PROTECTION OF COMPUTER CODE (57) Abstract A method for protecting computer code copyrights by encoding the code into a data resource with a digital watermark. The digital watermark contains licensing information interwoven with essential code resources encoded into data resources. The result is that while an application program can be copied in an uninhibited manner, only the licensed user having the license code can access essential code resources to operate the program and any descendant copies bear the required license code.		

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.

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

METHOD FOR STEGA-CIPHER PROTECTION OF COMPUTER CODEFIELD OF INVENTION

With the advent of computer networks and digital
5 multimedia, protection of intellectual property has
become a prime concern for creators and publishers of
digitized copies of copyrightable works, such as musical
recordings, movies, video games, and computer software.
One method of protecting copyrights in the digital
10 domain is to use "digital watermarks."

The prior art includes copy protection systems
attempted at many stages in the development of the
software industry. These may be various methods by
which a software engineer can write the software in a
15 clever manner to determine if it has been copied, and if
so to deactivate itself. Also included are undocumented
changes to the storage format of the content. Copy
protection was generally abandoned by the software
industry, since pirates were generally just as clever as
20 the software engineers and figured out ways to modify
the software and deactivate the protection. The cost of
developing such protection was not justified considering
the level of piracy which occurred despite the copy
protection.

25 Other methods for protection of computer software
include the requirement of entering certain numbers or
facts that may be included in a packaged software's
manual, when prompted at start-up. These may be

overcome if copies of the manual are distributed to unintended users, or by patching the code to bypass these measures. Other methods include requiring a user to contact the software vendor and to receive "keys" for
5 unlocking software after registration attached to some payment scheme, such as credit card authorization. Further methods include network-based searches of a user's hard drive and comparisons between what is registered to that user and what is actually installed
10 on the user's general computing device. Other proposals, by such parties as AT&T's Bell Laboratories, use "kerning" or actual distance in pixels, in the rendering of text documents, rather than a varied number of ASCII characters. However, this approach can often
15 be defeated by graphics processing analogous to sound processing, which randomizes that information. All of these methods require outside determination and verification of the validity of the software license.

Digital watermarks can be used to mark each
20 individual copy of a digitized work with information identifying the title, copyright holder, and even the licensed owner of a particular copy. When marked with licensing and ownership information, responsibility is created for individual copies where before there was
25 none. Computer application programs can be watermarked by watermarking digital content resources used in conjunction with images or audio data. Digital watermarks can be encoded with random or pseudo random keys, which act as secret maps for locating the
30 watermarks. These keys make it impossible for a party to find the watermark without having the key. In addition, the encoding method can be enhanced to force a party to cause damage to a watermarked data stream when trying to erase a random-key watermark. Digital
35 watermarks are described in "Steganographic Method and Device" - The DICE Company, Serial No. 08/489,172, the disclosure of which is hereby incorporated by reference.

Other information is disclosed in "Technology: Digital Commerce", Denise Caruso, New York Times, August 7, 1995; and "Copyrighting in the Information Age", Harley Ungar, ONLINE MARKETPLACE, September 1995, Jupiter
5 Communications.

Additionally, other methods for hiding information signals in content signals, are disclosed in U.S. Patent No. 5,319,735 - Preuss et al. and U.S. Patent No. 5,379,345 - Greenberg.

10 It is desirable to use a "stega-cipher" or watermarking process to hide the necessary parts or resources of the executable object code in the digitized sample resources. It is also desirable to further modify the underlying structure of an executable
15 computer application such that it is more resistant to attempts at patching and analysis by memory capture. A computer application seeks to provide a user with certain utilities or tools, that is, users interact with a computer or similar device to accomplish various tasks
20 and applications provide the relevant interface. Thus, a level of authentication can also be introduced into software, or "digital products," that include digital content, such as audio, video, pictures or multimedia, with digital watermarks. Security is maximized because
25 erasing this code watermark without a key results in the destruction of one or more essential parts of the underlying application, rendering the "program" useless to the unintended user who lacks the appropriate key. Further, if the key is linked to a license code by means
30 of a mathematical function, a mechanism for identifying the licensed owner of an application is created.

It is also desirable to randomly reorganize program memory structure intermittently during program run time, to prevent attempts at memory capture or object code
35 analysis aimed at eliminating licensing or ownership information, or otherwise modifying, in an unintended manner, the functioning of the application.

In this way, attempts to capture memory to determine underlying functionality or provide a "patch" to facilitate unauthorized use of the "application," or computer program, without destroying the functionality and thus usefulness of a copyrightable computer program can be made difficult or impossible.

It is thus the goal of the present invention to provide a higher level of copyright security to object code on par with methods described in digital watermarking systems for digitized media content such as pictures, audio, video and multimedia content in its multifarious forms, as described in previous disclosures, "Steganographic Method and Device" and "Human Assisted Random Key Generation and Application for Digital Watermark System", filed on even date herewith, the disclosure of which is hereby incorporated by reference.

It is a further goal of the present invention to establish methods of copyright protection that can be combined with such schemes as software metering, network distribution of code and specialized protection of software that is designed to work over a network, such as that proposed by Sun Microsystems in their HotJava browser and Java programming language, and manipulation of application code in proposed distribution of documents that can be exchanged with resources or the look and feel of the document being preserved over a network. Such systems are currently being offered by companies including Adobe, with their Acrobat software. This latter goal is accomplished primarily by means of the watermarking of font, or typeface, resources included in applications or documents, which determine how a bitmap representation of the document is ultimately drawn on a presentation device.

The present invention includes an application of the technology of "digital watermarks." As described in previous disclosures, "Steganographic Method and

Device" and "Human Assisted Random Key Generation and Application for Digital Watermark System," watermarks are particularly suitable to the identification, metering, distributing and authenticating digitized content such as pictures, audio, video and derivatives thereof under the description of "multimedia content." Methods have been described for combining both cryptographic methods, and steganography, or hiding something in plain view. Discussions of these technologies can be found in Applied Cryptography by Bruce Schneier and The Code Breakers by David Kahn. For more information on prior art public-key cryptosystems see US Pat No 4,200,770 Diffie-Hellman, 4,218,582 Hellman, 4,405,829 RSA, 4,424,414 Hellman Pohlig. Computer code, or machine language instructions, which are not digitized and have zero tolerance for error, must be protected by derivative or alternative methods, such as those disclosed in this invention, which focuses on watermarking with "keys" derived from license codes or other ownership identification information, and using the watermarks encoded with such keys to hide an essential subset of the application code resources.

SUMMARY OF THE INVENTION

It is thus a goal of the present invention, to provide a level of security for executable code on similar grounds as that which can be provided for digitized samples. Furthermore, the present invention differs from the prior art in that it does not attempt to stop copying, but rather, determines responsibility for a copy by ensuring that licensing information must be preserved in descendant copies from an original. Without the correct license information, the copy cannot function.

An improvement over the art is disclosed in the present invention, in that the software itself is a set of commands, compiled by software engineer, which can be

configured in such a manner as to tie underlying
functionality to the license or authorization of the
copy in possession by the user. Without such
verification, the functions sought out by the user in
5 the form of software cease to properly work. Attempts
to tamper or "patch" substitute code resources can be
made highly difficult by randomizing the location of
said resources in memory on an intermittent basis to
resist most attacks at disabling the system.

10

DETAILED DESCRIPTION

An executable computer program is variously
referred to as an application, from the point of view of
a user, or executable object code from the point of view
15 of the engineer. A collection of smaller, atomic (or
indivisible) chunks of object code typically comprise
the complete executable object code or application which
may also require the presence of certain data resources.
These indivisible portions of object code correspond
20 with the programmers' function or procedure
implementations in higher level languages, such as C or
Pascal. In creating an application, a programmer writes
"code" in a higher level language, which is then
compiled down into "machine language," or, the
25 executable object code, which can actually be run by a
computer, general purpose or otherwise. Each function,
or procedure, written in the programming language,
represents a self-contained portion of the larger
program, and implements, typically, a very small piece
30 of its functionality. The order in which the programmer
types the code for the various functions or procedures,
and the distribution of and arrangement of these
implementations in various files which hold them is
unimportant. Within a function or procedure, however,
35 the order of individual language constructs, which
correspond to particular machine instructions is
important, and so functions or procedures are considered

indivisible for purposes of this discussion. That is, once a function or procedure is compiled, the order of the machine instructions which comprise the executable object code of the function is important and their order
5 in the computer memory is of vital importance. Note that many "compilers" perform "optimizations" within functions or procedures, which determine, on a limited scale, if there is a better arrangement for executable instructions which is more efficient than that
10 constructed by the programmer, but does not change the result of the function or procedure. Once these optimizations are performed, however, making random changes to the order of instructions is very likely to "break" the function. When a program is compiled, then,
15 it consists of a collection of these sub-objects, whose exact order or arrangement in memory is not important, so long as any sub-object which uses another sub-object knows where in memory it can be found.

The memory address of the first instruction in one
20 of these sub-objects is called the "entry point" of the function or procedure. The rest of the instructions comprising that sub-object immediately follow from the entry point. Some systems may prefix information to the entry point which describes calling and return
25 conventions for the code which follows, an example is the Apple Macintosh Operating System (MacOS). These sub-objects can be packaged into what are referred to in certain systems as "code resources," which may be stored separately from the application, or shared with other
30 applications, although not necessarily. Within an application there are also data objects, which consist of some data to be operated on by the executable code. These data objects are not executable. That is, they do not consist of executable instructions. The data
35 objects can be referred to in certain systems as "resources."

When a user purchases or acquires a computer program, she seeks a computer program that "functions" in a desired manner. Simply, computer software is overwhelmingly purchased for its underlying
5 functionality. In contrast, persons who copy multimedia content, such as pictures, audio and video, do so for the entertainment or commercial value of the content. The difference between the two types of products is that multimedia content is not generally interactive, but is
10 instead passive, and its commercial value relates more on passive not interactive or utility features, such as those required in packaged software, set-top boxes, cellular phones, VCRs, PDAs, and the like. Interactive digital products which include computer code may be
15 mostly interactive but can also contain content to add to the interactive experience of the user or make the underlying utility of the software more aesthetically pleasing. It is a common concern of both of these creators, both of interactive and passive multimedia
20 products, that "digital products" can be easily and perfectly copied and made into unpaid or unauthorized copies. This concern is especially heightened when the underlying product is copyright protected and intended for commercial use.

25 The first method of the present invention described involves hiding necessary "parts" or code "resources" in digitized sample resources using a "digital watermarking" process, such as that described in the "Steganographic Method and Device" patent application.
30 The basic premise for this scheme is that there are a certain sub-set of executable code resources, that comprise an application and that are "essential" to the proper function of the application. In general, any code resource can be considered "essential" in that if
35 the program proceeds to a point where it must "call" the code resource and the code resource is not present in memory, or cannot be loaded, then the program fails.

However, the present invention uses a definition of "essential" which is more narrow. This is because, those skilled in the art or those with programming experience, may create a derivative program, not unlike
5 the utility provided by the original program, by writing additional or substituted code to work around unavailable resources. This is particularly true with programs that incorporate an optional "plug-in architecture," where several code resources may be made
10 optionally available at run-time. The present invention is also concerned with concentrated efforts by technically skilled people who can analyze executable object code and "patch" it to ignore or bypass certain code resources. Thus, for the present embodiment's
15 purposes, "essential" means that the function which distinguishes this application from any other application depends upon the presence and use of the code resource in question. The best candidates for this type of code resources are NOT optional, or plug-in
20 types, unless special care is taken to prevent work-arounds.

Given that there are one or more of these essential resources, what is needed to realize the present invention is the presence of certain data resources of a
25 type which are amenable to the "stega-cipher" process described in the "Steganographic Method and Device" patent application. Data which consists of image or audio samples is particularly useful. Because this data consists of digital samples, digital watermarks can be
30 introduced into the samples. What is further meant is that certain applications include image and audio samples which are important to the look and feel of the program or are essential to the processing of the application's functionality when used by the user.
35 These computer programs are familiar to users of computers but also less obvious to users of other devices that run applications that are equivalent in

some measure of functionality to general purpose computers including, but not limited to, set-top boxes, cellular phones, "smart televisions," PDAs and the like. However, programs still comprise the underlying
5 "operating systems" of these devices and are becoming more complex with increases in functionality.

One method of the present invention is now discussed. When code and data resources are compiled and assembled into a precursor of an executable program
10 the next step is to use a utility application for final assembly of the executable application. The programmer marks several essential code resources in a list displayed by the utility. The utility will choose one or several essential code resources, and encode them
15 into one or several data resources using the steganographic process. The end result will be that these essential code resources are not stored in their own partition, but rather stored as encoded information in data resources. They are not accessible at run-time
20 without the key. Basically, the essential code resources that provide functionality in the final end-product, an executable application or computer program, are no longer easily and recognizably available for manipulation by those seeking to remove the underlying
25 copyright or license, or its equivalent information, or those with skill to substitute alternative code resources to "force" the application program to run as an unauthorized copy. For the encoding of the essential code resources, a "key" is needed. Such a key is
30 similar to those described in the "Steganographic Method and Device." The purpose of this scheme is to make a particular licensed copy of an application distinguishable from any other. It is not necessary to distinguish every instance of an application, merely
35 every instance of a license. A licensed user may then wish to install multiple copies of an application, legally or with authorization. This method, then, is to

choose the key so that it corresponds, is equal to, or is a function of, a license code or license descriptive information, not just a text file, audio clip or identifying piece of information as desired in digital watermarking schemes extant and typically useful to stand-alone, digitally sampled content. The key is necessary to access the underlying code, i.e., what the user understands to be the application program.

The assembly utility can be supplied with a key generated from a license code generated for the license in question. Alternatively, the key, possibly random, can be stored as a data resource and encrypted with a derivative of the license code. Given the key, it encodes one or several essential resources into one or several data resources. Exactly which code resources are encoded into which data resources may be determined in a random or pseudo random manner. Note further that the application contains a code resource which performs the function of decoding an encoded code resource from a data resource. The application must also contain a data resource which specifies in which data resource a particular code resource is encoded. This data resource is created and added at assembly time by the assembly utility. The application can then operate as follows:

- 1) when it is run for the first time, after installation, it asks the user for personalization information, which includes the license code. This can include a particular computer configuration;
- 2) it stores this information in a personalization data resource;
- 3) Once it has the license code, it can then generate the proper decoding key to access the essential code resources.

Note that the application can be copied in an uninhibited manner, but must contain the license code issued to the licensed owner, to access its essential code resources. The goal of the invention, copyright

protection of computer code and establishment of responsibility for copies, is thus accomplished.

This invention represents a significant improvement over prior art because of the inherent difference in use
5 of purely informational watermarks versus watermarks which contain executable object code. If the executable object code in a watermark is essential to an application which accesses the data which contains the watermark, this creates an all-or-none situation.
10 Either the user must have the extracted watermark, or the application cannot be used, and hence the user cannot gain full access to the presentation of the information in the watermark bearing data. In order to extract a digital watermark, the user must have a key.
15 The key, in turn, is a function of the license information for the copy of the software in question. The key is fixed prior to final assembly of the application files, and so cannot be changed at the option of the user. That, in turn, means the license
20 information in the software copy must remain fixed, so that the correct key is available to the software. The key and the license information are, in fact, interchangeable. One is merely more readable than the other. In the earlier developed "Steganographic Method and Device," the possibility of randomization erasure
25 attacks on digital watermarks was discussed. Simply, it is always possible to erase a digital watermark, depending on how much damage you are willing to do to the watermark-bearing content stream. The present
30 invention has the significant advantage that you must have the watermark to be able to use the code it contains. If you erase the watermark you have lost a key piece of the functionality of the application, or even the means to access the data which bear the
35 watermark.

A preferred embodiment would be implemented in an embedded system, with a minimal operating system and

memory. No media playing "applets," or smaller sized applications as proposed in new operating environments envisioned by Sun Microsystems and the advent of Sun's Java operating system, would be permanently stored in the system, only the bare necessities to operate the device, download information, decode watermarks and execute the applets contained in them. When an applet is finished executing, it is erased from memory. Such a system would guarantee that content which did not contain readable watermarks could not be used. This is a powerful control mechanism for ensuring that content to be distributed through such a system contains valid watermarks. Thus, in such networks as the Internet or set-top box controlled cable systems, distribution and exchange of content would be made more secure from unauthorized copying to the benefit of copyright holders and other related parties. The system would be enabled to invalidate, by default, any content which has had its watermark(s) erased, since the watermark conveys, in addition to copyright information, the means to fully access, play, record or otherwise manipulate, the content.

A second method according to the present invention is to randomly re-organize program memory structure to prevent attempts at memory capture or object code analysis. The object of this method is to make it extremely difficult to perform memory capture-based analysis of an executable computer program. This analysis is the basis for a method of attack to defeat the system envisioned by the present invention.

Once the code resources of a program are loaded into memory, they typically remain in a fixed position, unless the computer operating system finds it necessary to rearrange certain portions of memory during "system time," when the operating system code, not application code, is running. Typically, this is done in low memory systems, to maintain optimal memory utilization. The

MacOS for example, uses Handles, which are double-indirect pointers to memory locations, in order to allow the operating system to rearrange memory transparently, underneath a running program. If a computer program
5 contains countermeasures against unlicensed copying, a skilled technician can often take a snapshot of the code in memory, analyze it, determine which instructions comprise the countermeasures, and disable them in the stored application file, by means of a "patch." Other
10 applications for designing code that moves to prevent scanning-tunnelling microscopes, and similar high sensitive hardware for analysis of electronic structure of microchips running code, have been proposed by such parties as Wave Systems. Designs of Wave Systems'
15 microchip are intended for preventing attempts by hackers to "photograph" or otherwise determine "burn in" to microchips for attempts at reverse engineering. The present invention seeks to prevent attempts at understanding the code and its organization for the
20 purpose of patching it. Unlike systems such as Wave Systems', the present invention seeks to move code around in such a manner as to complicate attempts by software engineers to reengineer a means to disable the methods for creating licensed copies on any device that
25 lacks "trusted hardware." Moreover, the present invention concerns itself with any application software that may be used in general computing devices, not chipsets that are used in addition to an underlying computer to perform encryption. Wave Systems' approach
30 to security of software, if interpreted similarly to the present invention, would dictate separate microchip sets for each piece of application software that would be tamperproof. This is not consistent with the economics of software and its distribution.

35 Under the present invention, the application contains a special code resource which knows about all the other code resources in memory. During execution

time, this special code resource, called a "memory scheduler," can be called periodically, or at random or pseudo random intervals, at which time it intentionally shuffles the other code resources randomly in memory, so
5 that someone trying to analyze snapshots of memory at various intervals cannot be sure if they are looking at the same code or organization from one "break" to the next. This adds significant complexity to their job. The scheduler also randomly relocates itself when it is
10 finished. In order to do this, the scheduler would have to first copy itself to a new location, and then specifically modify the program counter and stack frame, so that it could then jump into the new copy of the scheduler, but return to the correct calling frame.
15 Finally, the scheduler would need to maintain a list of all memory addresses which contain the address of the scheduler, and change them to reflect its new location.

The methods described above accomplish the purposes of the invention - to make it hard to analyze captured
20 memory containing application executable code in order to create an identifiable computer program or application that is different from other copies and is less susceptible to unauthorized use by those attempting to disable the underlying copyright protection system.
25 Simply, each copy has particular identifying information making that copy different from all other copies.

What is Claimed Is:

1 1. A method of associating executable object code with
2 a digital sample stream by means of a digital watermark
3 wherein the digital watermark contains executable object
4 code and is encoded into the digital sample stream.

1 2. The method of claim 1 wherein a key to access the
2 digital watermark is a function of a collection of
3 license information pertaining to the software which is
4 accessing the watermark

5 where license information consists of one or more
6 of the following items:

7 Owing Organization name;
8 Personal Owner name;
9 Owner Address;
10 License code;
11 Software serialization number;
12 Distribution parameters;
13 Appropriate executable general computing
14 device architecture;
15 Pricing; and
16 Software Metering details.

1 3. The method of claim 1 further comprising the step
2 of transmitting the digital sample stream, via a
3 transmission means, from a publisher to a subscriber
4 wherein transmission means can selected from the
5 group of

6 soft sector magnetic disk media;
7 hard sector magnetic disk media;
8 magnetic tape media;
9 optical disc media;
10 Digital Video Disk media;
11 magneto-optical disk media;
12 memory cartridge;
13 telephone lines;

14 SCSI;
15 Ethernet or Token Ring Network;
16 ISDN;
17 ATM network;
18 TCP/IP network;
19 analog cellular network;
20 digital cellular network;
21 wireless network;
22 digital satellite;
23 cable network;
24 fiber optic network; and
25 electric powerline network.

1 4. The method of claim 1 where the object code to be
2 encoded is comprised of series of executable machine
3 instructions which perform the function of
4 processing a digital sample stream for the purpose
5 of modifying it or playing the digital sample stream.

1 5. The method of claim 3 further comprising the steps
2 of:
3 decoding said digital watermark and extracting
4 object code;
5 loading object code into computer memory for the
6 purpose of execution;
7 executing said object code in order to process said
8 digital sample stream for the purpose of playback.

1 6. A method of assembling an application to be
2 protected by watermark encoding of essential resources
3 comprising the steps of:
4 assembling a list of identifiers of essential
5 code resources of an application where identifiers allow
6 the code resource to be accessed and loaded into memory;
7 providing license information on the
8 licensee who is to receive an individualized copy of the
9 application;

10 storing license information in a
11 personalization resource which is added to the list of
12 application data resources;
13 generating a digital watermark key from
14 the license information; using the key as a pseudo-
15 random number string to select a list of suitable
16 digital sample data resources, the list of essential
17 code resources, and a mapping of which essential code
18 resources are to be watermarked into which data
19 resources;
20 storing the map, which is a list of
21 paired code and data resource identifiers, as a data
22 resource, which is added to the application;
23 adding a digital watermark decoder code
24 resource to the application, to provide a means for
25 extracting essential code resource from data resources,
26 according to the map;
27 processing the map list and encoding
28 essential code resources into digital sample data
29 resources with a digital watermark encoder;
30 removing self-contained copies of the
31 essential code resources which have been watermarked
32 into data resources; and
33 combining all remaining code and data
34 resources into a single application or installer.

1 7. A method of intermittently relocating application
2 code resources in computer memory, in order to prevent,
3 discourage, or complicate attempts at memory capture
4 based code analysis.

1 8. The method of claim 7 additionally comprising the
2 step of
3 assembling a list of identifiers of code resources
4 of an application where identifiers allow the code
5 resource to be accessed and loaded into memory.

1 9. The method of claim 8 additionally comprising the
2 step of modifying application program structure to make
3 all code resource calls indirectly, through the memory
4 scheduler, which looks up code resources in its list and
5 dispatches calls.

1 10. The method of claim 9 additionally comprising the
2 step of intermittently rescheduling or shuffling all
3 code resources prior to or following the dispatch of a
4 code resource call through the memory scheduler.

1 11. The method of claim 10 additionally comprised of
2 the step of the memory scheduler copying itself to a new
3 location in memory.

1 12. The method of claim 11 additionally comprising the
2 step of modifying the stack frame, program counter, and
3 memory registers of the CPU to cause the scheduler to
4 jump to the next instruction comprising the scheduler,
5 in the copy, to erase the previous memory instance of
6 the scheduler, and changing all memory references to the
7 scheduler to reflect its new location, and to return
8 from the copy of the scheduler to the frame which called
9 the previous copy of the scheduler.

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US97/00651

A. CLASSIFICATION OF SUBJECT MATTER																										
IPC(6) : H04L 9/00 US CL : 380/54 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. : 380/54, 2, 4, 9, 21, 23, 25, 28, 49, 50, 59; 283/73, 113, 17																										
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)																										
C. DOCUMENTS CONSIDERED TO BE RELEVANT																										
Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.																								
Y	US 5,349,655 A (MANN) 20 September 1994, see Abstract.	1																								
X	US 4,262,329 A (BRIGHT et al) 14 April 1981, see Abstract.	7																								
<input type="checkbox"/> Further documents are listed in the continuation of Box C. <input type="checkbox"/> See patent family annex.																										
<table border="0"> <tr> <td colspan="2">* Special categories of cited documents:</td> <td>*T</td> <td>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</td> </tr> <tr> <td>*A</td> <td>document defining the general state of the art which is not considered to be of particular relevance</td> <td>*X</td> <td>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</td> </tr> <tr> <td>*E</td> <td>earlier document published on or after the international filing date</td> <td>*Y</td> <td>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</td> </tr> <tr> <td>*L</td> <td>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)</td> <td>*Z</td> <td>document member of the same patent family</td> </tr> <tr> <td>*O</td> <td>document referring to an oral disclosure, use, exhibition or other means</td> <td></td> <td></td> </tr> <tr> <td>*P</td> <td>document published prior to the international filing date but later than the priority date claimed</td> <td></td> <td></td> </tr> </table>			* 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)	*Z	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		
* 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)	*Z	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 04 APRIL 1997		Date of mailing of the international search report 29 APR 1997																								
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 <i>Bernarr Earl Gregory</i> BERNARR EARL GREGORY Telephone No. (703) 306-4153																								

Form PCT/ISA/210 (second sheet)(July 1992)*

Exhibit 7



United States Patent [19]
Hasebe et al.

[11] **Patent Number:** **5,935,243**
 [45] **Date of Patent:** ***Aug. 10, 1999**

- [54] **LICENSEE NOTIFICATION SYSTEM**
- [75] Inventors: **Takayuki Hasebe; Naoya Torii**, both of Kawasaki, Japan
- [73] Assignee: **Fujitsu Ltd.**, Kawasaki, Japan
- [*] Notice: This patent issued on a continued prosecution application filed under 37 CFR 1.53(d), and is subject to the twenty year patent term provisions of 35 U.S.C. 154(a)(2).
- [21] Appl. No.: **08/673,108**
- [22] Filed: **Jul. 1, 1996**
- [30] **Foreign Application Priority Data**
 Aug. 31, 1995 [JP] Japan 7-224338
- [51] Int. Cl.⁶ **G06F 11/00; H04L 9/00**
- [52] U.S. Cl. **713/200; 382/23**
- [58] **Field of Search** 380/3, 4, 23, 25, 380/26; 395/186, 187.01, 188.01, 182.16, 182.18, 183.12

2 245 724 1/1992 United Kingdom .

OTHER PUBLICATIONS

European Search Report issued Oct. 6, 1997.

Primary Examiner—Robert W. Beausoliel, Jr.
Assistant Examiner—Nadeem Iqbal
Attorney, Agent, or Firm—Staas & Halsey

[57] **ABSTRACT**

There is disclosed a licensee notification system for implementing a software sales system wherein license information for converting to executable form software that is presented to a user in non-executable form is communicated to the user from a management center on condition of payment of a charge, and the software is converted into executable form at the user terminal using this license information. The subject of the licensee notification system is software that decides whether or not the correspondence relationship between user identification information and signature information stored in the license file is legitimate, and, if it is legitimate, displays the user identification information to the user before starting proper operation; or, if it is not legitimate, does not start proper operation. The licensee notification system is constituted by connecting the management center and user terminals by communication circuits. If license information is requested from the user terminal, the management center transmits license information combining in integral form the user identification information identifying the user and conversion information for converting the software to executable form. The user terminal enables the software using the conversion information contained in this license information and writes user identification information and signature information whose content is determined in accordance with the content of the user identification information to a license file that is referred to when this software is operating.

[56] **References Cited**

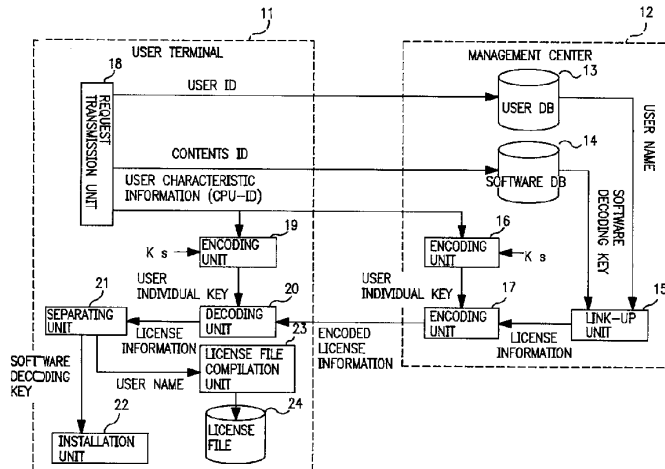
U.S. PATENT DOCUMENTS

4,866,769	9/1989	Karp	380/4
5,103,476	4/1992	Waite et al. .	
5,204,897	4/1993	Wyman	380/4
5,287,408	2/1994	Samson	380/4
5,291,598	3/1994	Grundy .	
5,319,705	6/1994	Halter et al. .	
5,479,612	12/1995	Kenton et al.	385/186
5,757,907	5/1998	Cooper et al.	380/4

FOREIGN PATENT DOCUMENTS

0 367 700 A2	5/1990	European Pat. Off. .
0 613 073 A1	8/1994	European Pat. Off. .

14 Claims, 6 Drawing Sheets



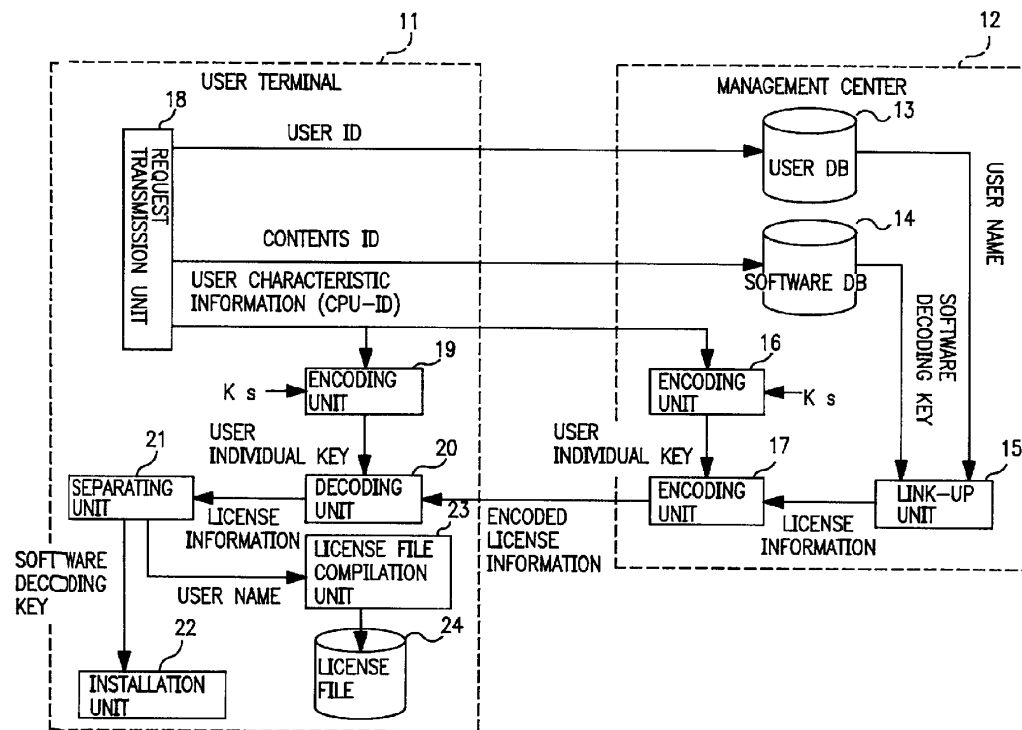


FIG. 1

FIG. 2

13

USER ID	USER NAME
M0001111	TOKKYO TARO

FIG. 3

14

CONTENTS ID	DECODING KEY
ABC00001	xxxxxxxxxx

FIG. 4

24

CONTENTS ID	USER NAME	SIGNATURE INFORMATION
ABC00001	TOKKYO TARO	zzzzzzzz

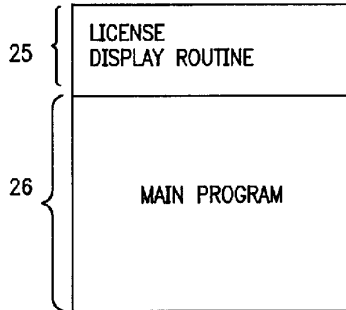


FIG. 5

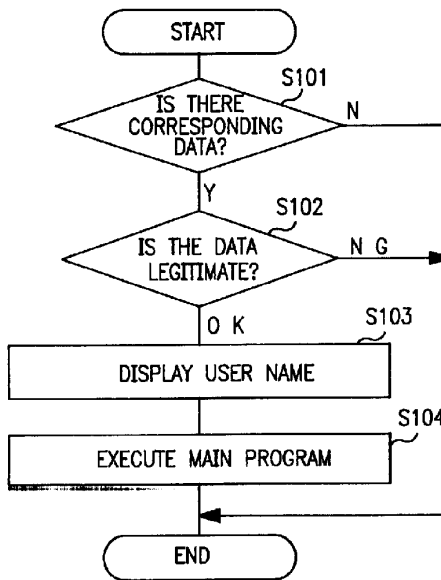


FIG. 6

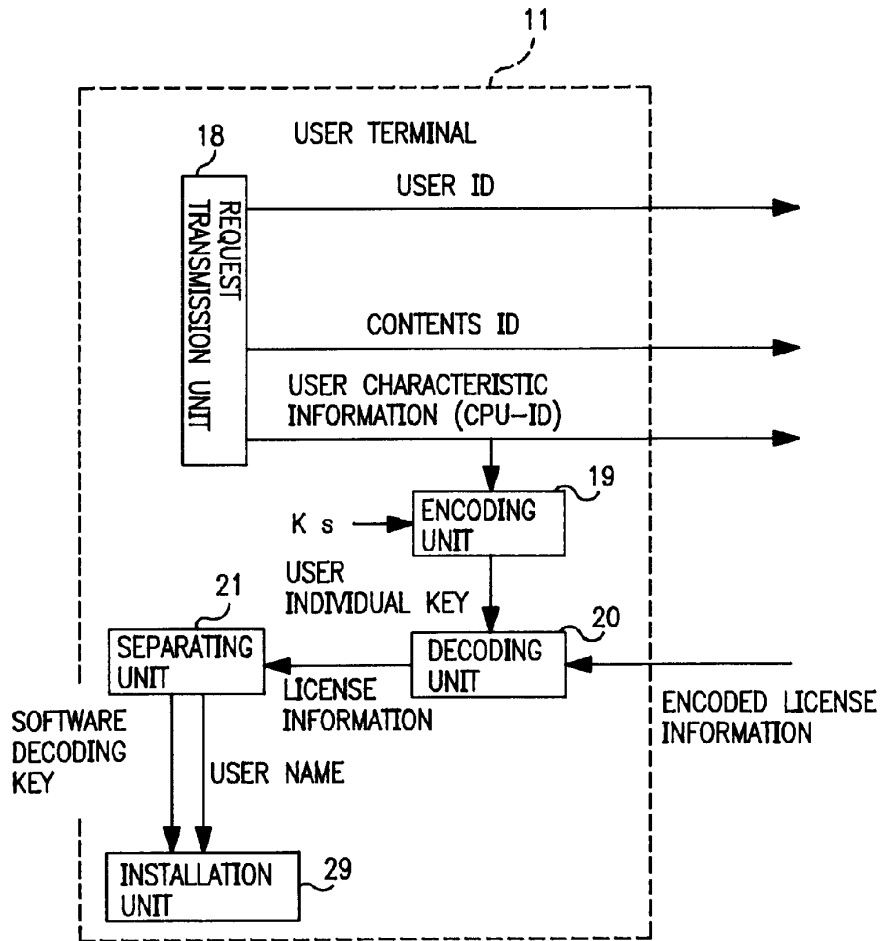


FIG. 7

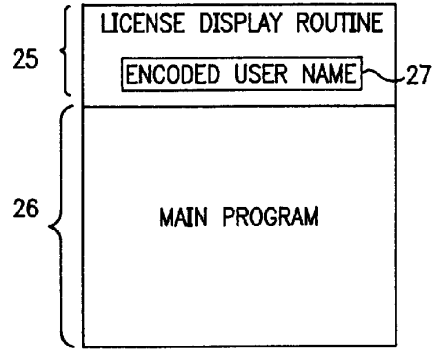


FIG. 8

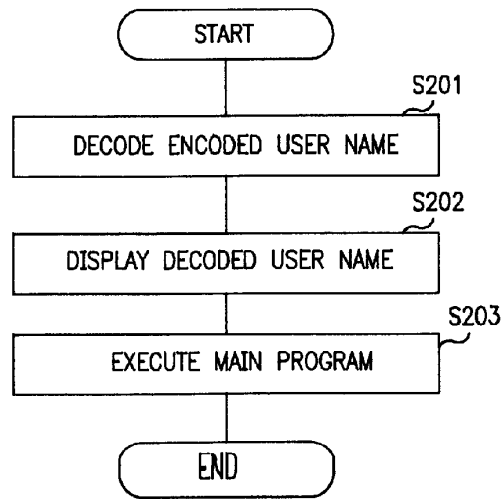


FIG. 9

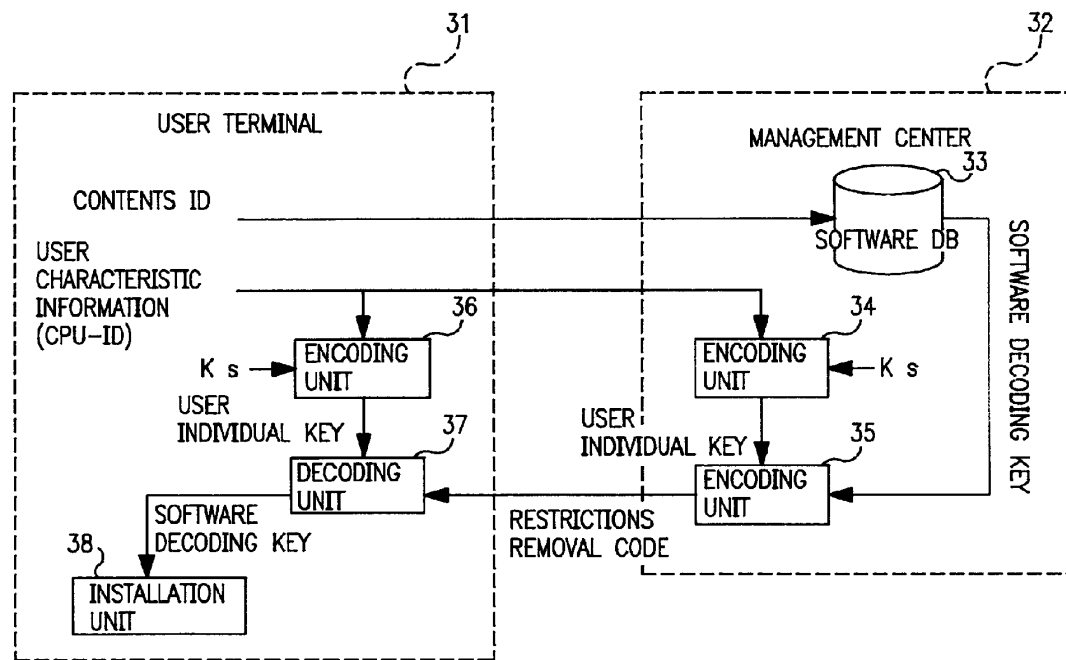


FIG. 10
PRIOR ART

1

LICENSEE NOTIFICATION SYSTEM

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to a licensee notification system employed for the sale of software using a high speed communication network such as B-ISDN and a large-capacity storage medium such as a CD-ROM.

2. Description of the Related Art

With the development of high speed communication technology such as B-ISDN (broad-band integrated services digital network) and high-capacity storage media such as CD-ROMs (compact disk read only memory) such means can now be used to distribute computer programs or video data or audio data. For example, video works which were previously supplied on video tape are now being sold stored on CD-ROM. Also, game programs etc, which contain a large amount of picture data, are being sold stored on CD-ROM. The same applies to high speed communication networks, in which the software supplier can now distribute the software by various methods. One of these methods of software sales is the so-called "locked software" sales system. In the locked software sales system, a CD ROM on which are stored a large number of software items whose functions are restricted is sold cheaply. By using the various items of software on the CD-ROM that is purchased, in a condition with the functional restrictions imposed, the end user is able to make a decision as to whether or not he needs each software item. Then, if the end user does require the software, he obtains (purchases) a restriction-removal code corresponding to this software from a management center operated by the software distributor, and is able to use this restriction-removal code to remove the functional restrictions on the software.

Such a sales system may be implemented, as a specific example, using the software sales system shown in FIG. 10. As shown in this Figure, this software sales system comprises user terminals 31 and management center 32. The user terminal 31 and the management center 32 are connected by means of a communication circuit.

When actually purchasing the software (i.e. when purchasing a restriction-removal code), the end user, using a user ID etc, sets up a communication path with the management center and executes the prescribed procedure required to request that a restriction-removal code be sent to the user terminal 31. This procedure includes the input of a "contents ID", which is information for identifying the software item that is to be purchased actually. In response to the execution of such a procedure, the user terminal 31 sends to the management center 32 the contents ID and for example the characteristic information of the user, consisting of the ID of the CPU provided in user terminal 31.

Within the management center 32, there is provided a software database (software DB) in which software decoding keys employed for encoding the various software items are stored in association with the contents ID. When a contents ID is received from user terminal 31, the software decoding key corresponding to the contents ID is read from software database 33. Also, encoding unit 34 in management center 32 generates a user individual key by encoding the user characteristic information from user terminal 31 by the key "Ks". Encoding unit 35 sends the results of the encoding of the software decoding key from software database 33 to user terminal 31 as restriction-removal code, using the user individual key from encoding unit 34.

Encoding unit 36 in user terminal 31 generates a user individual key by encoding the user characteristic informa-

2

tion with the key "Ks". Decoding unit 37 uses the user individual key generated by encoding unit 36 to decode the restriction removal code from management center 32, thereby generating the software decoding key. Installation unit 38 then uses this software decoding key to decode the software in CD-ROM corresponding to the contents ID sent to center terminal 32: thus the software is put in a condition where it can be used with the functional restrictions removed, and, in this form, is installed on to a storage device such as a hard disk device.

With such a software sales system, it is possible to determine the software item to be purchased after actually ascertaining its contents: thus, the possibility that the purchased software might be completely different from that intended, as could happen if the purchase were made solely on the basis of the details contained in a catalogue, can be completely eliminated. Also, since the software on the CD ROM is stored in a form which is not executable without knowing special information, illicit installation can be prevented.

However, once the software has been installed, it is an extremely easy operation to copy this. Thus, the problem has arisen of unscrupulous persons copying the software without the consent of the software supplier. Various methods (so-called protection methods) of preventing such illicit copying are known but there is no way to prevent illicit copying by a person possessing knowledge at the level of the BIOS (basic input/output system). Whichever method is used, it can do no more than make it more difficult to perform illicit copying.

For this reason, software is sold in which the name of the authorized user is displayed on start-up, with the object of preventing illicit copying psychologically rather than physically. That is, the aim is to prevent illicit copying of software by displaying the name of the authorized user of the software when the illicitly copied software is executed.

However, even with such software, if the copying is inclusive of the installation software that sets the user name, when the software is run, it can be made to display the name of the person who made the illicit copy: thus, sufficient effectiveness in preventing illicit copying was not obtained.

SUMMARY OF THE INVENTION

An object of the present invention is to provide a licensee notification system whose psychological effectiveness in preventing illicit copying is very high.

A first licensee notification system according to the present invention consists in a system for implementing a software sales system in which software in non-executable form is presented to a user, and license information for converting the software into executable form is informed to the user on condition of payment of a charge, and the software is converted into executable form using this license information.

The first licensee notification system is constituted of a management center and user terminals; its subject is software which includes instructions that command a terminal to read user identification information in a license file and to notify the user identification information to the user on commencement of its operation.

The management center comprises a license information generating unit that generates license information combining in integrated form user identification information that specifies a user and conversion information for converting software to executable form.

The user terminal comprises a storage unit, a conversion unit, and license file creating unit. In more detail, the storage

3

unit is employed for storing the license file and software converted to executable form. The license information, which is generated by the license information generating unit in the management center, is given to the conversion unit. The conversion unit then converts the software to executable form using the license information and installs it in the storage unit. The license file creating unit creates the license file which contains the user identification information contained in the license information, and stores the license file in the storage unit.

That is, in the first licensee notification system, software is installed in the user terminal so that the user identification information of the legitimate user is notified to the user on its start-up, using the license information which is generated in the management center and contains the user identification information.

A second licensee notification system according to the present invention is constituted of a management center and user terminal; its subject is software which includes instructions that commands the user terminal to read user identification information in the prescribed location in the software and to notify the user identification information to the user on commencement of its operation.

The management center comprises a license information generating unit that generates license information combining in integrated form user identification information identifying a user and conversion information for converting software into executable form.

The user terminal comprises a storage unit, a conversion unit and a software rewriting unit. Of these, the storage unit is employed for storing the software after this has been converted to executable form. The conversion unit converts the software to executable condition using the license information generated by the license information generating unit in the management center, and then installs it in the storage unit. The software rewriting unit rewrites the information of the prescribed location of the software that has been installed by the conversion unit with the user identification information contained in the license information.

That is, in this second licensee notification system, installation is performed with the content of the software rewritten such that the user identification information of the legitimate user is notified on start-up, using the license information which is generated in the management center and contains the user identification information.

The third licensee notification system according to the present invention has as its subject software that, on commencement of operation, includes instructions commanding the user terminal to read user identification information in a license file and to notify the user identification information to the user.

The management center in the third licensee notification system comprises a license information generating unit that generates license information consisting of an integral combination of conversion information for converting the software to executable form and user identification information identifying a user.

The user terminal comprises a storage unit for storing a license file, a license file creating unit, and a software execution unit. The license file creating unit creates the license file containing the license information generated by the license information generating unit, and stores the license file in the storing unit. The software execution unit, when execution of the software is designated, converts the software to executable form using the license information stored in the license file and expands it into memory, and commences operation in accordance with the expanded software.

4

That is, in the third licensee notification system, the software, which is presented to the user in non-executable form, is converted to executable form in accordance with the license information containing the user identification information every time execution is designated.

The fourth licensee notification system according to this invention is constituted of management center and user terminal. The subject of the system is software which judges the legitimacy of user identification information on the basis of signature information stored in a license file on commencement of operation and, if the user identification information is legitimate, commences proper operation after notifying this user identification information to the user, and, if the user identification information is not legitimate, terminates operation.

The management center comprises a license information generating unit that generates license information combining in integral form the user identification information identifying the user and signature information whose content is determined in accordance with the user identification information.

The user terminal comprises a storage unit for storing the license file and a license file creating unit that creates the license file containing the user identification information contained in the license information generated by the license information generating unit and stores the license file in the storage unit.

That is, in the fourth licensee notification system, the license information which is necessary for running the software normally is generated on the basis of the user identification information in the management center and is informed to the user terminal.

It may be noted that although in the first to the fourth licensee notification system any means could be employed for notification of the license information, if notification of license information is performed using a communication circuit, a system that is simple to operate can be formed.

Also, it is possible to employ information including the name of the user as user identification information. It is also possible to employ a unit that generates license information including user identification information encoded with a characteristic key of the software. In this case, software is presented to user which including instructions that command the user terminal to notify to the user the result of decoding the user identification information using the characteristic key.

In the first to the third licensee notification systems, it is also possible to make the software that is presented to the user encoded, and to make the conversion information for decoding the encoded software. Also, it is possible to employ, in such a licensee notification system, license information containing the user identification information in a form that cannot be separated without special information. For example, it is possible to employ information, as license information, which is the result of encoding the conversion information and user identification information, combined in integrated manner.

Also, it is possible to make the first to third licensee notification system a system whose subject is software that, if the signature information stored in the license file does not correspond to the user identification information, terminates operation, and, as the license file creating unit, to employ a unit that generates signature information whose content is determined in accordance with the content of the user identification information, and creates the license file containing the signature information. In this case, it can be made

5

more difficult to alter the user identification information that is notified to the user on start-up of the software. Also, in the case of such software, it is possible to employ as license information generating unit a unit that generates license information containing signature information whose contents are determined in accordance with the contents of the user identification information, and, as license file creating unit, to employ a unit that creates the license file containing signature information contained in the license information.

Also, it is possible to make the second licensee notification system a system whose subject is software that, if signature information stored in the second predetermined location does not correspond to user identification information stored in a prescribed location, terminates its operation, and, as software rewriting unit, to employ a unit that rewrites the information of the prescribed location of the software with the user identification information contained in the license information and that rewrites the information at the second prescribed location of the software with signature information whose content is determined in accordance with the user identification information. Also, in the case of such software, it is possible to employ as license information generating unit a unit that generates license information containing signature information whose content is determined in accordance with the content of the user identification information, and, as software rewriting unit, to employ a unit that rewrites information of the prescribed location with user identification information contained in the license information and that rewrites the information at the second prescribed location in the software by signature information contained in the license information.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a functional block diagram illustrating the layout of a licensee notification system according to a first embodiment of the present invention;

FIG. 2 is a diagram given in explanation of the content of the user database provided in the management center comprised in the licensee notification system according to the first embodiment;

FIG. 3 is a diagram illustrating the content of the software database provided in the management center comprised in the licensee notification system according to the first embodiment;

FIG. 4 is a diagram illustrating the content of a license file provided in a user terminal comprised in the licensee notification system according to the first embodiment;

FIG. 5 is a diagram illustrating the structure of software that is the subject of the licensee notification system according to the first embodiment;

FIG. 6 is a flow chart illustrating the operating sequence of software that is the subject of the licensee notification system according to the first embodiment;

FIG. 7 is a function block diagram illustrating the organization of a user terminal employed in the licensee notification system according to a second embodiment;

FIG. 8 is a diagram illustrating the structure of software that is the subject of the licensee notification system according to the second embodiment;

FIG. 9 is a flow chart showing the operating sequence of software that is the subject of the licensee notification system according to the second embodiment; and

FIG. 10 is a functional block diagram showing the structure of the licensee notification system used in a prior art locked software sales system.

6

DESCRIPTION OF THE PREFERRED EMBODIMENTS

The present invention is described in detail below with reference to the drawings.

First embodiment

FIG. 1 is a functional block diagram of a licensee notification system according to a first embodiment of the present invention. This licensee notification system is a system where CD-ROMs storing a large number of software items of restricted function are sold cheaply, and software sales are effected by selling the information needed to cancel the function restrictions of the software in this CD-ROM. Payment of the fee could be effected by for example notification of the subscriber number of a cash card or notification of a bank account withdrawal number or the like.

As shown in the drawings, the licensee notification system is constituted by user terminals 11 and management center 12 connected by means of a communication circuit. User terminals 11 and management center 12 may be described as computers and commence operation as an ensemble of the function blocks illustrated when prescribed programs are run.

First of all, the operation of management center 12 will be described.

Management center 12 is provided with two databases, called user database (user DB) 13 and software database (software DB) 14. As shown in FIG. 2, user DB 13 stores the correspondence relationship between the user ID, which is identification information given to users of this system by the manager, and the user name, which is the identification information of the user as employed in ordinary society. As shown in FIG. 3, software DB 14 stores the correspondence relationship between the contents ID, which is the identification information of each software item supplied and stored in the CD ROM, and the software decoding key, which is the decoding information needed to decode this software item.

A link-up unit 15 in management center 12 generates license information by combining the two data items: user name and software decoding key. An encoding unit 16 generates a user's individual key by encoding with key "Ks" the user characteristic information (details to be explained later) from user terminal 11. An encoding unit 17 generates coded license information by encoding the license information from link-up unit 15 using the user's individual key generated by encoding unit 16. In the present licensee notification system, a DES (data encryption standard) algorithm is employed for encoding and decoding.

The various function blocks that are not in management center 12 are arranged to operate synchronously when there is a request from user terminal 11 for information for removal of the function restrictions. Specifically, when management center 12 receives a request for information for removal of function restrictions relating to a software item from user terminal 11, it transmits to user terminal 11 coded license information containing the user's name and the software decoding key needed to remove the functional restrictions on the software item.

Next, the operation of user terminal 11 will be described. When user terminal 11 runs the programs for communication and installation, it executes the operation described below.

A request transmission unit 18 in user terminal 11 transmits to management center 12 information including the user ID, contents ID, and user's characteristic information.

Request transmission unit 18 commences operation when the keyboard (not shown) of user terminal 11 is operated in accordance with a prescribed procedure that is predetermined as the procedure for request of information for removal of functional restrictions. This request procedure includes keyboard input of the user ID and contents ID; request transmission unit 18 transmits to management center 12 the keyboard input information and the user's characteristic information, which is constituted by the ID of the CPU which is employed in user terminal 11.

As already explained, when a request for information for removal of functional restrictions is received from user terminal 11, management center 12 sends to user terminal 11 encoded license information. As a result, after request transmission unit 18 has been operated, user terminal 11 receives encoded license information from management center 12.

As shown in the drawings, the encoded license information is input to decoding unit 20 in user terminal 11. Decoding unit 20 also inputs the user's individual key, which is generated by encoding unit 19 using the user's characteristic information and "Ks". Using this user's individual key, decoding unit 20 decodes the encoded license information from center terminal 12. The license information, which is the result of this decoding, is input to separating unit 21, which is a unit that performs reverse processing against link-up unit 15 in management center 12. Separating unit 21 separates and extracts the software decoding key and user name from the license information, and respectively supplies the extracted software decoding key and user name to installation unit 22 and license file compilation unit 23.

Installation unit 22, using the software decoding key from separating unit 21, removes the functional restrictions on the specific software item (details to be described later) in accordance with the contents ID transmitted by request transmission unit 18. License file compilation unit 23 compiles a license file 24 using the user name and contents ID from separating unit 21.

FIG. 4 shows diagrammatically the contents of license file 24. As shown in the drawing, license file 24 stores information consisting of contents ID and user name, and signature information, which is information encoded using a signature key.

Further detailed description of the operation of installation unit 22 and the operation of the software installed by installation unit 22 is given below using FIG. 5 and FIG. 6. Of these Figures, FIG. 5 is a view showing diagrammatically the structure of software that is the subject of the present licensee notification system and FIG. 6 is a flow chart showing the operating sequence of the CPU in the user terminal when the software that is the subject of the present licensee notification system is actuated.

As shown in FIG. 5, the software that is the subject of the present system includes a license display routine 25 and main program 26. In the main program there are defined the operating procedures relating to the proper functions of this software; in license display routine 25, there is defined the content to be executed prior to execution of main program 26.

When this software is actuated, as shown in FIG. 6, the CPU, first of all, by checking the contents ID in the license file, decides whether or not data corresponding to the software that is being actuated is present in the license file (step S101). Then, if the corresponding data exists (step S101:Y), the CPU performs a check of the legitimacy of the corresponding data (step 102). In this step, the CPU encodes

the information consisting of contents ID and user name stored in the license file using the signature key that is set as data in license display routine 25, and if the result of this encoding agrees with the signature information, decides that the data is legitimate.

If it is legitimate (step S102:OK), the CPU displays the user name which is read from the license file (step S103), and commences operation in accordance with the main program (step S104).

Also, if the corresponding data is not present in the license file (step S101:N) or if the content of the license file is found to be not legitimate (step S102:NG), i.e. if the content of the license file is found to be different from the result of the compilation performed by license file compilation unit 23, the CPU terminates operation without displaying the user name or executing the main program.

As described above, with the licensee notification system according to the first embodiment, in the user terminal, installation of the software is performed such that the user name is displayed on start-up, using the encoded license information supplied from the management center. Also, the installed software is executed only when the legitimacy of the license file is verified. As a result, with this licensee notification system, even if the software and license file are copied illicitly after being installed, it is difficult to change the user name appearing on start-up of the software. The person who has made the illicit copy has no alternative but to use the software with the name of another person being displayed. As a result, illicit copying of the software can be prevented if the present licensee notification system is employed.

It should be noted that the licensee notification system of the first embodiment could be modified in various ways.

It would for example be possible to constitute the system such that notification of the contents ID etc to the management center and notification of the encoded license information to the user terminal were performed by another information transmission unit, such as the post. In this case, the user terminal is constituted such that installation is effected using encoded license information input from the keyboard. It is also possible to constitute the system such that the license information is notified in un-encoded form.

It is also possible to arrange that the signature information is generated at the management center end, and encoded license information containing this signature information is notified to the user terminal.

It is also possible to constitute the system such that, instead of the user name and signature information, information representing the user name in encoded form is stored in the license file, and, when the installed software is executed, the information in the license file is decoded by the software and displayed.

It would also be possible to arrange that the software was converted into executable condition not on installation of the software but rather every time execution of the software was specified, the software then being expanded in the memory and operation commenced in accordance with the software now in the memory.

Also, the medium whereby the software is supplied is not restricted to CD-ROM; a supply mode could be adopted in which the software was stored on another recording medium such as a floppy disk, or downloaded through a communication circuit.

Second embodiment

A licensee notification system according to a second embodiment of the present invention is described below

with reference to FIG. 7 thru FIG. 9. Of these Figures, FIG. 7 is a functional block diagram illustrating the layout of a user terminal wherein a licensee notification system according to the second embodiment is provided. FIG. 8 is a diagram illustrating the structure of software that is the subject of this licensee notification system. FIG. 9 is a flow chart showing the operating procedure of the CPU when the software that is the subject of the present licensee notification system is executed.

In the licensee notification system according to the second embodiment, a management center of the same construction as management center 12 in the first embodiment is employed. Also, as can be seen from the functional block diagram shown in FIG. 7, the difference of the action of the user terminal 11 is slight, so the description will be confined to the parts of which the details of operation differ with respect to the licensee notification system of the first embodiment.

As shown in FIG. 7, in user terminal 11 according to the second embodiment, the software decoding key and user name that are separated by separating unit 21 are both input to the installation unit 29. Installation unit 29 effects installation by decoding the software in the CD ROM using the software decoding key, and generates the user name in encoded form by encoding the user name. Thus, installation unit 29, as shown diagrammatically in FIG. 8, writes the encoded user name 27 that is thus generated in a prescribed location of license display routine 25.

As shown in FIG. 9, when the software that is the subject of the licensee notification system of the second embodiment is started up, the encoded user name that was written in the prescribed location in license display routine 25 is read and decoded (step S201). Then, after display of the decoded user name has been performed (step S202), main program 27 is executed (step S203).

That is, with this licensee notification system, the user name that is displayed on start-up of the software is set by directly rewriting the content of the software.

Even with the licensee notification system of this second embodiment, enabling of the software such that the user's name is displayed on start-up is effected independently of keyboard input from the user terminal, so it is not possible to alter the user name that is displayed by the software simply by making an illicit copy of the installation software. Also, the installed software is executed only when the legitimacy of the license file has been verified. Consequently, with this licensee notification system, even if the installed software is illicitly copied, it is difficult to alter the user name that is displayed on start-up, so the person who has made the illicit copy has no alternative but to use the software with another person's name displayed. Thus, use of this licensee notification system can psychologically prevent illicit copying.

It should be noted that with this licensee notification system according to the second embodiment, various modifications are possible just as in the case of the licensee notification system according to the first embodiment.

For example, it would be possible to constitute a system such that the notification of the contents ID etc to the management center and the notification of the encoded license information to the user terminal were performed by another information transmission unit such as the post. And it is also possible to constitute a system such that license information is notified in un-encoded form.

Also, it is possible to constitute a system such that the software in question is made software wherein operation is

stopped if signature information stored in a second prescribed location of the software does not correspond to user identification information stored in a first prescribed location and to arrange that the installation unit 29 writes the user name to the first prescribed location in the software and writes the signature information, consisting of this user name in encoded form, to the second prescribed location.

What is claimed is:

1. A licensee notification system for use in a software sales system in which software in non-executable form is presented to a user, and license information for converting the software into executable form is transmitted to the user on condition of payment of a charge, said licensee notification system comprising:

a management center that stores and maintains license information combining, in integrated form, conversion information for converting software to executable form and user identification information specifying the user; and

a user terminal including:

a storage;

a converter that converts the software in non-executable form into executable form using the license information maintained by said management center and installs the software in executable form into said storage; and

a license file creator that creates a license file containing the user identification information along with signature information created based on the user information and stores the license file in said storage, and

wherein the software includes instructions that command the user terminal to read user identification information in the license file and to notify the user identification information to the user on commencement of its operation and commands the user terminal to terminate operation if the signature information in the license file does not correspond to the user identification information in the license file.

2. A licensee notification system for use in a software sales system in which software in non-executable form is presented to a user, and license information for converting the software in non-executable form to executable form is informed to the user on condition of payment of a charge, said licensee notification system comprising:

a management center that stores and manages license information combining, in integrated form, conversion information for converting software to executable form and user identification information specifying a valid user; and

a user terminal including:

a storage;

a converter that converts the software in non-executable form into executable form using the license information maintained by said management center and installs the software in executable form in said storage; and

software rewriting means for rewriting the license information and user information in a prescribed location of the software and for rewriting signature information, determined in accordance with the user identification information, in a second location in the software; and

wherein the software includes instructions that commands the user terminal to read user identification information from the prescribed location in the software and to

11

notify the user identification information to the user on commencement of the software operation along with instructions that command the user terminal to terminate operation if the signature information stored in the second prescribed location does not correspond to the user identification information stored in the prescribed location.

3. A licensee notification system for use in a software sales system in which software in non-executable form is presented to a user, and license information for converting the software in non-executable form to executable form is transmitted to the user on condition of payment of a charge, said licensee notification system comprising:

a management center that stores and manages license information combining, in integrated form conversion, information for converting software to executable form and user identification information specifying the user; and

a user terminal including:

a storage;

a license file creator for creating a license file containing the user identification information and signature information created based on the content of the user identification information, and that stores the license file in said storage; and

software execution means for converting, when execution of the software is designated, the software into executable form using the license information in the license file and expanding the software in executable form into memory and executing operation in accordance with the software in the memory; and

wherein the software includes instructions that commands the user terminal to read user identification information in the license file and to notify the user identification information to a user on commencement of its operation and commands the user terminal to terminate operation if the signature information in the license file does not correspond to the user identification information in the license file.

4. A licensee notification system for use in a software sales system in which software that refers to license information is presented to a user, and the license information about the software is informed to the user on condition of payment of a charge, said licensee notification system comprising:

a management center that stores and maintains license information combining, in integrated form, user identification information specifying the user and signature information whose content is determined in accordance with the user identification information; and

a user terminal including:

a storage; and

license file creating means for creating the license file containing the license information maintained by said management center and that stores the license file in the storage; and

wherein the software includes instructions that command the user terminal to judge the legitimacy of the user identification information in the license file using the signature information in the license file on commencement of operation of the software and, if the user identification information is legitimate, to commence proper operation of the software after notifying the user identification information to the user, but, if the user identifica-

12

tion information is not legitimate, to stop the operation of the software.

5. A licensee notification system according to claim 1, wherein the software includes instructions that command the user terminal to display the user identification information on a display of the user terminal.

6. A licensee notification system according to claim 1, wherein the user terminal further comprises:

transmitting means for transmitting a request signal which requests license information to the management center through a communication circuit; and

said management center, when the request signal is received from the user terminal, generates license information and transmits the license information to the user terminal through the communication circuit.

7. A licensee notification system according to claim 1, wherein the user identification information includes the name of the user.

8. A licensee notification system according to claim 1, wherein the license information includes user identification information encoded with a characteristic key of the software; and

the software includes instructions that command the user terminal to inform to the user the result of decoding the user identification information using the characteristic key.

9. A licensee notification system according to claim 1, wherein the software is presented to the user in encoded form, and the conversion information is information for decoding the software.

10. A licensee notification system according to claim 1, wherein the license information contains the user identification information in a form that is incapable of being separated without special information.

11. A licensee notification system according to claim 1, wherein the license information is the result of encoding the conversion information and user identification information, combined in integrated manner.

12. A licensee notification system for use in a software sales system in which software in non-executable form is presented to a user, and license information for converting the software into executable form is transmitted to the user on condition of payment of a charge, said licensee notification system comprising:

a management center that stores and maintains license information combining, in integrated form, conversion information for converting software to executable form, user identification information specifying the user and signature information whose content is determined in accordance with the content of the user identification information; and

a user terminal including:

a storage;

a converter that converts the software in non-executable form into executable form using the license information maintained by said management center and installs the software in executable form into said storage; and

a license file creator that creates a license file containing the user identification information and signature information contained in the license information and stores the license file in said storage; and

wherein the software includes instructions that command the user terminal to read user identification information in the license file and to notify the user identification

13

information to the user on commencement of its operation along with instructions that command the user terminal to terminate operation if the signature information in the license file does not correspond to the user identification information in the license file.

13. A licensee notification system for use in a software sales system in which software in non-executable form is presented to a user, and license information for converting the software in non-executable form to executable form is informed to the user on condition of payment of a charge, said licensee notification system comprising:

a management center that stores and manages license information combining, in integrated form, conversion information for converting software to executable form, user identification information specifying a valid user and signature information determined in accordance with the content of the user identification information; and

a user terminal including

a storage;
a converter that converts the software in non-executable form into executable form using the license information maintained by said management center and installs the software in executable form in said storage; and

software rewriting means for rewriting the license information in a prescribed location of the software installed by said converter with the user identification information contained in the license information and rewriting the signature information in a second prescribed location in the software; and

wherein the software includes instructions that commands the user terminal to read user identification information in the prescribed location in the software and to notify the user identification information to the user on commencement of the software operation along with instructions that command the user terminal to terminate operation if the signature information stored in the second prescribed location does not correspond to the user identification information that is stored at the prescribed location.

14. A licensee notification system for use in a software sales system in which software in non-executable form is presented to a user, and license information for converting the software in non-executable form to executable form is

14

transmitted to the user on condition of payment of a charge, said licensee notification system comprising:

a management center that stores and manages license information combining, in integrated form conversion, information for converting software to executable form, user identification information specifying the user and signature information determined in accordance with the content of the user identification information; and

a user terminal including:

a storage;
license file creator for creating a license file containing the user identification information and signature information contained in the license information maintained by said management center, and that stores the license file in said storage; and

software execution means for converting, when execution of the software is designated, the software into executable form using the license information in the license file and expanding the software in executable form into memory and executing operation in accordance with the software in the memory; and

wherein the software includes instructions that commands the user terminal to read user identification information in the license file, notify the user identification information to a user on commencement of its operation and command the user terminal to terminate operation if the signature information in the license file does not correspond to the user identification information in the license file;

wherein said license information containing signature information whose content is determined in accordance with the content of the user identification information;

said license file creator creates the license file containing the signature information contained in the license information; and

the software includes instructions that command the user terminal to terminate operation if the signature information in the license file does not correspond to the user identification information in the license file.

* * * * *

Exhibit 9

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent of: Scott A. Moskowitz
U.S. Patent No.: 9,104,842
Issue Date: August 11, 2015
Appl. No.: 11/895,388
Filing Date: August 24, 2007
Title: DATA PROTECTION METHOD AND DEVICE
Control No.: To be assigned

Mail Stop Ex Parte Reexam
ATTN: Central Reexamination Unit
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

**DECLARATION OF DR. CLAUDIO T. SILVA IN SUPPORT OF REQUEST FOR
EX PARTE REEXAMINATION OF U.S. PATENT NO. 9,104,842**

TABLE OF CONTENTS

I. INTRODUCTION3

II. QUALIFICATIONS.....3

III. DOCUMENTS AND INFORMATION CONSIDERED7

IV. SUMMARY OF OPINIONS.....7

V. UNDERSTANDING OF THE LAW8

 A. Claim Construction8

 B. Anticipation under 35 U.S.C. § 1028

VI. PERSON OF ORDINARY SKILL IN THE ART8

VII. CLAIM CONSTRUCTION10

 A. “encoding algorithm” (claims 12-14)10

 B. “code resource” (claims 12-14)11

 C. “software code interrelationships” (claims 14)11

VIII. CLAIMS 11, 12, 13, AND 14 ARE ANTICIPATED IN VIEW OF THE
PRIOR ART.....12

 A. Claims 11, 12, 13, and 14 are Anticipated by Beetcher.12

 1. Beetcher Anticipates Independent Claim 11.....12

 2. Beetcher Anticipates Independent Claim 12.....22

 3. Beetcher Anticipates Independent Claim 13.....34

 4. Beetcher Anticipates Independent Claim 14.....41

 B. Claims 11, 12, 13, and 14 are Anticipated by Beetcher ’072.46

 1. Beetcher ’072 Anticipates Independent Claim 11.....46

 2. Beetcher ’072 Anticipates Independent Claim 12.....56

3. Beetcher Anticipates Independent Claim 13.....	67
4. Beetcher '072 Anticipates Independent Claim 14.....	73
C. Claims 11, 12, 13, and 14 are Anticipated by Cooperman.....	79
1. Cooperman Anticipates Independent Claim 11.	79
2. Cooperman Anticipates Independent Claim 12.	84
3. Cooperman Anticipates Independent Claim 13.	90
4. Cooperman Anticipates Independent Claim 14.	95
D. Claims 11, 12, 13, and 14 are Anticipated by Hasebe.....	102
1. Hasebe Anticipates Independent Claim 11.	102
2. Hasebe Anticipates Independent Claim 12.	109
3. Hasebe Anticipates Independent Claim 13.	117
4. Hasebe Anticipates Independent Claim 14.	122

I, Claudio T. Silva, declare as follows:

I. Introduction

1. I have been retained by Juniper Networks, Inc. as an independent expert consultant.

Although I am being compensated at my usual rate for the time I spend on this matter, no part of my compensation depends on the outcome of this proceeding, and I have no interest in the outcome of this proceeding.

2. I have been asked to consider whether claims 11-14 of U.S. Patent No. 9,104,842 (“the ‘842 Patent”) are valid in view of certain prior art discussed below. As I explain in more detail below, in my opinion, claims 11-14 are invalid in view of the prior art discussed in this declaration.

II. Qualifications

3. I am a Professor of Computer Science and Engineering and Data Science at New York University. Prior to my work in academia, I worked in industry for six years in the area of computer graphics and visualization. I received a Bachelor of Science in Mathematics from the Federal University of Ceará in Brazil, and a Ph.D. from State University of New York at Stony Brook in Computer Science. My curriculum vitae, which includes a more detailed account of my background, experience, and publications, is attached hereto (Ex. 9).

4. From July 1998 until July 2000, I served as an adjunct assistant professor in the Department of Applied Mathematics and Statistics at SUNY Stony Brook. From September 2002 until April 2006 I was an associate professor in the Department of Computer Science & Engineering at Oregon Health & Science University. From October 2003 until June 2011, I was a faculty member at the Scientific Computing and Imaging Institute at the University of Utah. From January 2008 until May 2009, I served as Associate Director at the University of Utah’s Scientific Computing and Imaging (SCI) Institute. I also served as an Associate Professor of

Computer Science from October 2003 until June 2010, and a Professor of Computer Science from July 2010 until June 2011 at the University of Utah. I am currently a Professor of Computer Science and Engineering at NYU's Tandon School of Engineering, a position I have held since July 2011 (when the school was called Polytechnic Institute of NYU). I also serve as a faculty member to a number of organizations within NYU, including the Center for Urban Science and Progress, the Center for Data Science, and Courant's Department of Computer Science.

5. Between 1998 and 2002, I worked in industry at the IBM T. J. Watson Research Center and AT&T Labs-Research. At both places, I worked on 3D data acquisition, modeling, and rendering techniques. As part of my activities at IBM, I was part of the MPEG-4 3D Model Coding (3DMC) standardization committee.

6. In 2011, I co-founded Modelo, Inc., a company that creates custom advanced 3-D modeling solutions for its clients.

7. I have published over 250 technical articles, most at highly competitive refereed conferences and rigorously reviewed journals. I currently serve as chair of the executive committee for the IEEE Computer Society Technical Committee on Visualization and Graphics. I also hold 12 U.S. patents. My publications have received awards from organizations and programs such as the IEEE Shape Modeling International, IEEE Visualization, EuroVis (a conference co-sponsored by Eurographics and the IEEE Visualization and Graphics Technical Committee), and Eurographics (the European Association for Computer Graphics).

8. My research has been funded by the National Science Foundation, the Department of Energy, the National Aeronautics and Space Administration, the National Institutes of Health, the Alfred P. Sloan Foundation, the Gordon and Betty Moore Foundation, Defense Advanced Research Projects Agency, AT&T, IBM, and MLB Advanced Media.

9. Regarding the subject matter of the '842 Patent relating to encoding and decoding license information into software applications, I have been an editor on several journals relating to digital encoding, such as Computer Graphics Forum, Computer and Graphics, IEEE Transactions on Visualization and Computer Graphics. I have been co-chair at several symposiums on digital encoding, such as the IEEE/SIGGRAPH Symposium on Volume Visualization and Graphics and the IEEE Parallel & Large-Data Visualization & Graphics Symposium. I have won several awards, such as Best Paper Award at the 2011 ACM Eurographics Symposium on Parallel Graphics and Visualization. I have been a member of program committees relating to digital encoding, such as the Pacific Graphics and Eurographics. I have also helped develop techniques, codes, and tools to enable new forms of encoding and decoding data with the MPEG-4 3D Model Coding (3DMC) standardization committee. Furthermore, I was the founding director of Graphics and Visualization Track at University of Utah's School of Computing. Lastly, I have done research on the subject of digital encoding.

10. With regard to these research projects, I have published several papers, including this small sample (please see my CV for many more):

- "Parallel Volume Rendering of Irregular Grids," Ph.D. thesis, State University of New York at Stony Brook (1996);
- "A Unified Infrastructure for Parallel Out-Of-Core Isosurface and Volume Rendering of Unstructured Grids," Y.-J. Chiang, R. Farias, C. Silva, and B. Wei, IEEE Parallel & Large-Data Visualization & Graphics Symposium, pages 59–66 (2001);

- “Out-Of-Core Sort-First Parallel Rendering for Cluster-Based Tiled Displays,” W. Corrêa, J. Klosowski, and C. Silva, *Parallel Computing*, Vol 29, pages 325–338 (2003);
- Image-Space Acceleration for Direct Volume Rendering of Unstructured Grids using Joint Bilateral Upsampling, S. P. Callahan and C. Silva, *Journal of Graphics, GPU, & Game Tools*, 14(1): page 115 (2009);
- Hardware Accelerated Simulated Radiography, D. Laney, S. Callahan, N. Max, C. Silva, S. Langer, and R. Frank. *IEEE Visualization 2005*, pages 343–350 (2005);
- “Multi-Fragment Effects on the GPU Using the k-Buffer,” L. Bavoil, S.P. Callahan, A. Lefohn, J.L.D. Comba, and C. Silva, *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 97–104 (2007);
- “Hardware-Assisted Visibility Sorting for Unstructured Volume Rendering,” S. Callahan, M. Ikits, J. Comba, and C. Silva, *IEEE Transactions on Visualization and Computer Graphics*, 11(3):285–295 (2005);
- *iWalk: Interactive Out-Of-Core Rendering of Large Models*, W. Correa, J. Klosowski, and C. Silva, Technical Report TR-653-02, Princeton University (2002);
- “Efficient Conservative Visibility Culling Using The Prioritized-Layered Projection Algorithm,” J. Klosowski and C. Silva, 7(4):365–379, *IEEE Transactions on Visualization and Computer Graphics* (2001); and
- “Efficient Compression of Non-Manifold Polygonal Meshes,” A. Gueziec, F. Bossen, G. Taubin, and C. Silva, 14(1–3):137–166, *Computational Geometry: Theory and Applications* (1999).

11. I have also taught graduate and undergraduate courses with a strong focus on digital encoding, courses which also cover topics related to cryptography and watermarking.

III. Documents and Information Considered

12. I have reviewed the '842 Patent, including the claims of the patent in view of the specification, and I have reviewed the '842 Patent's prosecution history. In addition, I have reviewed the following documents:

- U.S. Patent No. 5,933,497 ("Beetcher");
- Japanese Patent Application Publication No. H05334072 ("Beetcher '072");
- English Translation of Beetcher '072;
- U.S. Patent No. 5,935,243 ("Hasebe");
- PCT Application Publication No. WO 97/26732 ("Cooperman"); and
- Plaintiff Blue Spike LLC's Proposed Terms for Construction, Pursuant to Patent Rule (P.R.) 4-2 in *Blue Spike, LLC v. Juniper Networks, Inc.*, Case No. 6:17-cv-16-KNM (E.D. Tex.)

IV. Summary of Opinions

13. In my opinion, claims 11, 12, 13, and 14 of the '842 Patent are anticipated by the prior art. As I explain in more detail throughout this declaration, Beetcher anticipates every element of claims 11-14. Moreover, Beetcher '072 anticipates every element of claims 11-14. And as I further explain in this declaration, Cooperman anticipates every element of claims 11-14. Additionally, as I further explain in this declaration, Hasebe anticipates every element of claims 11-14. Therefore, claims 11-14 are invalid as anticipated by the prior art.

V. Understanding of the Law

14. Counsel has advised me of the legal concepts, summarized below, that are relevant to reexamination proceedings. I have applied those concepts in rendering my opinions in this declaration.

A. Claim Construction

15. I understand that during a reexamination of an unexpired patent, claim terms are accorded their broadest reasonable interpretation in light of the specification to a person of ordinary skill in the art at the time the invention was made. Counsel has advised me that the broadest reasonable interpretation must be consistent with the specification, and that claim language should be read in light of the specification and teachings in the underlying patent.

B. Anticipation under 35 U.S.C. § 102

16. I understand that anticipation of a claim requires that every element of a claim be disclosed expressly or inherently in a single prior art reference, and arranged in the prior art reference as arranged in the claim. A single prior art reference inherently discloses a claim feature if that feature is necessarily present, or inherent, in the reference.

VI. Person of Ordinary Skill in the Art

17. I understand that I must analyze and apply the prior art from the perspective of a person having ordinary skill in the art as of March 24, 1998, which I understand is the patent's earliest possible priority date. When forming my opinions, I analyzed and applied the prior art from the perspective of a skilled artisan as of March 24, 1998.

18. It is my opinion that in March 24, 1998, a person of ordinary skill in the art in digital encoding would have been a person with a computer science degree, or closely related field, and 2 years of experience in the field of data encoding and/or digital watermarking. I recognize that a person of ordinary skill in the art could have less education and more industry experience, or

vice versa, and still meet the definition of a person of ordinary skill in the art. My opinion is based on my personal knowledge and experience working with persons of ordinary skill in the art in the 1998 timeframe.

19. In March 1998, I had a Ph.D. in computer science and had several years of practical experience both in industry and academia (including M.S. and Ph.D.). As of the year 1998, I was teaching and working with individuals who met the above criteria for persons of ordinary skill in the art. In particular, I have taught and worked with distinct groups of graduate students, and even back in 1998 I had advised a number of MS students on various projects. One group entered the graduate program with B.S. degrees in CS/CE/EE and several years of industry training. Finally, I have worked with and taught advanced Ph.D. students that had at least two years of post-BS experience and knowledge gained while in the graduate program. During my time in industry, many of my colleagues possessed at least a B.S. in the relevant fields and had several years of work experience.

20. These students and colleagues all possessed basic knowledge regarding the design and development of digital encoding and/or watermarking technologies. Further, many of these students ultimately found employment at companies that had an expressed interest in and need for skills relating to these technologies, further corroborating that these were ordinarily skilled artisans.

21. Thus, I am familiar with the understanding and knowledge of persons of ordinary skill in the art as of March 24, 1998, and was at least as qualified as the POSITA that I have identified above. Thus, I understand the perspective of a POSITA, which I have applied in my analysis. My opinions would be the same, however, even if the level of ordinary skill varied by some time or varied somewhat with respect to subject matter.

VII. Claim Construction

A. “encoding algorithm” (claims 12-14)

22. The term “encoding algorithm” should be given its broadest reasonable interpretation consistent with the specification of “a process or set of instructions for encoding data.” The ’842 specification includes several examples of encoding algorithms illustrating that these functions are processes or sets of instructions for encoding data to generate license keys. In one instance, the specification states that “any authenticating function can be combined, such as Digital Signature Standard (DSS) or Secure Hash Algorithm (SHA)” to generate an encoded key.¹ In another, the specification states:

A block cipher, such as a Data Encryption Standard (DES) algorithm, in combination with a sufficiently random seed value, such as one created using a Message Digest 5 (MD5) algorithm, emulates a cryptographically secure random bit generator.²

A POSITA would have interpreted these examples as processes or sets of instructions for encoding data.

23. This is also consistent with how a POSITA would have understood an “encoding algorithm.” An algorithm, whether for encoding or some other function, is a process or set of instructions for performing a task. An encoding algorithm is thus a process or set of instruction for encoding data.

24. Therefore, the term “encoding algorithm” should be interpreted as “a process or set of instruction for encoding data.”

¹ ’842 Patent at 8:5-9, 21-23.

² ’842 Patent at 8:12-16.

B. “code resource” (claims 12-14)

25. Based on my review of the '842 patent and its prosecution history, the meaning of term “code resource” is unclear to a POSITA. Yet, I understand that a requester for an *ex parte* reexamination may not challenge a claim based on indefiniteness of a claim term.

26. The '842 Patent states that sub-objects and a memory scheduler, as well as simply data, are examples of code resources.³ But the '842 Patent provides no objective boundaries on what resources in software code would qualify as “code resources,” which would have left a POSITA uncertain as to the meaning of the term and the scope of the claims.

27. I understand that, in the litigation involving the '842 Patent, Patent Owner proposes that this term should have its “plain and ordinary meaning.” For the purposes of analyzing the term and the prior art, I use Patent Owner’s proposed interpretation for this term.

C. “software code interrelationships” (claims 14)

28. Based on my review of the '842 patent and its prosecution history, the meaning of term “software code interrelationships” is also unclear to a POSITA. As previously explained, I understand that a requester for an *ex parte* reexamination may not challenge a claim based on indefiniteness of a claim term.

29. As an expert with more than 27 years of relevant experience, I have never encountered this term outside of the '842 Patent. I therefore looked to the '842 Patent for guidance on the meaning of the term.

30. The term “software code interrelationship” does not appear in the specification nor is there any meaningful discussion regarding interrelationships between code resources. I also looked to the '842 Patent’s prosecution history for guidance as to the meaning of this term.

³ '842 Patent at 11:55-65, 15:36-42.

During the prosecution, Patent Owner stated an “interrelationship” is “the way in which two or more things affect each other because they are related in some way.”⁴ But this statement provides little guidance to a POSITA as to the objective boundaries as what constitutes “software code interrelationships.” This would have left a POSITA uncertain as to the meaning of the term and the scope of the claims.

31. I understand that, in the litigation involving the ’842 Patent, Patent Owner proposes that this term should have its “plain and ordinary meaning.” For the purposes of analyzing the term and the prior art, I use Patent Owner’s proposed interpretation for this term.

VIII. Claims 11, 12, 13, and 14 are Anticipated in View of the Prior Art.

A. Claims 11, 12, 13, and 14 are Anticipated by Beetcher.

1. Beetcher Anticipates Independent Claim 11.

a) Claim 11’s Preamble

32. The preamble of claim 11 reads: “A method for licensed software use, the method comprising.”

33. I understand that a claim’s preamble generally does not limit the scope of the claim under the broadest reasonable interpretation applied during reexamination. Still, Beetcher discloses claim 11’s preamble.

34. Specifically, Beetcher teaches a method of controlling access to licensed software using an encrypted entitlement key.⁵ Beetcher summarizes its invention as:

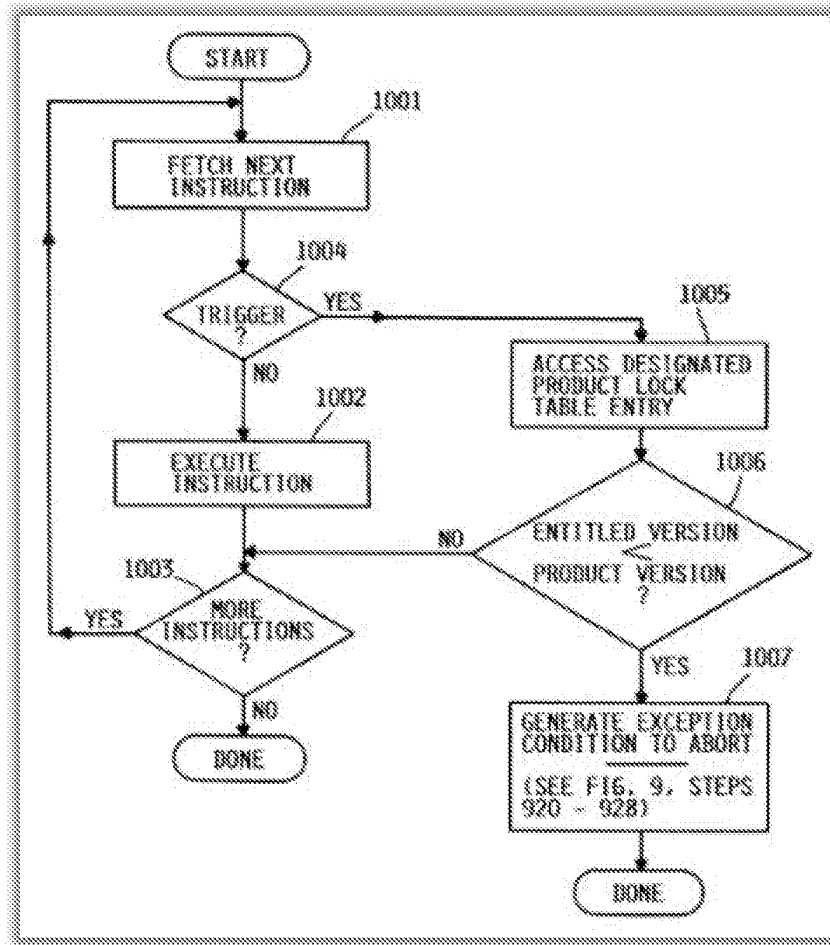
Software is distributed according to the present invention without entitlement to run. A separately distributed encrypted entitlement key enables execution of the Software. The key includes the serial number of the machine for which the Software

⁴ ’842 Prosecution History at 518.

⁵ Beetcher at Abstract, 4:3-13, 4:39-44, 10:48-11:3; *see also* Beetcher at 1:7-11, 1:54-57, 3:54-62.

is licensed, together with a plurality of entitlement bits indicating which Software modules are entitled to run on the machine.⁶

35. Beetcher's Figure 10, as provided below, depicts the use of an entitled version of software based on the customer's license:



36. As I detail below, Beetcher teaches the remaining steps that comprise the method.

⁶ Beetcher at 4:3-9.

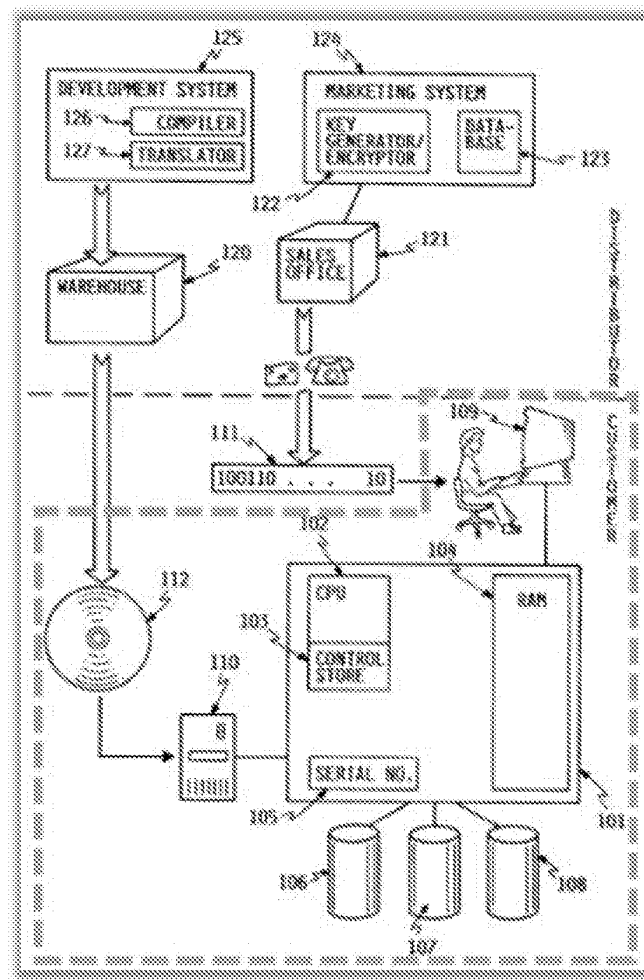
b) *Element 11.1*

37. The first element of claim 11 reads: “loading a software product on a computer, said computer comprising a processor, memory, an input, and an output, so that said computer is programmed to execute said software product.” I refer to this as Element 11.1 throughout this declaration.

38. Beetcher discloses element 11.1. Specifically, Beetcher’s system includes a customer computer 101 including a CPU 102, memory 104, and storage devices 106-108.⁷ This customer computer 101 also includes a media reader 110 (i.e., an input) and an operator console 109 (i.e., an output).⁸ As illustrated in annotated Figure 1, Beetcher discloses a computer having software product 112 loaded for execution (dashed perimeter):

⁷ Beetcher at 5:14-21, Fig. 1.

⁸ Beetcher at 5:25-32, 6:7-15, Fig. 1.



39. Beetcher explains that the customer loads the media, such as an optical disk, containing a software product onto the computer to execute the software product:

[S]oftware media 112 comprise one or more optical read/only disks, and unit 110 is an optical disk reader, it being understood that electronic distribution or other distribution media could be used. Upon receipt of software media 112, the customer will typically load the desired software modules from unit 110 into system 101, and store the software modules on storage devices 106-108.⁹

40. Thus, each limitation of element 11.1 is disclosed by Beetcher.

⁹ Beetcher at 6:7-15; *see also* Beetcher at Abstract, 3:48-50, 9:51-55, Fig. 1, claim 6.

c) *Element 11.2*

41. The second element of claim 11 reads: “said software product outputting a prompt for input of license information.” I refer to this as Element 11.2 throughout this declaration for convenience.

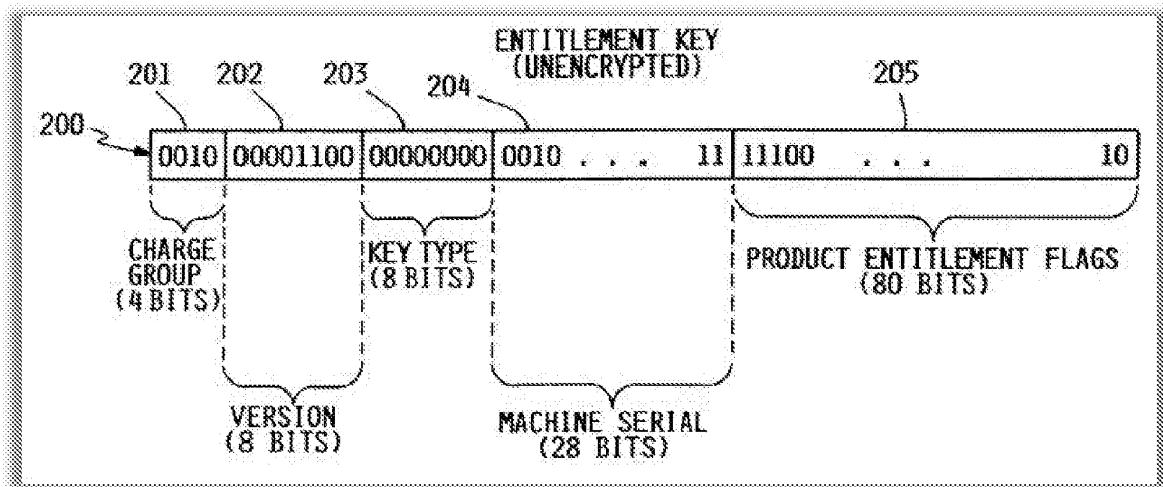
42. Beetcher discloses element 11.2. Beetcher explains that its software product includes a user interface routine for the customer to input a license key into the computer before the product can be used.¹⁰ As an example, Beetcher explains that the software product prompts the user to input license information:

This operation system support at **virtual machine level 404 contains two user interface routines needed to support input of the entitlement key**. General input routine 441 is used to handle input during normal operations. In addition, **special install input routine 440 is required to input the key during initial installation of the operating system**. This is required because that part of the operating system above machine interface level 405 is treated for purposes of this invention as any other program product; it will have a product number and its object code will be infected with entitlement verification triggers.¹¹

Beetcher illustrates an unencrypted version of this license information in Figure 2, provided below:

¹⁰ Beetcher at 7:66-8:8; *see also* Beetcher at 3:25-28.

¹¹ Beetcher at 7:66-8:8.



43. Beetcher goes on to explain that the software's "install input routine 440 interacts with the operator to receive the input" of the customer's license information during the software's initial installation.¹² And as I explain with respect to element 11.1, the customer's computer includes an operator console 109 shown with a monitor and keyboard that "can receive input from an operator."¹³

44. Thus, each limitation of element 11.2 is disclosed by Beetcher.

d) Element 11.3

45. The third element of claim 11 reads: "said software product using license information entered via said input in response to said prompt in a routine designed to decode a first license code encoded in said software product." I refer to this as Element 11.3 throughout this declaration for convenience.

¹² Beetcher at 9:51-55; *see also* Beetcher at Fig. 4 (reference number 440), claim 6.

¹³ Beetcher at 3:25-28, Fig. 1.

46. Beetcher discloses element 11.3. Beetcher explains that, after inserting the software's disk 112, the operator console prompts the customer to enter a license key.¹⁴ Beetcher teaches that the customer enters entitlement key 111, i.e., license information, in response to the prompt initiated by install input routine 440.¹⁵ After entering that key, Beetcher discloses that the customer's computer uses a decode key to initiate unlock routine 430 to decode the license code encoded in the software product.¹⁶ Beetcher's Figures 4 and 9a, provided below, show the software using the key (i.e., license information) entered by the customer to decode a first license code encoded in the software product. For instance, annotated Figure 4 shows that the install input routine 440 starts unlock routine 430 once the customer inputs key 111 into the computer.¹⁷ And "[u]nlock routine 430 uses the unique machine key to decode[] entitlement key 111" (dashed perimeter):¹⁸

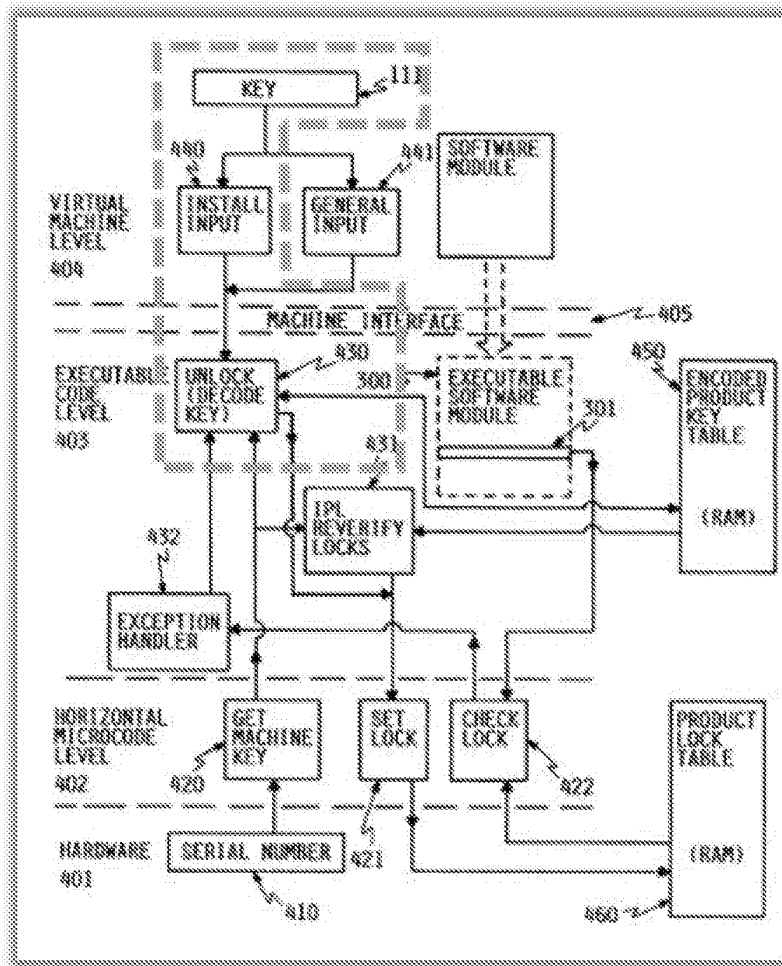
¹⁴ Beetcher at 6:11-19, 7:66-8:8, Figs. 1, 9a.

¹⁵ Beetcher at 7:66-8:8; *see also* Beetcher at 9:51-55, Figs. 1, 4, claim 6.

¹⁶ Beetcher at 7:39-42, 9:49-60; *see also* Beetcher at 6:66-7:5, 8:60-62 Figs. 4, 9a.

¹⁷ Beetcher at 8:3-13, 9:52-60.

¹⁸ Beetcher at 7:39-42; *see also* Beetcher at 8:62-62; 10:27-36.

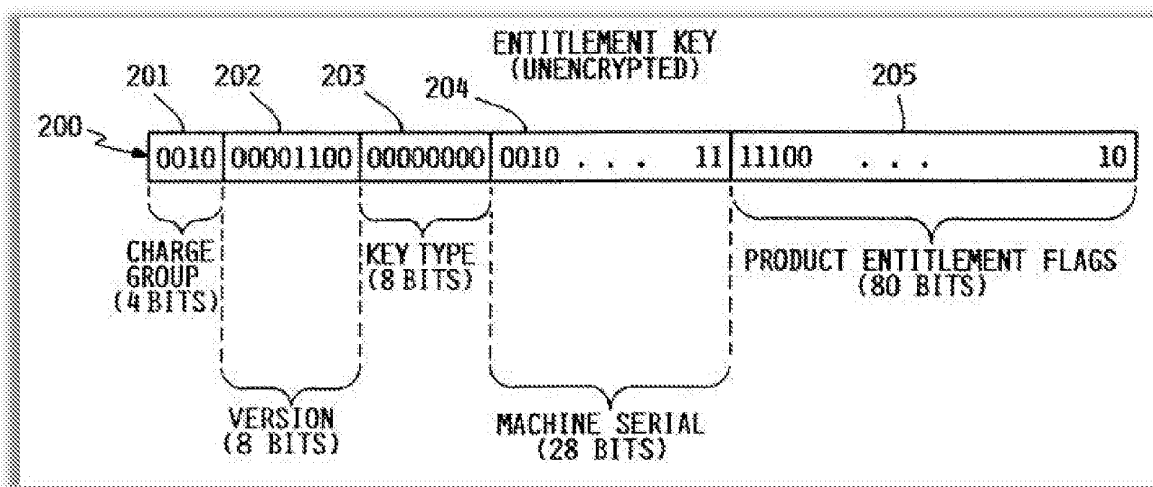


47. Beetcher details that unlock routine 430 “handles the decoding process,” illustrated in Figure 9a’s steps 902-909: “Unlock routine 430 causes get machine key function 420 to retrieve the machine serial number and generate the machine key at 902. Unlock routine 430 then uses the machine key to decode the entitlement key 111 at step 903.”¹⁹

48. The unencrypted entitlement key includes, among other things, version field 202 specifying the user’s entitled version level as well as product entitlement flags field 205

¹⁹ Beetcher at 9:57-60.

specifying which product number to which the user is entitled.²⁰ Beetcher illustrates an unencrypted version of this license information in Figure 2, provided below:



49. Beetcher's unlock routine 430 will complete the decoding process by building an encoded product key table (step 904), populating the key table for the relevant software product specified in the entitlement key (steps 905-908), and saving the key table (step 909).²¹ And Beetcher's RAM includes table 460 reflecting which products the user has entitlement keys.²² As I detail below, the license information decodes a license code in the software product using the key's version and product number fields.

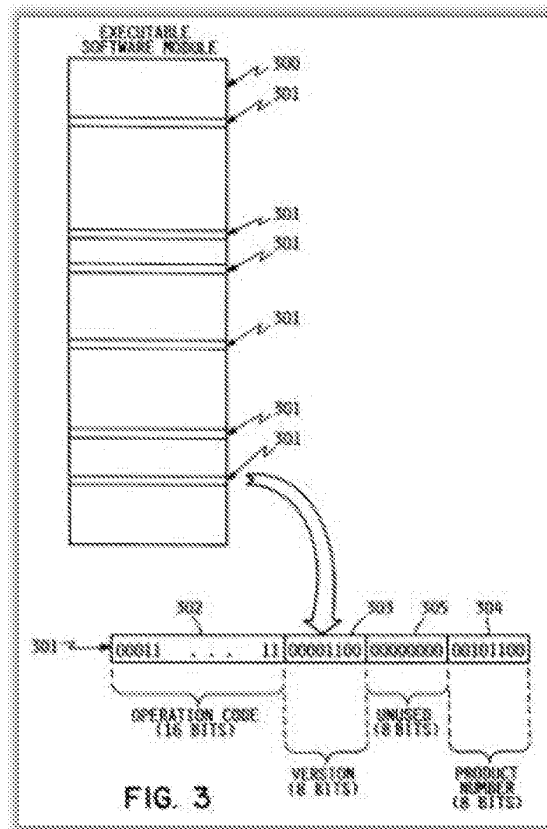
50. When generating its software code, Beetcher explains that the code includes a series of entitlement verification triggering instructions.²³ These triggering instructions are encoded into the software code when being compiled and translated, as shown in Figure 3 below:

²⁰ Beetcher at 6:22-40.

²¹ Beetcher at 9:60-10:19, Figs. 5, 9a.

²² Beetcher at 7:42-44, 8:43-52, 10:20-47, Fig. 6, Fig. 9a.

²³ Beetcher at 6:41-58, 11:4-39; *see also* Beetcher at 4:14-23, 8:5-22, 8:56-9:20.



51. Whenever Beetcher's software code encounters one of the triggering instructions, the code verifies that the customer is entitled to use the software. It does so by accessing the license key information stored in the key table 460.²⁴ For instance, Beetcher details that the customer's computer will access routines, such as check lock function 422, to interpret the license code information contained in one of the triggering instructions:

If any instruction is an entitlement verification triggering instruction 301 (step 1004) check lock function 422 is invoked. Check lock function 422 accesses the product lock table entry 601 corresponding to the product number contained in the triggering instruction at step 1005. If the version number in product lock table 460

²⁴ Beetcher at 10:48-11:39, see also Beetcher at Abstract, 8:14-22, 8:53-9:20, Fig. 10.

is equal to or greater than the version number 303 contained in triggering instruction 301, the software is entitled to execute (step 1006).²⁵

Thus, a POSITA would have understood that Beetcher teaches using license information in a routine designed to decode a first license code encoded in a software product.

52. Moreover, Beetcher explains that the triggering instructions are encoded into the code resources to control software functionality:

[An] additional barrier would be to define the entitlement triggering instruction to simultaneously perform some other function.... The alternative function must be so selected that any compiled software module will be reasonably certain of containing a number of instructions performing the function. If these criteria are met, the compiler can automatically generate the object code to perform the alternative function (and simultaneously, the entitlement verification trigger) as part of its normal compilation procedure. This definition would provide a significant barrier to patching of the object code to nullify the entitlement triggering instructions.²⁶

Beetcher further teaches that “the triggering instruction is also a direct instruction to perform some other useful work [E]xecution of the triggering instruction causes system 101 to perform some other operation simultaneous with the entitlement verification.”²⁷

53. Therefore, each limitation of element 11.3 is disclosed by Beetcher. And as I explain above, Beetcher discloses all the other elements of claim 11. Thus, in my opinion, claim 11 is anticipated by Beetcher.

2. Beetcher Anticipates Independent Claim 12.

a) Claim 12's Preamble

54. The preamble of claim 12 reads: “A method for encoding software code using a computer having a processor and memory, the method comprising.”

²⁵ Beetcher at 10:52-62, Fig. 10.

²⁶ Beetcher at 11:14-28; *see also* Beetcher at 4:25-33, 6:58-65.

²⁷ Beetcher at 6:58-65 (Beetcher specifies that these functions are those “which do not require that an operand for the action be specified in the instruction.”).

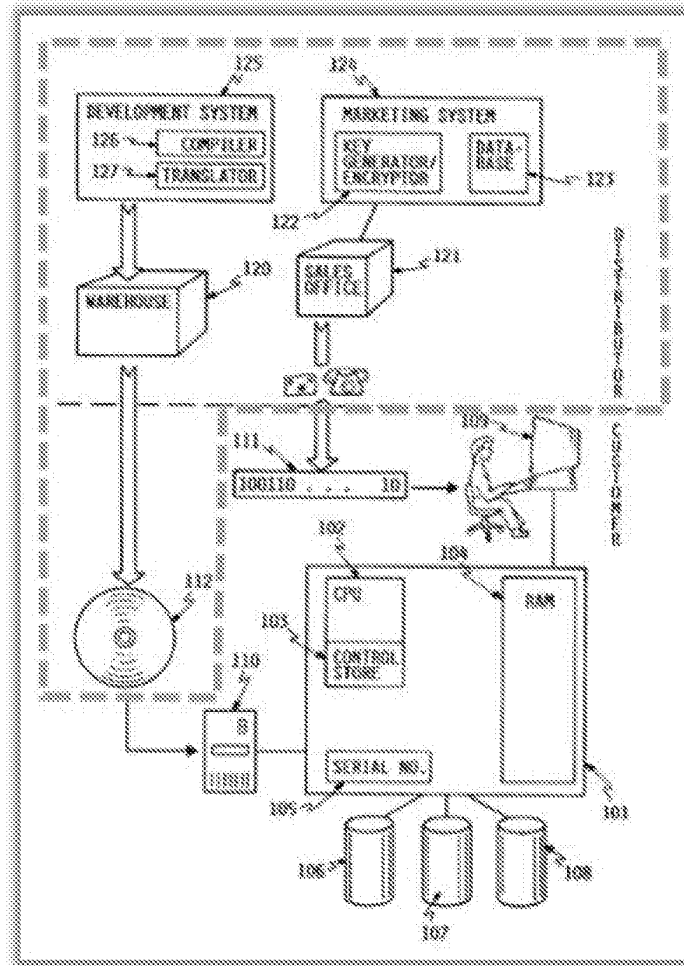
55. I understand that a claim's preamble generally does not limit the scope of the claim under the broadest reasonable interpretation applied during reexamination. Nevertheless, Beetcher discloses claim 12's preamble.

56. Claim 12 recites both a "computer" and a "computer system." It is unclear whether those elements refer to the same computing device or separate computing devices. When analyzing claim 12 using the broadest reasonable interpretation, I interpret the "computer" recited in the preamble to be a device separate from the term "computer system."

57. Beetcher discloses a method for encoding software code using a computer with a processor and memory. Beetcher explains that the software distributor has "development computer system 125, which contains compiler 126 and translator 127" where "[t]he software modules are recorded on software recording media 112" and "entitlement key generator/encrypter 122 and a database 123 containing customer information."²⁸ Beetcher specifies these compiling and key generating functions may be performed by a single computer.²⁹ Annotated Figure 1, below, illustrates the distributor's computer system distributing memory media 112 and compiling encoded software code:

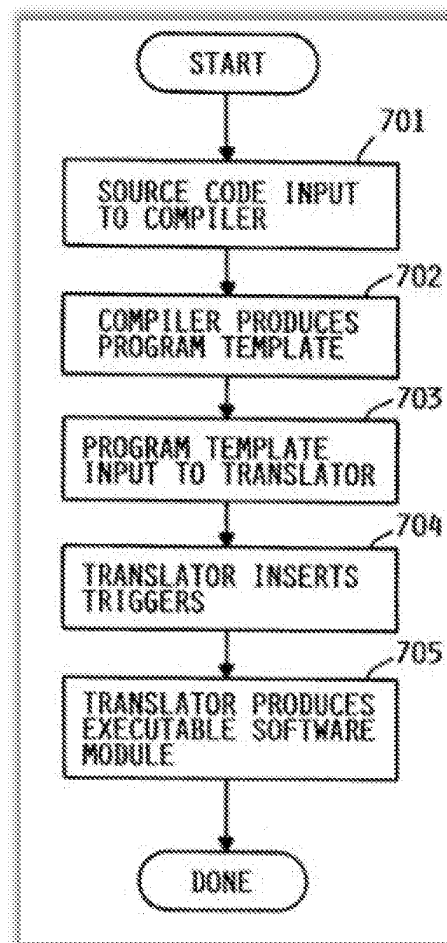
²⁸ Beetcher at 5:38-48; *see also* Beetcher at 9:1-20.

²⁹ Beetcher at 5:51-58.



58. Beetcher's Figure 7 shows the software code being encoded to include watermarking triggers decoded by the customer's licensing information.³⁰

³⁰ Beetcher at 9:1-20, Fig. 7.



59. Thus, a POSITA would have understood that Beetcher's distributor compiles and stores the encode software code using a processor and memory akin to the console's CPU 102 and memory devices 1106-108. Indeed, for as long as computers have been around, it has been standard practice to store the computer code that executes programs—such as the software code used for Beetcher's invention—in memory. In fact, a POSITA would have had no option but to store Beetcher's software code in memory, as this is required in computer programming. Similarly, it has been standard practice to execute such programs using a processor in the computer.

60. As I detail below, Beetcher teaches the remaining steps that comprise the method.

b) Element 12.1

61. The first element of claim 12 reads: “storing a software code in said memory.” I refer to this as Element 12.1 throughout this declaration.

62. Beetcher discloses element 12.1. Specifically, Beetcher describes a development system 125 for compiling and translating for the software code.³¹ Beetcher states that the software code is stored as disks 112 in warehouse 120. A POSITA would have understood that developer system 125 stores the compiled and translated code in memory and records that code onto disks 112 for distribution to customers. And as I discuss regarding claim 12’s preamble, it has been standard practice to store computer code—such as Beetcher’s software code—in memory. In fact, a POSITA would have had no option but to store this software code in memory, as this is required in computer programming.

63. Thus, each limitation of element 12.1 is disclosed by Beetcher.

c) Element 12.2

64. The second element of claim 12 reads: “wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system.” I refer to this as Element 12.2 throughout this declaration.

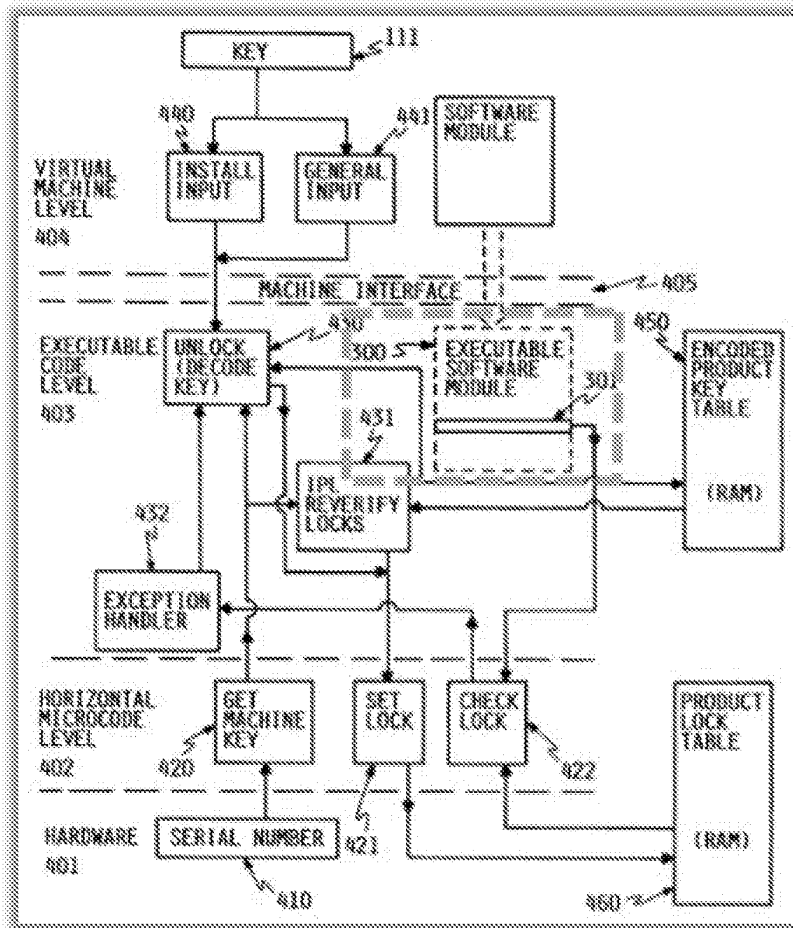
65. Beetcher discloses element 12.2. Specifically, Beetcher teaches that its software code has multiple code resources that include a first code resource.³² Beetcher’s code resources include software modules 300 (dashed box) including sub-objects within the code, as shown below in annotated Figure 4 and Figure 3.³³ These sub-objects control multiple functions of the software

³¹ Beetcher at 5:38-48, 9:1-20.

³² Beetcher at 5:40-43, 6:1-15.

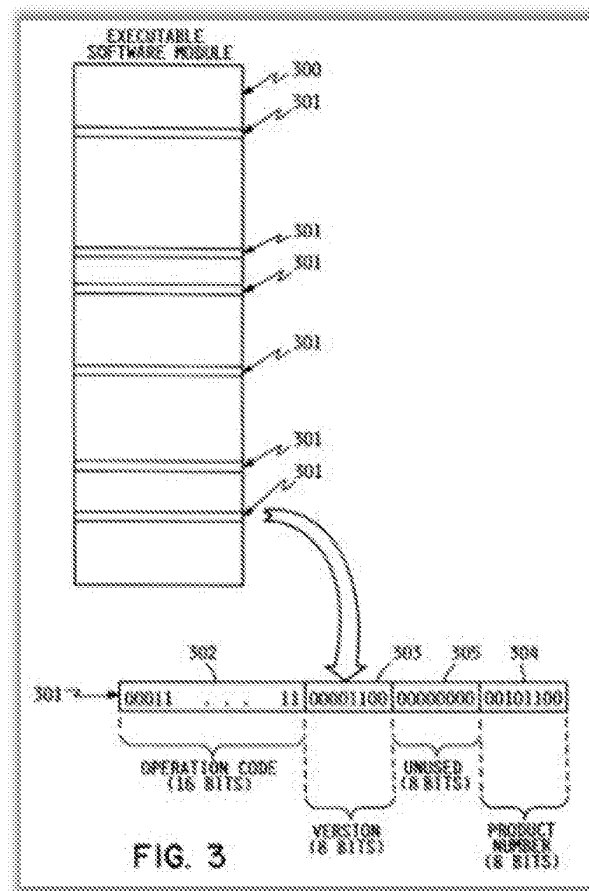
³³ Beetcher at 6:41-45, 8:14-17, Fig. 4; *see also* Beetcher at 7:45-48, Fig. 3.

installed on the customer's computer system 101.³⁴ And Beetcher's software prevents unwanted "patching" of these sub-objects by including entitlement verification triggering instructions 301.³⁵



³⁴ Beetcher at 6:58-65, 11:4-39; *see also* Beetcher at Abstract, 4:28-33, 6:65-7:5, claim 3.

³⁵ Beetcher at 4:25-33, 11:11-39; *see also* Beetcher at Abstract, 3:14-18.



66. The '842 Patent refers to sub-objects and a memory scheduler as examples of code resources.³⁶ A POSITA would have understood that Beetcher's module sub-objects are sub-objects.

67. Relying on Beetcher's description, a POSITA would have understood that one sub-object in module 300) is a first code resource providing a specified underlying functionality when installed on the customer's computer system 101 and unlocked using the license information (key).

68. Thus, each limitation of element 12.2 is disclosed by Beetcher.

³⁶ '842 Patent at 11:55-65, 15:36-42.

d) Element 12.3

69. The third element of claim 12 reads: “encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key encoded software code.” I refer to this as Element 12.3 throughout this declaration.

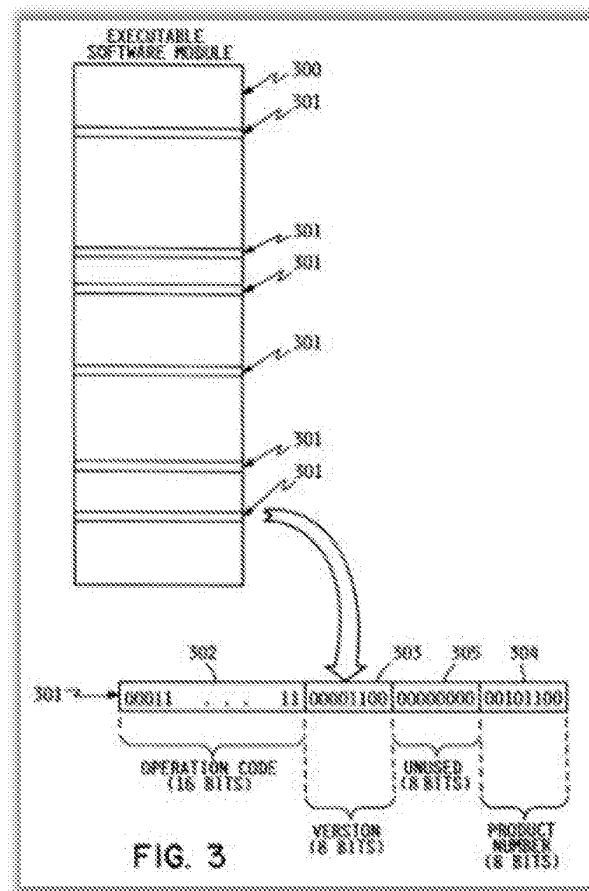
70. Beetcher discloses element 12.3. Beetcher details encoding its software code by the distributor system which includes development system 125 and marketing system 124, which may be “a single computer system performing both functions.”³⁷ As demonstrated, Beetcher describes encoding a first license key into the software code where that key is used to authorize access to the software product:

Software module 300 is part of a program product in compiled object code form which executes on system 101.... [T]he actual executable code operates at executable code level 403, as shown by the box in broken lines. The executable code contains entitlement verification triggering instructions 301 (only one shown), which are executed by horizontal microcode check lock function 422.³⁸

71. The encoding referenced is illustrated in Figure 3:

³⁷ Beetcher at 5:37-58, 6:41-65, 11:4-39.

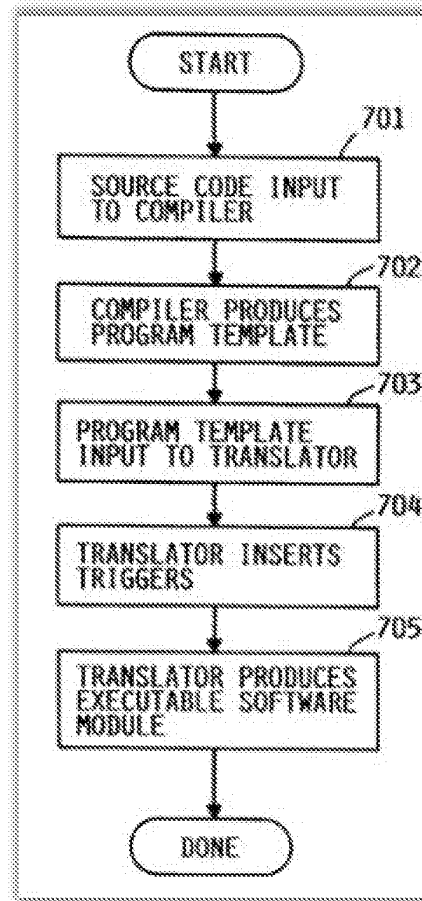
³⁸ Beetcher at 8:13-23; *see also* Beetcher at 4:3-21, 6:20-55, 7:39-44, 8:58-67, 9:51-56, 10:22-38.



72. The computer in Beetcher's development system 125 performs the encoding, as depicted in Figure 7 at step 704, and described as: "The program template serves as input to translator 127 at step 704, along with its product number and version number identification. Translator 127 automatically generates a substantial number of entitlement verification triggers, inserts them in random locations in the object code"³⁹

³⁹ Beetcher at 9:10-16; see also Beetcher at 5:38-47, 9:1-10, 9:16-20, Fig. 7.

73. Furthermore, the computer in Beetcher's development system 125 uses an encoding algorithm to encode the first license key. Beetcher's system uses a set of instruction, as illustrated in Figure 7, to encode triggers into the software code to form the first license key.⁴⁰



74. The compiler starts the process by producing a template (step 702), next the template is input into the translator (step 703), then the translator encodes the triggers/license keys into the code (step 704), and finally the translator resolves references after key insertion to produce the executable module.⁴¹ The generation of “a substantial number of entitlement triggers” and

⁴⁰ Beetcher at 9:10-16, *see also* Beetcher at 5:38-47, 9:1-10, 9:16-20, Fig. 7.

⁴¹ Beetcher at 9:6-20, Fig. 7.

“insert[ing] them in random locations in the object code” that would require “an encrypted entitlement key” would require an encoding algorithm.⁴² Thus, a POSITA would have understood Beetcher’s Figure 7 illustrates an encoding algorithm. Beetcher’s encoding process is additionally described with respect to element 11.3.

75. Moreover, during the original prosecution, Patent Owner stated that “[e]ncoding using a key and an algorithm is known.”⁴³ Thus, a POSITA would have understood that Beetcher’s encoding technique necessarily includes a first license key and an encoding algorithm to form a first license key encoded software code.

76. Thus, each limitation of element 12.3 is disclosed by Beetcher.

e) Element 12.4

77. The fourth element of claim 12 reads: “wherein, when installed on a computer system, said first license key encoded software code will provide said specified underlying functionality only after receipt of said first license key.” I refer to this as Element 12.4 throughout this declaration.

78. Beetcher teaches element 12.4. Specifically, Beetcher discloses that its first license key encoded software code provides the specified underlying functionality only after receipt of the first license key.⁴⁴ For example, Beetcher states:

For support of such a traditional compilation path where the object code format is known by customers, additional barriers to patching of the object code to nullify or alter the entitlement triggering instructions may be appropriate. One such additional barrier would be to define the entitlement triggering instruction to simultaneously perform some other function. In this case, it is critical that the alternative function performed by the triggering instruction can not be performed by any other simple

⁴² Beetcher at 9:12-48.

⁴³ ’842 Prosecution History at 519.

⁴⁴ Beetcher at 6:58-65, 11:4-39; *see also* Beetcher at Abstract, 3:14-18, 4:25-33, 6:65-7:5, claim 3.

instruction. The alternative function must be so selected that any compiled software module will be reasonably certain of containing a number of instructions performing the function. If these criteria are met, the compiler can automatically generate the object code to perform the alternative function (and simultaneously, the entitlement verification trigger) as part of its normal compilation procedure. This definition would provide a significant barrier to patching of the object code to nullify the entitlement triggering instructions.⁴⁵

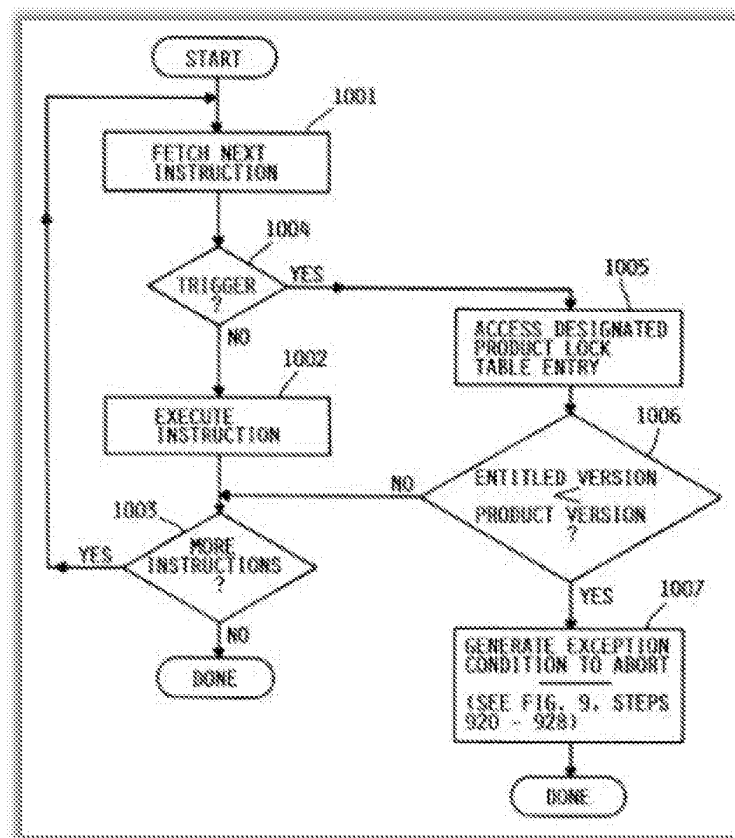
79. And as described with respect to element 12.3, Beetcher teaches encoding the triggering instructions into the software code that is decoded via the first license key.

80. Beetcher's Figure 10, as reproduced below, illustrates providing the software's underlying functionality based on the first license key (triggering information). For instance, Beetcher explains:

81. System 101 executes the module by fetching (step 1001) and executing (step 1002) object code instructions until done (step 1003). If any instruction is an entitlement verification triggering instruction 301 (step 1004) check lock function 422 is invoked. Check lock function 422 accesses the product lock table entry 601 corresponding to the product number contained in the triggering instruction at step 1005. If the version number in product lock table 460 is equal to or greater than the version number 303 contained in triggering instruction 301, the software is entitled to execute (step 1006).⁴⁶

⁴⁵ Beetcher at 11:10-28.

⁴⁶ Beetcher at 10:49-60; *see also* Beetcher at 10:48-49, 10:60-11:3.



82. Consequently, each limitation of element 12.4 is disclosed by Beetcher. And as I explain above, Beetcher discloses all the other elements of claim 12. Thus, in my opinion, claim 12 is anticipated by Beetcher.

3. Beetcher Anticipates Independent Claim 13.

a) Claim 13's Preamble

83. The preamble of claim 13 reads: "A method for encoding software code using a computer having a processor and memory, comprising"

84. I understand that a claim's preamble generally does not limit the scope of the claim under the broadest reasonable interpretation applied during reexamination. Nevertheless, Beetcher discloses claim 13's preamble.

85. Claim 13's preamble appears to be the same as claim 12's preamble. And as I explain above, Beetcher discloses a method for encoding software using a computer with a processor and memory. Thus, Beetcher teaches this preamble.

b) Element 13.1

86. The first element of claim 13 reads: "storing a software code in said memory." I refer to this as Element 13.1 throughout this declaration.

87. Element 13.1 is identical to element 12.1, which I discuss above. For the same reasons as I explain above, Beetcher discloses each limitation of element 13.1.

c) Element 13.2

88. The second element of claim 13 reads: "wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system." I refer to this as Element 13.2 throughout this declaration.

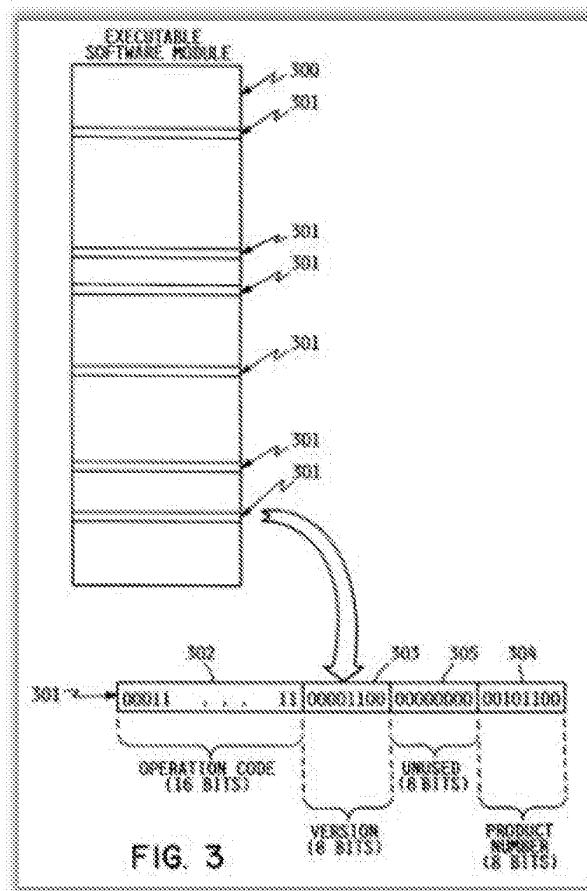
89. Element 13.2 is identical to element 12.2, which I discuss above. For the same reasons as I explain above, Beetcher discloses each limitation of element 13.2.

d) Element 13.3

90. The third element of claim 13 reads: "modifying, by said computer, using a first license key and an encoding algorithm, said software code, to form a modified software code; and wherein said modifying comprises encoding said first code resource to form an encoded first code resource." I refer to this as Element 13.3 throughout this declaration.

91. Beetcher discloses element 13.3. As identified with respect to element 12.3, Beetcher's distributor system includes a computer that encodes software code using a first license key (e.g., triggering information) and an encoding algorithm (e.g., Figure 7). And Beetcher's encoding

process modifies the software code by inserting triggering information into the software code.⁴⁷ For example, Beetcher teaches that compiled software code is input to a translator which modifies the code by “automatically generat[ing] a substantial number of entitlement verification triggers” and “insert[ing] them in random locations in the object code,” as shown in Figure 7’s steps 703 and 704.⁴⁸ Figure 3 illustrates this modifying by inserting triggering information 301 to form a modified software code:



⁴⁷ Beetcher at 8:13-23, 9:1-20; *see also* Beetcher at 5:38-47, 9:1-10, 9:16-20, Fig. 7.

⁴⁸ Beetcher at 9:11-15.

92. As explained with respect to elements 12.2, Beetcher's software code includes a series of code resources corresponding to sub-objects. And Beetcher teaches a given first code resource is modified to encode the first code resource via the triggering information.⁴⁹ For instance,

Beetcher teaches:

For support of such a traditional compilation path where the object code format is known by customers, additional barriers to patching of the object code to nullify or alter the entitlement triggering instructions may be appropriate. One such additional barrier would be to define the entitlement triggering instruction to simultaneously perform some other function. In this case, it is critical that the alternative function performed by the triggering instruction can not be performed by any other simple instruction. The alternative function must be so selected that any compiled software module will be reasonably certain of containing a number of instructions performing the function. If these criteria are met, the compiler can automatically generate the object code to perform the alternative function (and simultaneously, the entitlement verification trigger) as part of its normal compilation procedure. This definition would provide a significant barrier to patching of the object code to nullify the entitlement triggering instructions.⁵⁰

A POSITA would have understood that such modification results in an encoded first code resource.

93. Further, during the original prosecution, Patent Owner specified that "[e]ncoding using a key and an algorithm is known."⁵¹ Therefore, a POSITA would have understood that Beetcher's encoding technique necessarily includes a first license key and an encoding algorithm to form a modified encoded first code resource.

94. Thus, each limitation of element 13.3 is disclosed by Beetcher.

⁴⁹ Beetcher at 4:25-33, 11:11-39; *see also* Beetcher at Abstract, 3:14-18.

⁵⁰ Beetcher at 11:10-28.

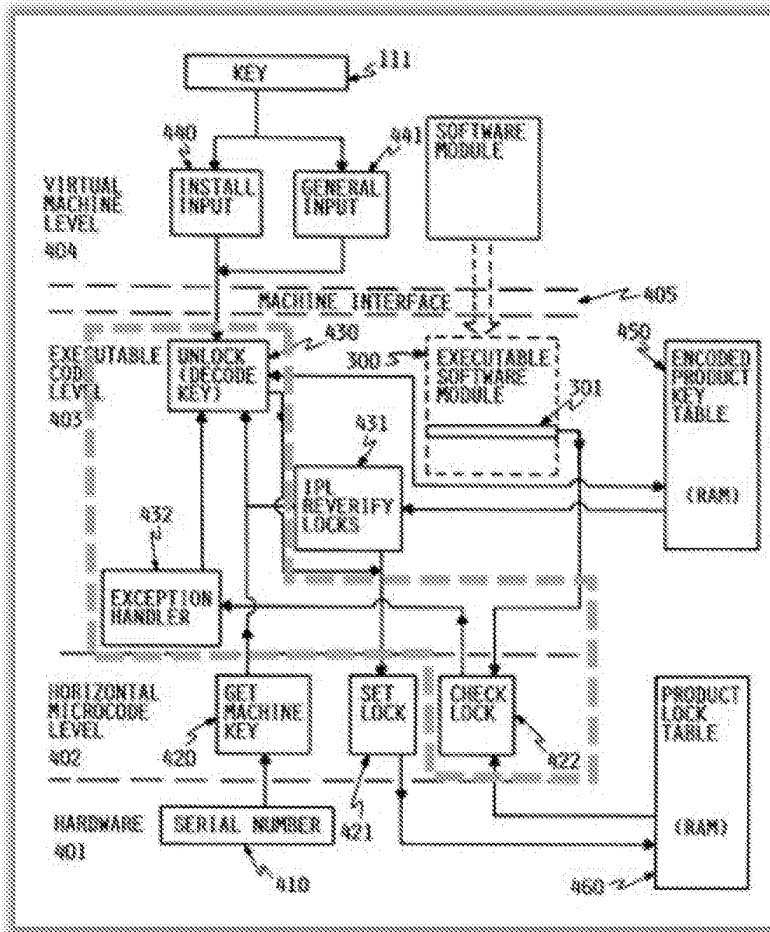
⁵¹ '842 Prosecution History at 519.

e) Element 13.4

95. The fourth element of claim 13 reads: “wherein said modified software code comprises said encoded first code resource, and a decode resource for decoding said encoded first code resource.” I refer to this as Element 13.4 throughout this declaration.

96. Beetcher discloses element 13.4. Beetcher explains that its modified software code includes a decode resource for decoding the encoded first code resource. Beetcher discloses that executing a trigger 301 invokes check lock function 422, which results in accessing “unlock (decode key)” function 430 upon confirmation that the customer possesses the software’s license key.⁵² Beetcher’s Figure 4, as annotated below, illustrates the decode resource (dashed perimeter) of the modified software code:

⁵² Beetcher at 10:22-39, 10:52-65, Figs. 9b, 10; *see also* Beetcher at 7:16-38, 8:18-22, 9:49-10:7.



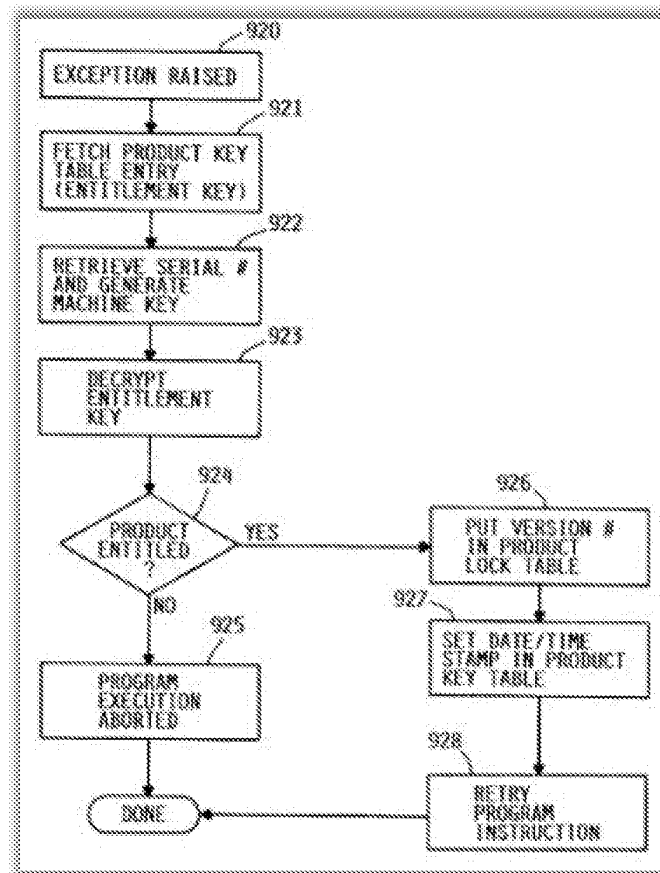
97. Thus, each limitation of element 13.4 is disclosed by Beetcher.

f) Element 13.5

98. The last element of claim 13 reads: “wherein said decode resource is configured to decode said encoded first code resource upon receipt of said first license key.” I refer to this as Element 13.5 throughout this declaration.

99. Beetcher discloses element 13.5. Beetcher states that its decode resource decodes the encoded first code resource upon receipt of the license key. Beetcher, for instance, states that unlock routine 430 “fetches the encrypted entitlement key from ... table 450 ... and decodes the entitlement key The triggering instruction is then retried and program execution continues at

step 928.”⁵³ And Beetcher’s Figure 9b illustrates accessing the decode resource to decode the encoded first code resources based on the entitlement key, reflected in steps 921 to 928:



Thus, a POSITA would have understood that Beetcher’s decode resource is configured to decode the encoded first code resource based on first license key.

100. Therefore, each limitation of element 13.5 is disclosed by Beetcher. And as I explain above, Beetcher discloses all the other elements of claim 13. In my opinion, claim 13 is anticipated by Beetcher.

⁵³ Beetcher at 10:27-38.

4. Beetcher Anticipates Independent Claim 14.

a) *Claim 14's Preamble*

101. The preamble of claim 14 reads: “A method for encoding software code using a computer having a processor and memory, comprising.”

102. I understand that a claim’s preamble generally does not limit the scope of the claim under the broadest reasonable interpretation applied during reexamination. Nevertheless, Beetcher discloses claim 14’s preamble.

103. Claim 14’s preamble appears to be the same as each of claim 12 and 13’s preamble. As I explain above, Beetcher discloses a method for encoding software using a computer with a processor and memory. Thus, Beetcher teaches this preamble.

b) *Element 14.1*

104. The first element of claim 14 reads: “storing a software code in said memory.” I refer to this as Element 14.1 throughout this declaration.

105. Element 14.1 is identical to element 12.1, which I discuss above. For the same reasons as I explain above, Beetcher discloses each limitation of element 14.1.

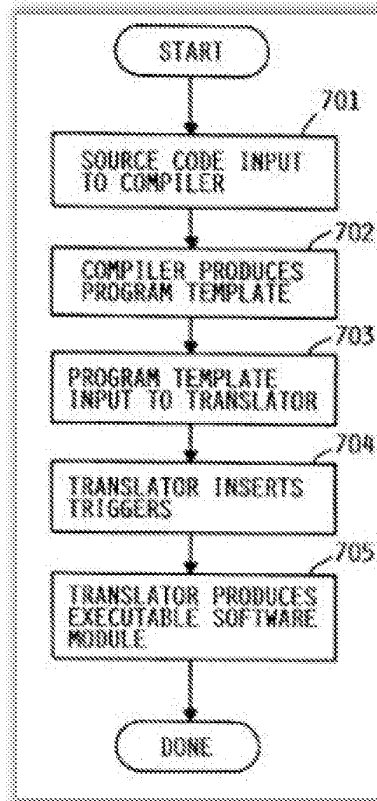
c) *Element 14.2*

106. The second element of claim 14 reads: “wherein said software code defines software code interrelationships between code resources that result in a specified underlying functionality when installed on a computer system.” I refer to this as Element 14.2 throughout this declaration.

107. Beetcher discloses element 14.2. Beetcher describes that its software code is compiled into executable code by compiler 126. This compiler works with translator 127 to compile the software sub-objects and insert triggering information.⁵⁴ And Beetcher specifies that translator

⁵⁴ Beetcher at 8:14-17.

127 “resolves references” in the software code, which corresponds to defining code interrelationships between code resources.⁵⁵ As shown in steps 701 and 702 of Figure 7, Beetcher discloses its software code is input into compiler 126 that produces a template of the software code.⁵⁶



108. A POSITA would have understood that this software code template also defines the code interrelationships between the code resources. As the Patent Owner stated during the original prosecution, software code interrelationships are defined during the compiling process of conventional software applications:

⁵⁵ Beetcher 9:11-18.

⁵⁶ Beetcher 8:14-17, 9:1-20, Fig. 7; *see also* Beetcher at 5:37-39, 6:41-45, 7:63-66

What the examiner has implied by alleging that the "specification ... fails to teach or mention 'software code interrelationships'" is that software code interrelationships were somehow unknown in the art, which clearly is not the case. As admitted, in the specification at the beginning of paragraph [0051], an "application" comprises "sub-objects" whose "order in the computer memory is of vital importance" in order to perform an intended function. And as admitted further in paragraph [0051], **"When a program is compiled, then, it consists of a collection of these sub-objects, whose exact order or arrangement in memory is not important, so long as any sub-object which uses another sub-object knows where in memory it can be found."** Paragraph [0051] of course refers to conventional applications. Accordingly, that is admittedly a discussion of what is already known by one skilled in the art. Accordingly, the examiner's statement that the specification lacks written description support for "software code interrelationships" is inconsistent with the fact that such **interrelationships were explained in paragraphs [0051] and [0052] as a fundamental basis of pre-existing modem computer programs.**⁵⁷

109. Additionally, during the original prosecution, Patent Owner specified that "interrelationships between code resources are not that which is novel."⁵⁸ Based on the Patent Owner's admissions, it is clear that a POSITA would have understood that Beetcher's code necessarily defines code interrelationships between code resources.

110. Beetcher further discloses that the code resource interrelationships specify the underlying application functionalities when installed on the customer's computer 101. For example, Beetcher's software code includes multiple entitlement verification triggers.⁵⁹ And Beetcher details that certain code resources include triggering instruction that controls the underlying functionalities of the software code:

[An] additional barrier would be to define the entitlement triggering instruction to simultaneously perform some other function.... The alternative function must be so selected that any compiled software module will be reasonably certain of containing a number of instructions performing the function. If these criteria are met, the compiler can automatically generate the object code to perform the alternative function (and simultaneously, the entitlement verification trigger) as

⁵⁷ '842 Prosecution History at 519.

⁵⁸ '842 Prosecution History at 519.

⁵⁹ Beetcher at 4:15-33, 9:1-3, 10:22-34, Fig. 3; *see also* Beetcher at 6:45-65, 8:19-22, 10:52-11:39.

part of its normal compilation procedure. This definition would provide a significant barrier to patching of the object code to nullify the entitlement triggering instructions.⁶⁰

111. Beetcher further teaches that “the triggering instruction is also a direct instruction to perform some other useful work [E]xecution of the triggering instruction causes system 101 to perform some other operation simultaneous with the entitlement verification.”⁶¹ Thus, a POSITA would have understood that the code interrelationships between Beetcher’s code resources result in a specified underlying functionality once installed.

112. Thus, each limitation of element 14.2 is disclosed by Beetcher.

d) Element 14.3

113. The third element of claim 14 reads: “encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key encoded software code.” I refer to this as Element 14.3 throughout this declaration.

114. Element 14.3 is identical to element 12.3, which I address above. For the same reasons I explain above, Beetcher discloses each limitation of element 14.3.

115. Also, during the original prosecution, Patent Owner stated that “[e]ncoding using a key and an algorithm is known” and that “an interrelationship in software code is necessarily defined by digital data, and digital data can obviously be encoded by an encoding process.”⁶² Therefore, a POSITA would have understood that Beetcher’s encoding technique necessarily includes a first license key and an encoding algorithm to form a first license key encoded software code.

⁶⁰ Beetcher at 11:14-28; *see also* Beetcher at 4:25-33, 6:58-65.

⁶¹ Beetcher at 6:58-65 (Beetcher specifies that these functions are those “which do not require that an operand for the action be specified in the instruction.”).

⁶² ’842 Prosecution History at 519.

e) *Element 14.4*

116. The fourth element of claim 14 reads: “in which at least one of said software code interrelationships are encoded.” I refer to this as Element 14.4 throughout this declaration.

117. Beetcher discloses element 14.4. As described with respect to element 14.2, Beetcher teaches that its software code defines code interrelationships between code resources and triggering information 301 in the code control certain underlying software functionality. And Beetcher explains that triggering information 301 is encoded into the software code.⁶³ For instance, Beetcher details that the triggering instructions will be encoded into the code resources controlling software functionality:

[An] additional barrier would be to define the entitlement triggering instruction to simultaneously perform some other function.... The alternative function must be so selected that any compiled software module will be reasonably certain of containing a number of instructions performing the function. If these criteria are met, the compiler can automatically generate the object code to perform the alternative function (and simultaneously, the entitlement verification trigger) as part of its normal compilation procedure. This definition would provide a significant barrier to patching of the object code to nullify the entitlement triggering instructions.⁶⁴

118. And Beetcher teaches that “the triggering instruction is also a direct instruction to perform some other useful work [E]xecution of the triggering instruction causes system 101 to perform some other operation simultaneous with the entitlement verification.”⁶⁵ Therefore, a POSITA would have understood that this encoded triggering information includes encoded code interrelationship of the code resources.

⁶³ Beetcher at 4:25-33, 6:58-65, 11:4-39.

⁶⁴ Beetcher at 11:14-28; *see also* Beetcher at 4:25-33, 6:58-65.

⁶⁵ Beetcher at 6:58-65 (Beetcher specifies that these functions are those “which do not require that an operand for the action be specified in the instruction.”).

119. Therefore, each limitation of element 14.4 is disclosed by Beetcher. And as I explain above, Beetcher discloses all the other elements of claim 14. Thus, in my opinion, claim 14 is anticipated by Beetcher.

B. Claims 11, 12, 13, and 14 are Anticipated by Beetcher '072.

120. It is my understanding that Beetcher '072 claims priority to U.S. Application No. 07/629,295, as reflected on the cover of Beetcher '072. It is also my understanding that Beetcher, which I discuss in Section VIII.B, also claims priority to U.S. Application No. 07/629,295. Throughout my discussion of Beetcher '072, I refer to the figures from Beetcher '072 and the English translation of Beetcher '072's specification and claims.

1. Beetcher '072 Anticipates Independent Claim 11.

a) Claim 11's Preamble

121. The preamble of claim 11 reads: "A method for licensed software use, the method comprising."

122. I understand that a claim's preamble generally does not limit the scope of the claim under the broadest reasonable interpretation applied during reexamination. Nevertheless, Beetcher '072 discloses claim 11's preamble.

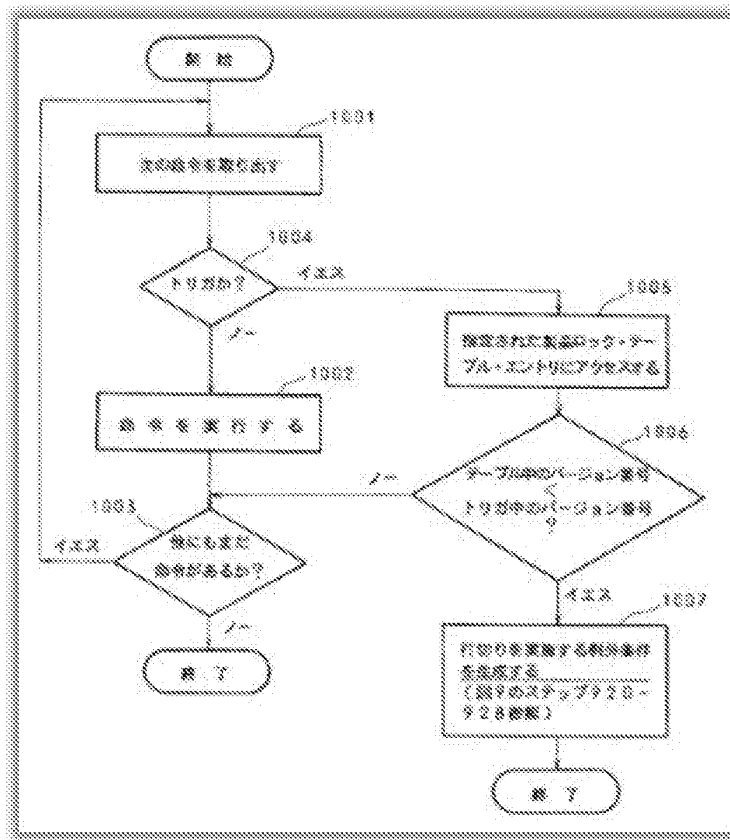
123. Beetcher '072 discloses a method of controlling access to licensed software using an encrypted entitlement key.⁶⁶ Beetcher '072 summarizes its invention as:

According to the present invention, software is distributed without the qualification grant for performing. Execution of software is attained by the enciphered qualification grant key which is distributed independently. This qualification grant key contains a plurality of qualification grant bits which instruct the consecutive numbers of the machine with which software is licensed to it, and which software module has the qualification it runs by that machine.⁶⁷

⁶⁶ Beetcher '072 at Abstract, ¶¶ 0020, 0022, 0043; *see also* Beetcher '072 at ¶¶ 0001, 0004, 0016.

⁶⁷ Beetcher '072 at ¶ 0020.

124. Beetcher '072's Figure 10, reproduced below, depicts the use of an entitled version of software based on the customer's license:

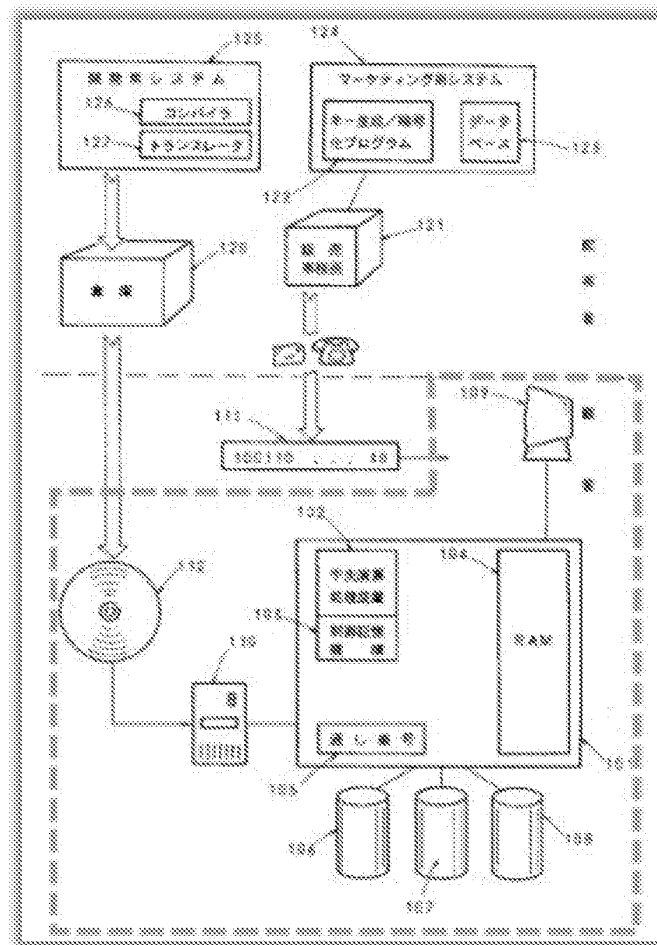


125. As I detail below, Beetcher '072 teaches the remaining steps that comprise the method.

b) *Element 11.1*

126. The first element of claim 11 reads: “loading a software product on a computer, said computer comprising a processor, memory, an input, and an output, so that said computer is programmed to execute said software product.” I refer to this as Element 11.1 throughout this declaration.

127. Beetcher '072 discloses element 11.1. Beetcher '072's system includes a customer computer 101 including a CPU 102, memory 104, and storage devices 106-108.⁶⁸ This customer computer 101 also includes a media reader 110 (i.e., an input) and an operator console 109 (i.e., an output).⁶⁹ As shown below in annotated Figure 1, Beetcher '072 discloses a computer having software product 112 loaded for execution (dashed perimeter):



⁶⁸ Beetcher '072 at ¶ 0023, Fig. 1.

⁶⁹ Beetcher '072 at ¶¶ 0023, 0027, Fig. 1.

128. Beetcher '072 teaches that the customer loads the media, such as an optical disk, containing a software product onto the computer to execute the software product:

[S]oftware media 112 comprise one sheet or a plurality of read-only optical discs, and the medium reader 110 is an optical disc reader. However, please understand that an electronic distribution medium and other distribution media can also be used. If the software media 112 are received, a customer will load a desired software module to the system 101 from the medium reader 110, and will usually memorize the software module to the memory storage 106-108.⁷⁰

129. Thus, each limitation of element 11.1 is disclosed by Beetcher '072.

c) Element 11.2

130. The second element of claim 11 reads: "said software product outputting a prompt for input of license information." I refer to this as Element 11.2 throughout this declaration for convenience.

131. Beetcher '072 discloses element 11.2. Beetcher '072 teaches that its software product contains a user interface routine for the customer to input a license key into the computer before the product can be used.⁷¹ For instance, Beetcher '072 explains that the software product prompts the user to input license information:

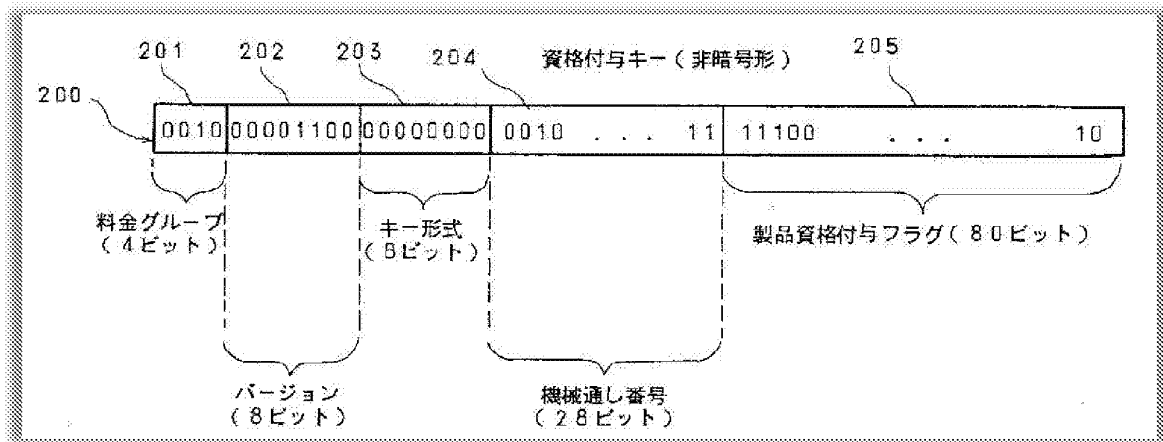
The support of this operation system contains **two user interface routines required to support the input of a qualification grant key on the virtual-machine level 404**. The general input routine 441 is used for processing an input in normal operation. The **installation input routine 440 special to inputting a qualification grant key is required during the initial introduction of an operation system**. The thing which needs this is because the portion of an upper level operating system is treated as other program products by the present invention from the machine interface level 405. Namely, such a portion has product number and the target code is subject to the influence of a qualification verification trigger.⁷²

⁷⁰ Beetcher '072 at ¶ 0027; *see also* Beetcher '072 at Abstract, ¶¶ 0014, 0040, Fig. 1, claim 6.

⁷¹ Beetcher '072 at ¶ 0033; *see also* Beetcher '072 at ¶ 0010.

⁷² Beetcher '072 at ¶ 0033.

Beetcher '072 illustrates an unencrypted version of this license information in Figure 2, provided below:



132. Beetcher '072 further teaches that the software's "installation input routine 440 has a dialog with an operator, and receives an input" of the customer's license information during the software's initial installation.⁷³ And as I discuss with respect to element 11.1, the customer's computer includes an operator console 109 shown with a monitor and keyboard that "receive the input from an operator."⁷⁴

133. Thus, each limitation of element 11.2 is disclosed by Beetcher '072.

d) Element 11.3

134. The third element of claim 11 reads: "said software product using license information entered via said input in response to said prompt in a routine designed to decode a first license code encoded in said software product." I refer to this as Element 11.3 throughout this declaration for convenience.

⁷³ Beetcher '072 at ¶ 0040; *see also* Beetcher '072 at Fig. 4 (reference number 440), claim 6.

⁷⁴ Beetcher '072 at ¶ 0023; *see also* Beetcher '072 at ¶¶ 0025, 0033, 0039, Fig. 1.

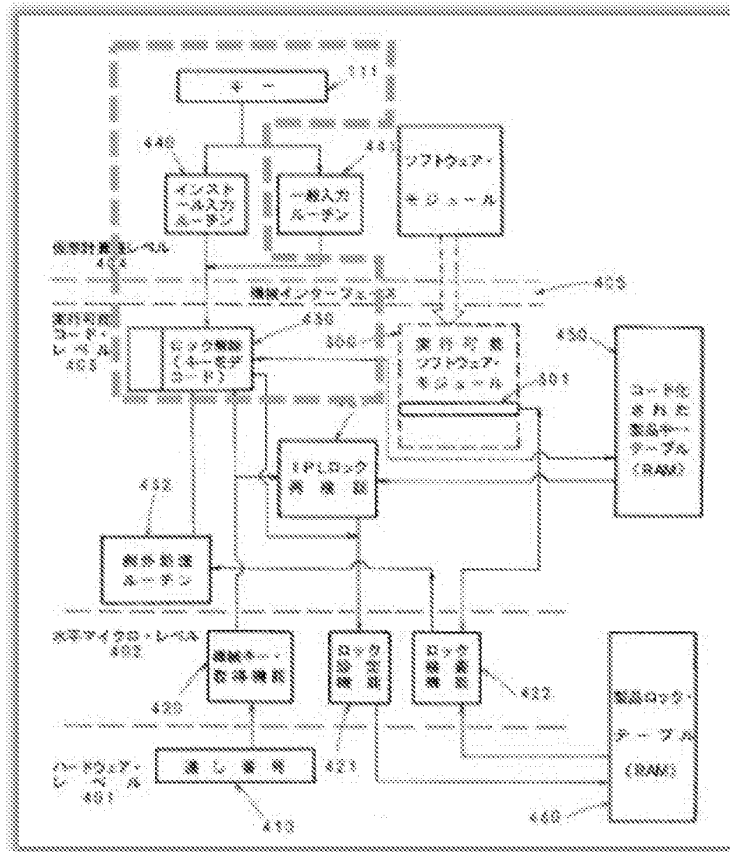
135. Beetcher '072 discloses element 11.3. Beetcher '072 explains that, after inserting the software's disk 112, the operator console prompts the customer to enter license information. Beetcher '072 explains that the customer enters entitlement key 111, i.e., license information, in response to the prompt initiated by install input routine 440.⁷⁵ After entering that key, Beetcher '072 teaches that the customer's computer uses a decode key to initiate unlock routine 430 to decode the license code encoded in the software product.⁷⁶ Beetcher '072's Figures 4 and 9a, which are provided below, show the software using the key (i.e., license information) entered by the customer to decode a first license code encoded in the software product. For example, annotated Figure 4 depicts that the install input routine 440 starts unlock routine 430 once the customer inputs key 111 into the computer.⁷⁷ And "unlocking routine 430 decodes the qualification grant key 111 using a peculiar machine key" (dashed perimeter):⁷⁸

⁷⁵ Beetcher '072 at ¶ 0033; *see also* Beetcher '072 at ¶ 0040, Figs. 1, 4, claim 6.

⁷⁶ Beetcher '072 at ¶¶ 0032, 0040; *see also* Beetcher '072 at ¶¶ 0030, 0037, Figs. 4, 9a.

⁷⁷ Beetcher '072 at ¶¶ 0033, 0040.

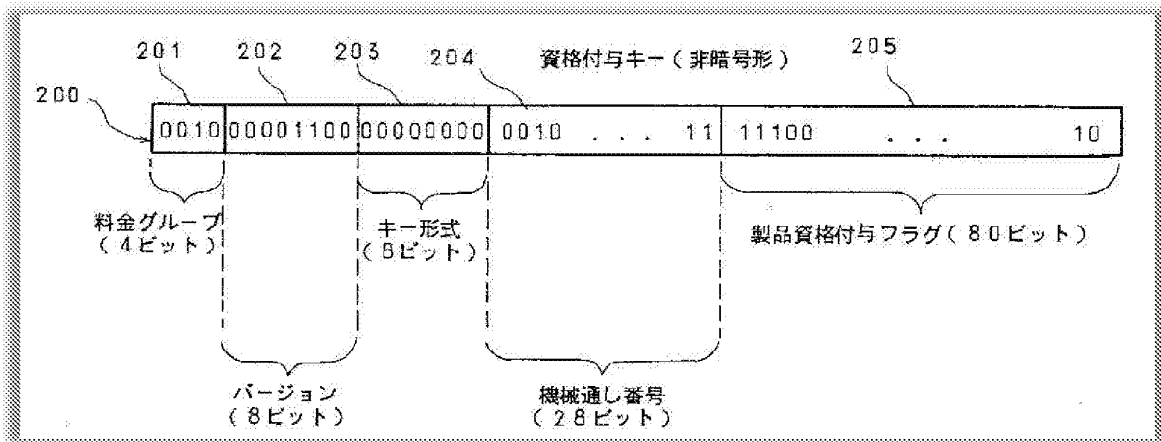
⁷⁸ Beetcher '072 at ¶ 0032; *see also* Beetcher '072 at ¶¶ 0037; 0041.



136. Beetcher '072 teaches that unlock routine 430 “handles the decoding process,” which is illustrated in Figure 9a’s steps 902-909: “The lock release routine 430 makes the machine-key acquisition function 420 search machine consecutive numbers with Step 902, and makes it generate a machine key at it. Subsequently, the lock release routine 430 decodes the qualification grant key 111 at Step 903 using a machine key.”⁷⁹ The unencrypted entitlement key includes, among other things, version field 202 specifying the user’s entitled version level as well as product entitlement flags field 205 specifying which product number to which the user is

⁷⁹ Beetcher '072 at ¶ 0040.

entitled.⁸⁰ Beetcher '072 illustrates an unencrypted version of this license information in Figure 2, provided below:



137. Beetcher '072's unlock routine 430 will complete the decoding process by building an encoded product key table (step 904), populating the key table for the relevant software product (steps 905-908), and saving the key table (step 909).⁸¹ And Beetcher '072's RAM includes table 460 reflecting which products the user has entitlement keys.⁸² As I detail below, the license information decodes a license code in the software product using the key's version field and product number fields.

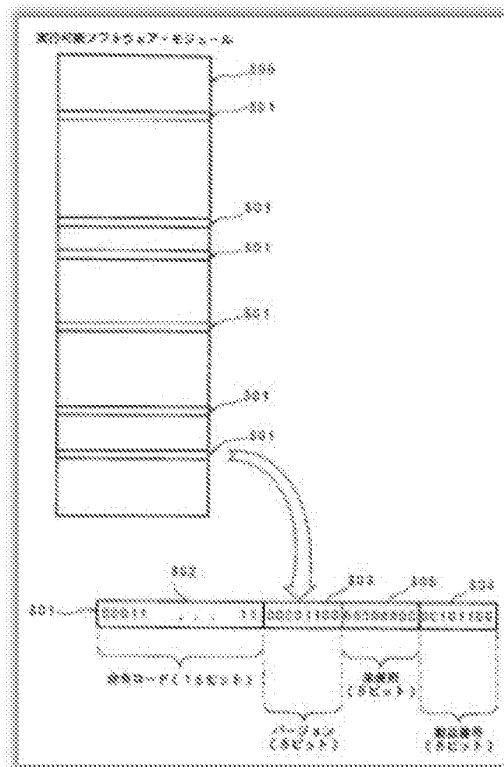
138. When generating its software code, Beetcher '072 teaches that the code includes a series of entitlement verification triggering instructions.⁸³ These triggering instructions are encoded into the software when being compiled and translated, as shows in Figure 3 provided below:

⁸⁰ Beetcher '072 at ¶ 0028.

⁸¹ Beetcher '072 at ¶ 0040, Figs. 5, 9a.

⁸² Beetcher '072 at ¶¶ 0032, 0036, 0041, 0042, Fig. 6, Fig. 9a.

⁸³ Beetcher '072 at ¶¶ 0029, 0044; *see also* Beetcher '072 at ¶¶ 0021, 0033-34, 0037-38.



139. Whenever Beetcher '072's software code encounters one of the verification triggers, the code verifies that the customer is entitled to use the software. It does so by accessing the license key information stored in the key table 460.⁸⁴ For instance, Beetcher '072 details that the customer's computer will access routines, such as check lock function 422, to interpret the license code information contained in one of the triggers:

When a command is the qualification verification trigger 301 (Step 1004), the lock checking feature 422 is called. At Step 1005, the lock checking feature 422 accesses the product locking table entry 601 to which it corresponds to the product number included in a qualification verification trigger. The qualification for the version number in the product locking table 460 being equal to the version number 303 contained in the qualification verification trigger 301, or performing software, in being larger than it is given (Step 1006). In this case, the lock checking feature 422

⁸⁴ Beetcher '072 at ¶¶ 0043-44; see also Beetcher '072 at Abstract, ¶¶ 0034, 0037-38, Fig. 10.

does not perform treatment beyond it, but a system proceeds to execution of the next target code command in a software module.⁸⁵

Thus, a POSITA would have understood that Beetcher '072 teaches using license information in a routine designed to decode a first license code encoded in a software product.

140. Additionally, Beetcher '072 teaches that the triggering instructions will be encoded into the code resources to control software functionality:

[An] additional barrier[] is defining a qualification verification trigger, as other functions of a certain are performed simultaneously.... This alternate function must be selected so that any compiled software modules may include some commands which perform that function quite reliably. When having coincided in these criteria, the compiler can generate automatically the target code which performs the alternate function (it is also a qualification verification trigger simultaneously with it) as a part of the usual compilation order. This definition should bring about the important barrier to 'patching' of a target code which invalidates a qualification verification trigger.⁸⁶

Beetcher '072 further discloses that "a qualification verification trigger is also the direct instruction ... which performs other useful work of a certain... [I]f a trigger command is executed, the system 101 will perform other operations of a certain simultaneously with qualification verification."⁸⁷

141. Therefore, each limitation of element 11.3 is disclosed by Beetcher '072. And as I explain above, Beetcher '072 discloses all the other elements of claim 11. Thus, in my opinion, claim 11 is anticipated by Beetcher '072.

⁸⁵ Beetcher '072 at ¶ 0043, Fig. 10.

⁸⁶ Beetcher '072 at ¶ 0044; *see also* Beetcher '072 at ¶¶ 0021, 0029.

⁸⁷ Beetcher '072 at ¶ 0029 (specifying that these functions are those "which does not need to divide, does not need to be ordering the operand for the processing and does not need to be specified").

2. Beetcher '072 Anticipates Independent Claim 12.

a) Claim 12's Preamble

142. The preamble of claim 12 reads: “A method for encoding software code using a computer having a processor and memory, the method comprising.”

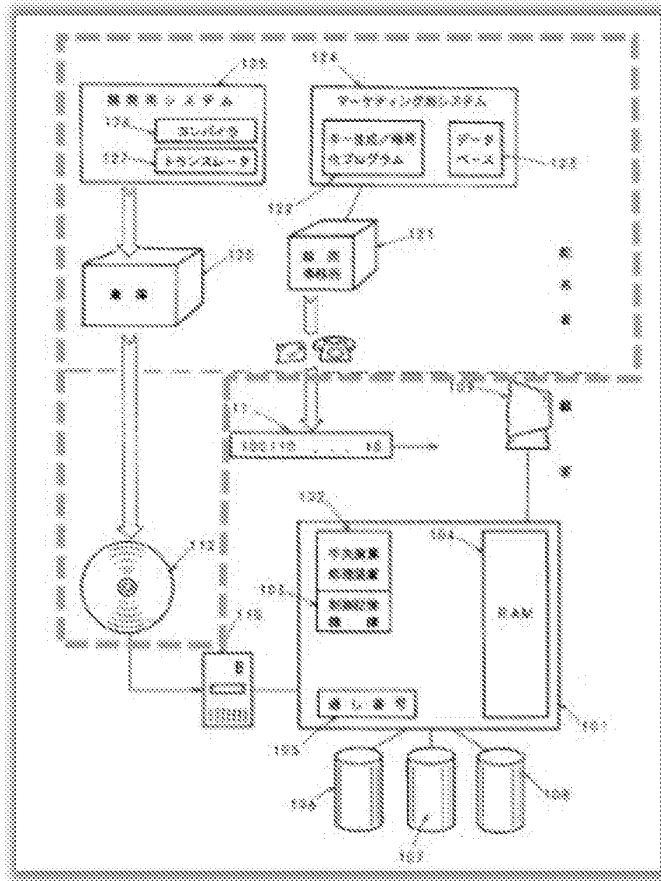
143. I understand that a claim's preamble generally does not limit the scope of the claim under the broadest reasonable interpretation applied during reexamination. Nevertheless, Beetcher '072 discloses claim 12's preamble.

144. Claim 12 recites both a “computer” and a “computer system.” It is unclear whether those elements refer to the same computing device or separate computing devices. When analyzing claim 12 using the broadest reasonable interpretation, I interpret the “computer” recited in the preamble to be a device separate from the term “computer system.”

145. Beetcher '072 teaches a method for encoding software code using a computer with a processor and memory. Beetcher '072 explains that the software distributor has “computer system 125 for development contain the compiler 126 and the translator 127” where “[a] software module is recorded on the software recording medium 112” and “generation/enciphered program 122 of a qualification grant key, and the data base 123 containing customer data.”⁸⁸ Beetcher '072 details these compiling and key generating functions may be performed by a single computer.⁸⁹ Below annotated Figure 1 illustrates the distributor's computer system distributing memory media 112 and compiling encoded software code:

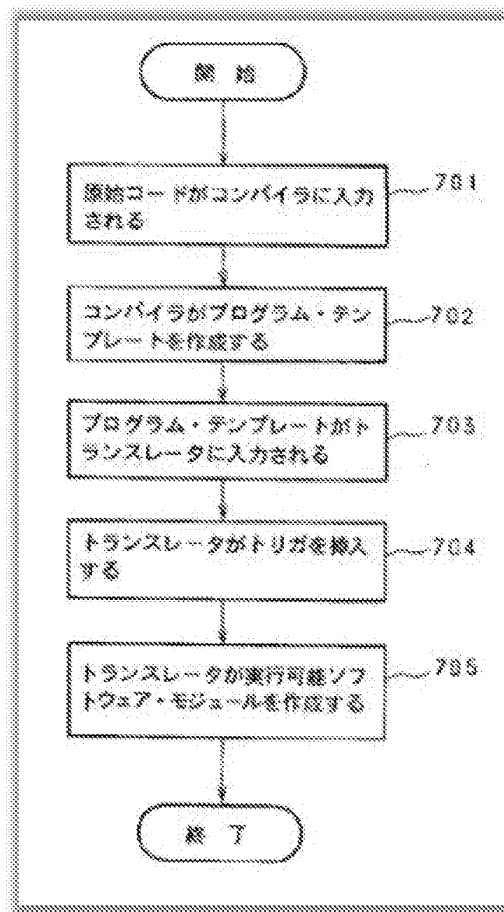
⁸⁸ Beetcher '072 at ¶ 0024; *see also* Beetcher '072 at ¶ 0038.

⁸⁹ Beetcher '072 at ¶ 0024.



146. Beetcher '072's Figure 7 illustrates the software code being encoded to include watermarking triggers decoded by the customer's licensing information.⁹⁰

⁹⁰ Beetcher '072 at ¶ 0038, Fig. 7.



147. Thus, a POSITA would have understood that Beetcher '072's distributor compiles and stores the encode software code using a processor and memory akin to the console's CPU 102 and memory devices 1106-108. Indeed, for as long as computers have been around, it has been standard practice to store the computer code that executes programs—such as the software code used for Beetcher '072's invention—in memory. In fact, a POSITA would have had no option but to store Beetcher '072's software code in memory, as this is required in computer programming. Similarly, it has been standard practice to execute such programs using a processor in the computer.

148. As I detail below, Beetcher '072 teaches the remaining steps that comprise the method.

b) Element 12.1

149. The first element of claim 12 reads: “storing a software code in said memory.” I refer to this as Element 12.1 throughout this declaration.

150. Beetcher '072 discloses element 12.1. Beetcher '072 teaches a development system 125 for compiling and translating for the software code.⁹¹ Beetcher '072 discloses that the software code is stored as disks 112 in warehouse 120. A POSITA would have understood that developer system 125 stores the compiled and translated code in memory and records that code onto disks 112 for distribution to customers. And as I discuss regarding claim 12's preamble, it has been standard practice to store computer code—such as Beetcher '072's software code—in memory. In fact, a POSITA would have had no option but to store this software code in memory, as this is required in computer programming.

151. Thus, each limitation of element 12.1 is disclosed by Beetcher '072.

c) Element 12.2

152. The second element of 12.2 reads: “wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system.” I refer to this as element 12.2 throughout this declaration.

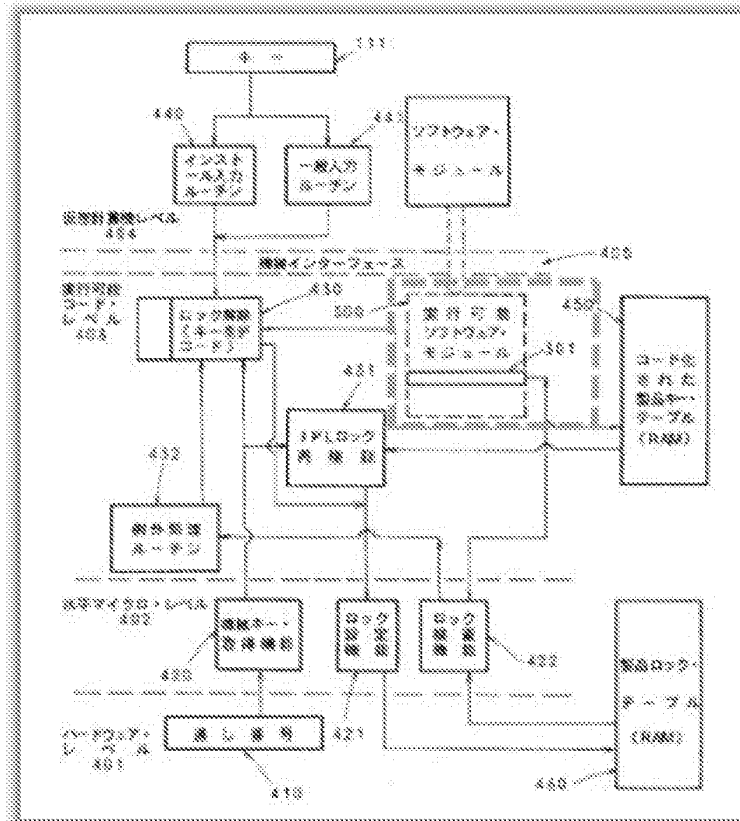
153. Beetcher '072 discloses element 12.2. Beetcher '072 states that its software code has multiple code resources that include a first code resource.⁹² Beetcher '072's code resources include software modules 300 (dashed box) including sub-objects within the code, as shown below in annotated Figure 4 and Figure 3.⁹³ These sub-objects control multiple functions of the

⁹¹ Beetcher '072 at ¶¶ 0024, 0038.

⁹² Beetcher '072 at ¶¶ 0024, 0026-27.

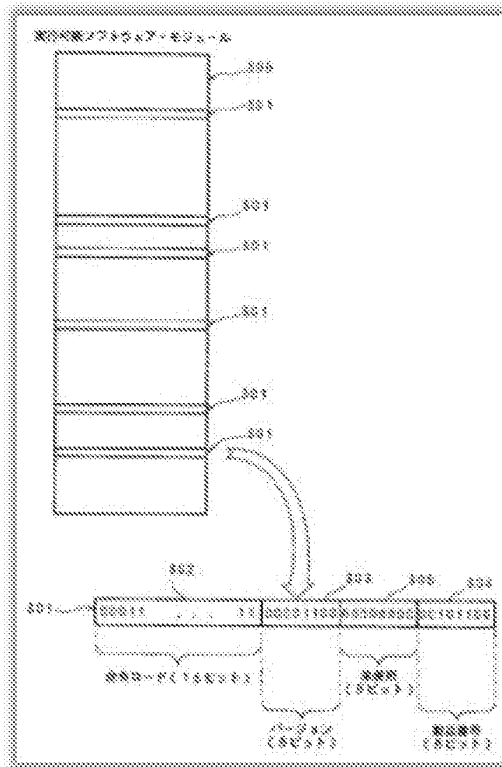
⁹³ Beetcher '072 at ¶¶ 0029, 0034, Fig. 4; *see also* Beetcher '072 at ¶ 0032, Fig. 3.

software installed on the customer's computer system 101.⁹⁴ And Beetcher '072's software prevents unwanted "patching" of these sub-objects by including entitlement verification triggering instructions 301.⁹⁵



⁹⁴ Beetcher '072 at ¶¶ 0029, 0044; *see also* Beetcher '072 at Abstract, ¶¶ 0021, 0030, claim 3.

⁹⁵ Beetcher '072 at ¶¶ 0021, 0044; *see also* Beetcher '072 at Abstract, ¶ 0009.



154. The '842 Patent refers to sub-objects and a memory scheduler as examples of code resources.⁹⁶ A POSITA would have understood that Beetcher '072's module sub-objects are sub-objects.

155. Relying on Beetcher '072's description, a POSITA would have understood that one sub-object in module 300) is a first code resource providing a specified underlying functionality when installed on the customer's computer system 101 and unlocked using the license information (key).

156. Thus, each limitation of element 12.2 is disclosed by Beetcher '072.

⁹⁶ '842 Patent at 11:55-65, 15:36-42.

d) *Element 12.3*

157. The third element of claim 12 reads: “encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key encoded software code.” I refer to this as Element 12.3 throughout this declaration.

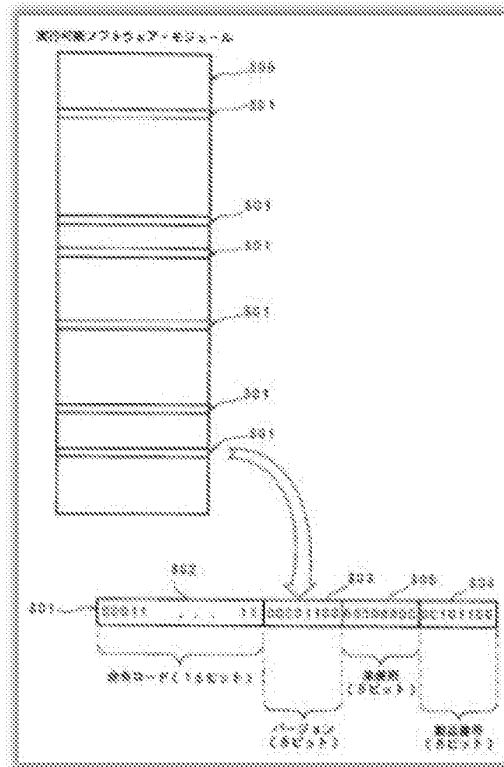
158. Beetcher '072 discloses element 12.3. Beetcher '072 details encoding its software code by the distributor system which includes development system 125 and marketing system 124, “single computer systems may be physically used performs both of functions.”⁹⁷ Beetcher '072 describes encoding a first license key into the software code where that key is used to authorize access to the software product:

The software modules 300 are some program products of the compiled target code form which is performed on the system 101.... [T]he code which can actually be executed operates on the executable code level 403 as shown by the frame of the broken lines. The executable code contains the qualification verification trigger 301 (only one is shown in the figure) performed by the lock checking feature 422 of a horizontal microcode.⁹⁸

159. This encoding is shown in Figure 3:

⁹⁷ Beetcher '072 at ¶¶ 0024, 0029, 0044.

⁹⁸ Beetcher '072 at ¶ 0034; *see also* Beetcher '072 at ¶¶ 0020-21, 0028-29, 0032, 0037, 0040-41.



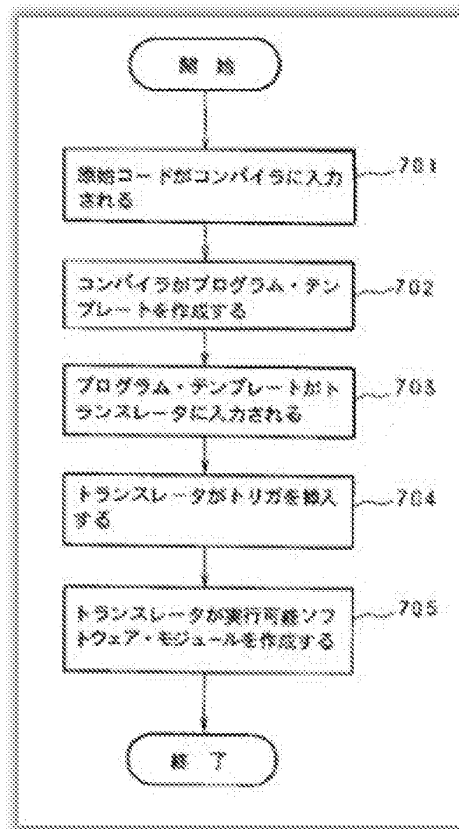
160. The computer in Beetcher '072's development system 125 performs the encoding, as shown in Figure 7 at step 704, described as: "At Step 704, a program template identifies the product number and version number, and it works as an input to the translator 127.

Automatically, the translator 127 generates most number of qualification verification triggers, inserts this in the random position in a target code"⁹⁹

161. Moreover, the computer in Beetcher '072's development system 125 uses an encoding algorithm to encode the first license key. Beetcher '072's system uses a set of instruction, as shown in Figure 7, to encode triggers into the software code to form the first license key.¹⁰⁰

⁹⁹ Beetcher '072 at ¶ 0038; *see also* Beetcher '072 at ¶ 0024, Fig. 7.

¹⁰⁰ Beetcher '072 at ¶ 0038; *see also* Beetcher '072 at ¶ 0024, Fig. 7.



162. The compiler begins the process by producing a template (step 702), next the template is input into the translator (step 703), then the translator encodes the triggers/license keys into the code (step 704), and finally the translator resolves references after key insertion to produce the executable module. The generation of a “number of qualification verification triggers” and “insert[ing] this in the random position in a target code” that would require “a qualification grant key” would require an encoding algorithm.¹⁰¹ Therefore, a POSITA would have understood Beecher ’072’s Figure 7 illustrates an encoding algorithm. Beecher ’072’s encoding process is further described with respect to element 11.3.

¹⁰¹ Beecher ’072 at ¶ 0038.

163. Moreover, during the original prosecution, Patent Owner stated that “[e]ncoding using a key and an algorithm is known.”¹⁰² Thus, a POSITA would have understood that Beetcher ’072’s encoding technique necessarily includes a first license key and an encoding algorithm to form a first license key encoded software code.

164. Thus, each limitation of element 12.3 is disclosed by Beetcher ’072.

e) Element 12.4

165. The fourth element of claim 12 reads: “wherein, when installed on a computer system, said first license key encoded software code will provide said specified underlying functionality only after receipt of said first license key.” I refer to this as Element 12.4 throughout this declaration.

166. Beetcher ’072 discloses element 12.4. Beetcher ’072 teaches that its first license key encoded software code provides the specified underlying functionality only after receipt of the first license key.¹⁰³ For instance, Beetcher ’072 explains:

[I]nvalidating a qualification verification trigger, in order that the format of a target code may support the compile course of the conventional type known by the customer] - - or it may become suitable to add the barrier to ‘patching’ of a target code which is changed. One of such the additional barriers is defining a qualification verification trigger, as other functions of a certain are performed simultaneously. In this case, it is important that the alternate function carried out by the qualification verification trigger cannot carry out with other simple commands. This alternate function must be selected so that any compiled software modules may include some commands which perform that function quite reliably. When having coincided in these criteria, the compiler can generate automatically the target code which performs the alternate function (it is also a qualification verification trigger simultaneously with it) as a part of the usual compilation order.

¹⁰² ’842 Prosecution History at 519.

¹⁰³ Beetcher ’072 at ¶¶ 0029, 0044; *see also* Beetcher ’072 at Abstract, ¶¶ 0009, 0021, 0030, claim 3.

This definition should bring about the important barrier to ‘patching’ of a target code which invalidates a qualification verification trigger.¹⁰⁴

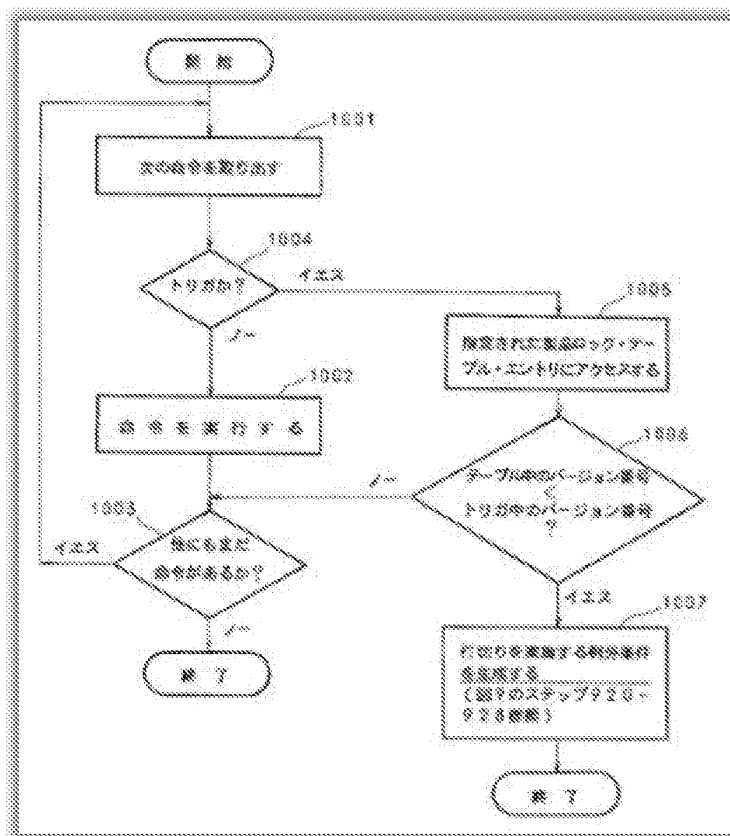
167. And as described with respect to element 12.3, Beetcher '072 discloses encoding the triggering instructions into the software code that is decoded via the first license key.

168. Beetcher '072's Figure 10, as provided below, illustrates providing the software's underlying functionality based on the first license key (triggering information). For instance, Beetcher '072 explains:

Execution of the software module by the system 101 is made by what this is taken out and performed for (Step 1002) (Step 1001) until a modular target code command is completed (step 1003). When a command is the qualification verification trigger 301 (Step 1004), the lock checking feature 422 is called. At Step 1005, the lock checking feature 422 accesses the product locking table entry 601 to which it corresponds to the product number included in a qualification verification trigger. The qualification for the version number in the product locking table 460 being equal to the version number 303 contained in the qualification verification trigger 301, or performing software, in being larger than it is given (Step 1006).¹⁰⁵

¹⁰⁴ Beetcher '072 at ¶ 0044.

¹⁰⁵ Beetcher '072 at ¶ 0043.



169. Thus, each limitation of element 12.4 is disclosed by Beetcher '072. And as I explain above, Beetcher '072 discloses all the other elements of claim 12. Therefore, in my opinion, claim 12 is anticipated by Beetcher '072.

3. Beetcher Anticipates Independent Claim 13.

a) Claim 13's Preamble

170. The preamble of claim 13 reads: "A method for encoding software code using a computer having a processor and memory, comprising."

171. I understand that a claim's preamble generally does not limit the scope of the claim under the broadest reasonable interpretation applied during reexamination. Nevertheless, Beetcher '072 discloses claim 13's preamble.

172. Claim 13's preamble appears to be the same as claim 12's preamble. And as I explain above, Beetcher '072 teaches a method for encoding software using a computer with a processor and memory. Thus, Beetcher '072 teaches this preamble.

b) Element 13.1

173. The first element of claim 13 reads: "storing a software code in said memory." I refer to this as Element 13.1 throughout this declaration.

174. Element 13.1 is identical to element 12.1, which I discuss above. For the same reasons as I explain above, Beetcher '072 discloses each limitation of element 13.1.

c) Element 13.2

175. The second element of claim 13 reads: "wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system." I refer to this as Element 13.2 throughout this declaration.

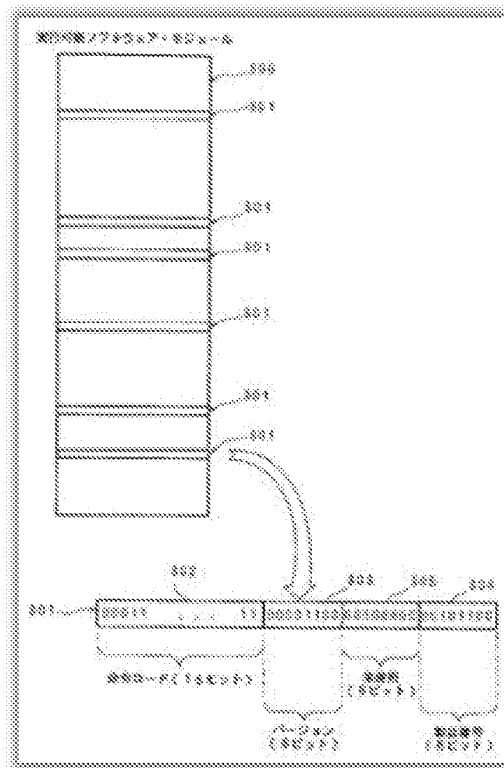
176. Element 13.2 is identical to element 12.2, which I discuss above. For the same reasons as I explain above, Beetcher '072 discloses each limitation of element 13.2.

d) Element 13.3

177. The third element of claim 13 reads: "modifying, by said computer, using a first license key and an encoding algorithm, said software code, to form a modified software code; and wherein said modifying comprises encoding said first code resource to form an encoded first code resource." I refer to this as Element 13.3 throughout this declaration.

178. Beetcher '072 discloses element 13.3. As described with respect to element 12.3, Beetcher '072's distributor system includes a computer that encodes software code using a first license key (e.g., triggering information) and an encoding algorithm (e.g., Figure 7). And Beetcher '072's encoding process modifies the software code by inserting triggering information

into the software code.¹⁰⁶ For instance, Beetcher '072 teaches that compiled software code is input to a translator which modifies the code by “automatically ... generat[ing] most number of qualification verification triggers” and “insert[ing] this in the random position in a target code,” as shown in Figure 7's steps 703 and 704.¹⁰⁷ Figure 3 illustrates this modifying by inserting triggering information 301 to form a modified software code:



179. As described with respect to elements 12.2, Beetcher '072's software code includes a series of code resources corresponding to sub-objects. And Beetcher '072 details a given first

¹⁰⁶ Beetcher '072 at ¶¶ 0034, 0038, *see also* Beetcher '072 at ¶ 0024, Fig. 7.

¹⁰⁷ Beetcher '072 at ¶ 0038.

code resource is modified to encode the first code resource via the triggering information.¹⁰⁸ For instance, Beetcher '072 teaches:

[I]nvalidating a qualification verification trigger, in order that the format of a target code may support the compile course of the conventional type known by the customer] - - or it may become suitable to add the barrier to 'patching' of a target code which is changed. One of such the additional barriers is defining a qualification verification trigger, as other functions of a certain are performed simultaneously. In this case, it is important that the alternate function carried out by the qualification verification trigger cannot carry out with other simple commands. This alternate function must be selected so that any compiled software modules may include some commands which perform that function quite reliably. When having coincided in these criteria, the compiler can generate automatically the target code which performs the alternate function (it is also a qualification verification trigger simultaneously with it) as a part of the usual compilation order. This definition should bring about the important barrier to 'patching' of a target code which invalidates a qualification verification trigger.¹⁰⁹

A POSITA would have understood that such modification results in an encoded first code resource.

180. Moreover, during the original prosecution, Patent Owner stated that "[e]ncoding using a key and an algorithm is known."¹¹⁰ Therefore, a POSITA would have understood that Beetcher '072's encoding technique necessarily includes a first license key and an encoding algorithm to form a modified encoded first code resource.

181. Thus, each limitation of element 13.3 is disclosed by Beetcher '072.

e) *Element 13.4*

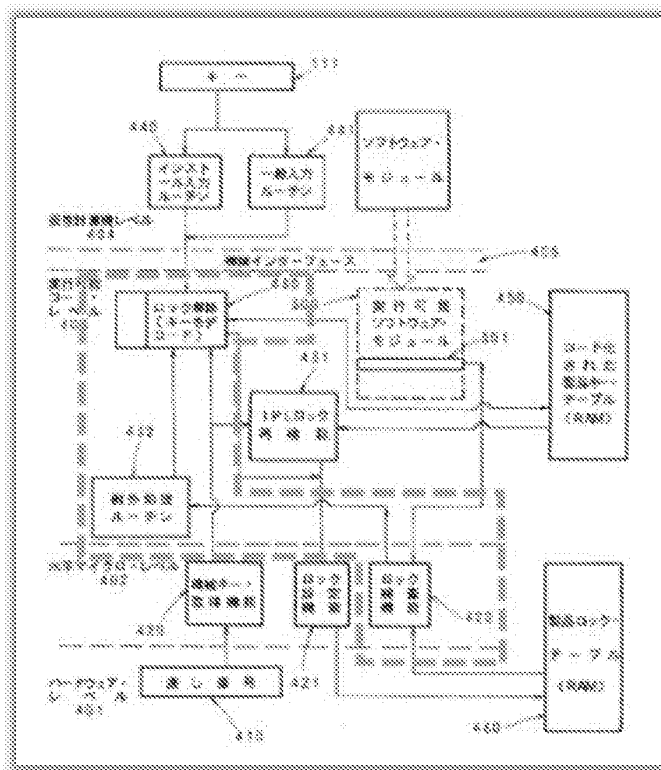
182. The fourth element of claim 13 reads: "wherein said modified software code comprises said encoded first code resource, and a decode resource for decoding said encoded first code resource." I refer to this as Element 13.4 throughout this declaration.

¹⁰⁸ Beetcher '072 at ¶¶ 0021, 0044; *see also* Beetcher '072 at Abstract, ¶ 0009.

¹⁰⁹ Beetcher '072 at ¶ 0044.

¹¹⁰ '842 Prosecution History at 519.

183. Beetcher '072 discloses element 13.4. Beetcher '072 teaches that its modified software code includes a decode resource for decoding the encoded first code resource. Beetcher '072 explains that executing a trigger 301 invokes check lock function 422, which results in accessing "unlock (decode key)" function 430 upon confirmation that the customer possesses the software's license key.¹¹¹ Beetcher '072's Figure 4, as annotated below, depicts the decode resource (dashed perimeter) of the modified software code:



184. Thus, each limitation of element 13.4 is disclosed by Beetcher '072.

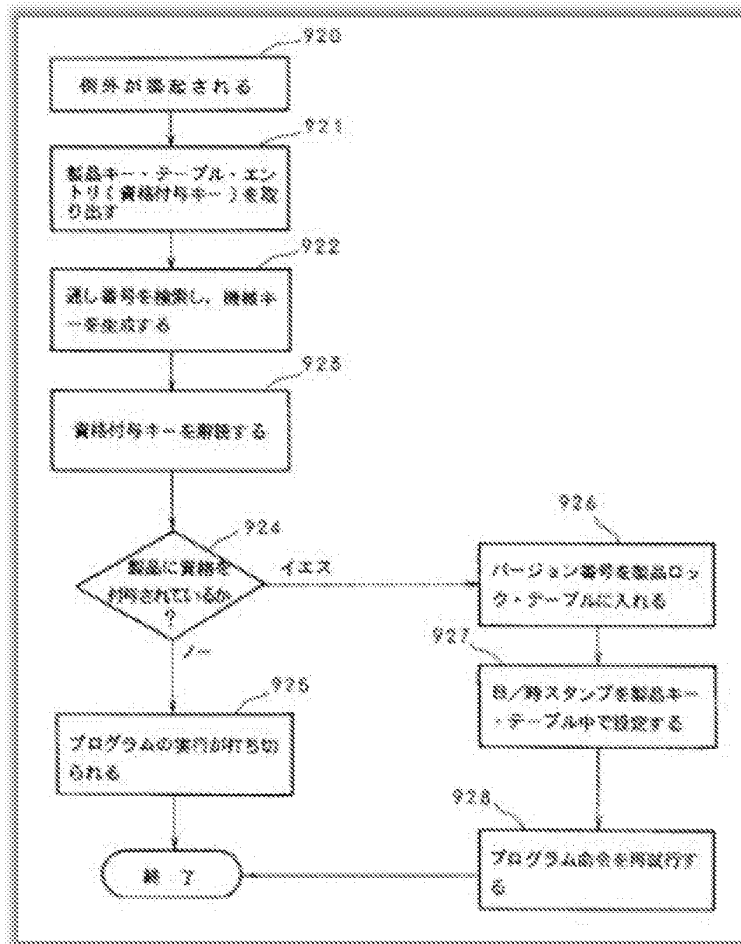
¹¹¹ Beetcher '072 at ¶¶ 0041, 0043, Figs. 9b, 10; see also Beetcher '072 at ¶¶ 0031-32, 0034, 0040.

f) Element 13.5

185. The last element of claim 13 reads: “wherein said decode resource is configured to decode said encoded first code resource upon receipt of said first license key.” I refer to this as Element 13.5 throughout this declaration.

186. Beetcher '072 discloses element 13.5. Beetcher '072 details that its decode resource decodes the encoded first code resource upon receipt of the license key. Beetcher '072, for example, teaches that “the qualification grant key enciphered from the suitable entry in the product key table 450 in which the lock release routine 430 was coded ... is taken out ... and a qualification grant key is decoded Subsequently, at Step 928, a qualification verification trigger is retried and execution of a program is continued.”¹¹² And Beetcher '072's Figure 9b illustrates accessing the decode resource to decode the encoded first code resources based on the entitlement key, reflected in steps 921 to 928:

¹¹² Beetcher '072 at ¶ 0041.



Thus, a POSITA would have understood that Beetcher '072's decode resource is configured to decode the encoded first code resource based on first license key.

187. Therefore, each limitation of element 13.5 is disclosed by Beetcher '072. And as I explain above, Beetcher '072 discloses all the other elements of claim 13. Thus, in my opinion, claim 13 is anticipated by Beetcher '072.

4. Beetcher '072 Anticipates Independent Claim 14.

a) Claim 14's Preamble

188. The preamble of claim 14 reads: "A method for encoding software code using a computer having a processor and memory, comprising"

189. I understand that a claim's preamble generally does not limit the scope of the claim under the broadest reasonable interpretation applied during reexamination. Nevertheless, Beetcher '072 discloses claim 14's preamble.

190. Claim 14's preamble appears to be the same as each of claim 12 and 13's preamble. As I explain above, Beetcher '072 teaches a method for encoding software using a computer with a processor and memory. Thus, Beetcher '072 discloses this preamble.

b) Element 14.1

191. The first element of claim 14 reads: "storing a software code in said memory." I refer to this as Element 14.1 throughout this declaration.

192. Element 14.1 is identical to element 12.1, which I discuss above. For the same reasons as I explain above, Beetcher '072 teaches each limitation of element 14.1.

c) Element 14.2

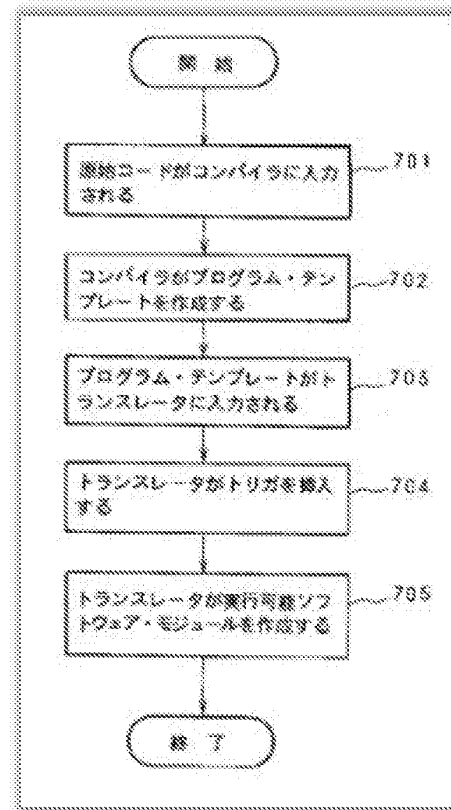
193. The second element of claim 14 reads: "wherein said software code defines software code interrelationships between code resources that result in a specified underlying functionality when installed on a computer system." I refer to this as Element 14.2 throughout this declaration.

194. Beetcher '072 discloses element 14.2. Beetcher '072 teaches that its software code is compiled into executable code by compiler 126. This compiler works with translator 127 to compile the software sub-objects and insert triggering information.¹¹³ And Beetcher '072 details that translator 127 generates the verification triggers and randomly inserts the triggers into the target code.¹¹⁴ Translator 127 then resolves references to the positions of the triggers in the target

¹¹³ Beetcher '072 at ¶ 0034.

¹¹⁴ Beetcher '072 at ¶ 0038.

code, which corresponds to defining code interrelationships between code resources.¹¹⁵ As shown in steps 701 and 702 of Figure 7, Beetcher '072 specifies its software code is input into compiler 126 that produces a template of the software code.¹¹⁶



195. A POSITA would have understood that this software code template also defines the code interrelationships between the code resources. As the Patent Owner stated during the original prosecution, software code interrelationships are defined during the compiling process of conventional software applications:

What the examiner has implied by alleging that the "specification ... fails to teach or mention 'software code interrelationships'" is that software code interrelationships were somehow unknown in the art, which clearly is not the case.

¹¹⁵ Beetcher '072 at ¶¶ 0038.

¹¹⁶ Beetcher '072 ¶¶ 0034, 0038, Fig. 7; see also Beetcher '072 at ¶¶ 0024, 0029, 0033.

As admitted, in the specification at the beginning of paragraph [0051], an "application" comprises "sub-objects" whose "order in the computer memory is of vital importance" in order to perform an intended function. And as admitted further in paragraph [0051], **"When a program is compiled, then, it consists of a collection of these sub-objects, whose exact order or arrangement in memory is not important, so long as any sub-object which uses another sub-object knows where in memory it can be found."** Paragraph [0051] of course refers to conventional applications. Accordingly, that is admittedly a discussion of what is already know by one skilled in the art. Accordingly, the examiner's statement that the specification lacks written description support for "software code interrelationships" is inconsistent with the fact that such **interrelationships were explained in paragraphs [0051] and [0052] as a fundamental basis of pre-existing modem computer programs.**¹¹⁷

196. Moreover, during the original prosecution, Patent Owner stated that "interrelationships between code resource are not that which is novel."¹¹⁸ Based on the Patent Owner's concessions, it is clear that a POSITA would have understood that Beetcher '072's code necessarily defines code interrelationships between code resources.

197. Beetcher '072 further discloses that the code resource interrelationships specify the underlying application functionalities when installed on the customer's computer 101. For instance, Beetcher '072's software code includes multiple entitlement verification triggers.¹¹⁹

And Beetcher '072 teaches that certain code resources include triggering instruction that controls the underlying functionalities of the software code:

[An] additional barrier[] is defining a qualification verification trigger, as other functions of a certain are performed simultaneously.... This alternate function must be selected so that any compiled software modules may include some commands which perform that function quite reliably. When having coincided in these criteria, the compiler can generate automatically the target code which performs the alternate function (it is also a qualification verification trigger simultaneously with it) as a part of the usual compilation order. This definition should bring about the

¹¹⁷ '842 Prosecution History at 519.

¹¹⁸ '842 Prosecution History at 519.

¹¹⁹ Beetcher '072 at ¶¶ 0021, 0038, 0041, Fig. 3; *see also* Beetcher '072 at ¶¶ 0029, 0034, 0043-44.

important barrier to ‘patching’ of a target code which invalidates a qualification verification trigger.¹²⁰

198. Beetcher ’072 further discloses that “a qualification verification trigger is also the direct instruction ... which performs other useful work of a certain.... [I]f a trigger command is executed, the system 101 will perform other operations of a certain simultaneously with qualification verification.”¹²¹ Thus, a POSITA would have understood that the code interrelationships between Beetcher ’072’s code resources result in a specified underlying functionality once installed.

199. Thus, each limitation of element 14.2 is disclosed by Beetcher ’072.

d) Element 14.3

200. The third element of claim 14 reads: “encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key encoded software code.” I refer to this as Element 14.3 throughout this declaration.

201. Element 14.3 is identical to element 12.3, which I address above. For the same reasons I explain above, Beetcher ’072 discloses each limitation of element 14.3.

202. Moreover, during the original prosecution, Patent Owner stated that “[e]ncoding using a key and an algorithm is known” and that “an interrelationship in software code is necessarily defined by digital data, and digital data can obviously be encoded by an encoding process.”¹²²

Therefore, a POSITA would have understood that Beetcher ’072’s encoding technique

¹²⁰ Beetcher ’072 at ¶ 0044; *see also* Beetcher ’072 at ¶¶ 0021, 0029.

¹²¹ Beetcher ’072 at ¶ 0029 (specifying that these functions are those “which does not need to divide, does not need to be ordering the operand for the processing and does not need to be specified”).

¹²² ’842 Prosecution History at 519.

necessarily includes a first license key and an encoding algorithm to form a first license key encoded software code.

e) Element 14.4

203. The fourth element of claim 14 reads: “in which at least one of said software code interrelationships are encoded.” I refer to this as Element 14.4 throughout this declaration.

204. Beetcher '072 discloses element 14.4. As described with respect to element 14.2, Beetcher '072 explains that its software code defines code interrelationships between code resources and triggering information 301 in the code control certain underlying software functionality. And Beetcher '072 teaches that triggering information 301 is encoded into the software code.¹²³ For instance, Beetcher '072 details that the triggering instructions will be encoded into the code resources controlling software functionality:

[An] additional barrier[] is defining a qualification verification trigger, as other functions of a certain are performed simultaneously.... This alternate function must be selected so that any compiled software modules may include some commands which perform that function quite reliably. When having coincided in these criteria, the compiler can generate automatically the target code which performs the alternate function (it is also a qualification verification trigger simultaneously with it) as a part of the usual compilation order. This definition should bring about the important barrier to ‘patching’ of a target code which invalidates a qualification verification trigger.¹²⁴

205. And Beetcher '072 teaches that “a qualification verification trigger is also the direct instruction ... which performs other useful work of a certain.... [I]f a trigger command is executed, the system 101 will perform other operations of a certain simultaneously with

¹²³ Beetcher '072 at ¶¶ 0021, 0029, 0044.

¹²⁴ Beetcher '072 at ¶ 0044; *see also* Beetcher '072 at ¶¶ 0021, 0029.

qualification verification.”¹²⁵ Therefore, a POSITA would have understood that this encoded triggering information includes encoded code interrelationship of the code resources.

206. Therefore, each limitation of element 14.4 is disclosed by Beetcher '072. And as I explain above, Beetcher '072 discloses all the other elements of claim 14. Thus, in my opinion, claim 14 is anticipated by Beetcher '072.

C. Claims 11, 12, 13, and 14 are Anticipated by Cooperman.

1. Cooperman Anticipates Independent Claim 11.

a) Claim 11's Preamble

207. The preamble of claim 11 reads: “A method for licensed software use, the method comprising.”

208. I understand that a claim's preamble generally does not limit the scope of the claim under the broadest reasonable interpretation applied during reexamination. Nevertheless, Cooperman discloses claim 11's preamble.

209. Cooperman describes a method for use of licensed software.¹²⁶ Cooperman, for instance, teaches a method of encoding a license key into software code where the code operates by “ask[ing] the user for personalization information, which include the license code.”¹²⁷ And Cooperman explains that, to extract a digital watermark essential to operate the software, “the

¹²⁵ Beetcher '072 at ¶ 0029 (specifying that these functions are those “which does not need to divide, does not need to be ordering the operand for the processing and does not need to be specified”).

¹²⁶ Cooperman at 5:35-6:5, 11:24-33; *see also* Cooperman at 3:24-31, 11:34-37, 12:13-35, claim 2.

¹²⁷ Cooperman at 11:24-33.

user must have a key. The key, in turn, is a function of the license information for the copy of the software in question.”¹²⁸

210. As I detail below, Cooperman teaches the remaining steps that comprise the method.

b) *Element 11.1*

211. The first element of claim 11 reads: “loading a software product on a computer, said computer comprising a processor, memory, an input, and an output, so that said computer is programmed to execute said software product.” I refer to this as Element 11.1 throughout this declaration.

212. Cooperman discloses element 11.1. Cooperman’s system includes a computer having a processor, memory, input, and output. Cooperman initially recognizes that “[a] computer application seeks to provide a user with certain utilities or tools, that is, users interact with a computer or similar device to accomplish various tasks and applications provide the relevant interface.”¹²⁹ And Cooperman teaches loading software object code into “computer memory for the purpose of execution.”¹³⁰ Cooperman further details that software products include functions made from executable object code whose “order in the computer memory is of vital importance.”¹³¹ Therefore, a POSITA would have understood that Cooperman’s computer includes a processor and memory for executing the stored software code. Indeed, for as long as computers have been around, it has been standard practice to store the computer code that executes programs—such as the software product used for Cooperman’s invention—in memory. In fact, a POSITA would have had no option but to store Cooperman’s software code in memory,

¹²⁸ Cooperman at 12:13-16.

¹²⁹ Cooperman at 3:16-20.

¹³⁰ Cooperman at claim 5; *see also* Cooperman at 13:31-36, claim 7.

¹³¹ Cooperman at 7:1-5.

as this is required in computer programming. Similarly, it has been standard practice to execute such programs using a processor in the computer.

213. Cooperman details that the computer may “process[] a digital sample stream for the purpose of modifying it or playing the digital sample stream.”¹³² A POSITA would have understood that such digital sample stream processing is performed by a computer’s processor and an output plays the digital sample stream.

214. Cooperman further teaches loading a software product on the computer, so the computer can execute the software product. Cooperman describes the operation of the disclosed software product requires:

Installing, i.e., loading, the software on the computer;

Asking the user to input a license code;

Generating, i.e., outputting, a decoding key after receiving the license code to access the software resources.¹³³

Thus, each limitation of element 11.1 is disclosed by Cooperman.

c) Element 11.2

215. The second element of claim 11 reads: “said software product outputting a prompt for input of license information.” I refer to this as Element 11.2 throughout this declaration for convenience.

216. Cooperman discloses element 11.2. Cooperman teaches that its software product requests that the user input license information, i.e., a license key, into the computer before the product can be used.¹³⁴ For instance, Cooperman explains that the software product prompts the user to

¹³² Cooperman at claim 4; *see also* Cooperman at claims 5, 6 (processing digital sample stream and a map list).

¹³³ Cooperman at 11:24-34.

¹³⁴ Cooperman 11:24-33; *see also* Cooperman at Abstract, 3:24-28, 5:35-6:5, 11:6-8, 12:10-16.

input license information: “1) when it is run for the first time, after installation, it asks the user for personalization information, which includes the license code. This can include a particular computer configuration.”¹³⁵ Cooperman details that such license code are entered by the user “when prompted at start-up.”¹³⁶ A POSITA would have understood this request corresponds to the software product outputting a prompt to input license information.

217. During the original prosecution, Patent Owner confirmed that such a teaching disclosed by Cooperman meets element 11.2. For instance, Patent Owner’s 05/14/2012 Appeal Brief states that element 11.2 is taught by: “1) when it is run for the first time, after installation, it asks the user for personalization information, which includes the license code. This can include a particular computer configuration.”¹³⁷ Cooperman includes this same teaching.

218. Thus, each limitation of element 11.2 is disclosed by Cooperman.

d) Element 11.3

219. The third element of claim 11 reads: “said software product using license information entered via said input in response to said prompt in a routine designed to decode a first license code encoded in said software product.” I refer to this as Element 11.3 throughout this declaration for convenience.

220. Cooperman discloses element 11.3. Cooperman teaches that its system includes a routine designed to decode a first license code encoded in the software product based on inputted license information. For instance, Cooperman explains:

Given that there are one or more of these essential resources, what is needed to realize the present invention is the presence of certain data resources of a type which

¹³⁵ Cooperman at 11:25-28.

¹³⁶ Cooperman at 1:25-28.

¹³⁷ ‘842 Prosecution History at 577 (original claim 58 issued as claim 11).

are amenable to the "stega-cipher" process described in the "Steganographic Method and Device" patent application.¹³⁸

221. And Cooperman discloses: "3) Once it has the license code, it can then generate the proper decoding key to access the essential code resources."¹³⁹ As I explain regarding element 11.2, Cooperman teaches that the user enters license information via an input in response to the prompt.

222. During the original prosecution, Patent Owner confirmed that such a teaching disclosed by Cooperman meets element 11.3. For instance, Patent Owner's 05/14/2012 Appeal Brief states that element 11.3 is taught by:

Given that there are one or more of these essential resources, what is needed to realize the present invention is the presence of certain data resources of a type which are amenable to the "stega-cipher" process described in the "Steganographic Method and Device" patent U.S. Pat. No. 5,613,004 [issued from U.S. Application No. 08/489,172].

* * * *

3) Once it has the license code, it can then generate the proper decoding key to access the essential code resources.¹⁴⁰

Cooperman includes these same teachings.

223. Therefore, each limitation of element 11.3 is disclosed by Cooperman. And as I explain above, Cooperman discloses all the other elements of claim 11. Thus, in my opinion, claim 11 is anticipated by Cooperman.

¹³⁸ Cooperman at 9:22-27; *see also* Cooperman at 2:34-37, 4:7-17 (incorporating by reference U.S. Patent Application No. 08/489,172 entitled "Steganographic Method and Device").

¹³⁹ Cooperman at 11:31-33.

¹⁴⁰ '842 Prosecution History at 577 (original claim 58 issued as claim 11); *see also* Cooperman at 664 (Patent Owner explaining that element 11.3 is met by teachings corresponding to Cooperman at 10:7-11:33).

2. Cooperman Anticipates Independent Claim 12.

a) Claim 12's Preamble

224. The preamble of claim 12 reads: “A method for encoding software code using a computer having a processor and memory, the method comprising.”

225. I understand that a claim’s preamble generally does not limit the scope of the claim under the broadest reasonable interpretation applied during reexamination. Nevertheless, Cooperman discloses claim 12’s preamble.

226. Claim 12 recites both a “computer” and a “computer system.” It is unclear whether those elements refer to the same computing device or separate computing devices. When analyzing claim 12 using the broadest reasonable interpretation, I interpret the “computer” recited in the preamble to be a device separate from the term “computer system.”

227. Cooperman teaches a method for encoding software code using a computer with a processor and memory. Cooperman describes that, during the software code assembly, the computer system will “choose one or several essential code resources, and encode them into one or several data resources using the stegacipher process.”¹⁴¹ Indeed, for as long as computers have been around, it has been standard practice to store the computer code that executes programs—such as the software code used for Cooperman’s invention—in memory. In fact, a POSITA would have had no option but to store Cooperman’s software code in memory, as this is required in computer programming. Similarly, it has been standard practice to execute such programs using a processor in the computer.

228. As I explain below, Cooperman teaches the remaining steps that comprise the method.

¹⁴¹ Cooperman at 10:13-16; *see also* Cooperman at claim 6.

b) *Element 12.1*

229. The first element of claim 12 reads: “storing a software code in said memory.” I refer to this as Element 12.1 throughout this declaration.

230. Cooperman discloses element 12.1. Cooperman teaches techniques for randomizing the location of software code stored in memory.¹⁴² Cooperman explains that this randomization makes the software code more resistant to patching and memory capture analysis.¹⁴³ Thus, a POSITA would have understood that these techniques are used for code stored in memory.

231. Cooperman further describes that its software code is compiled and assembled: “When code and data resources are compiled and assembled into a precursor of an executable program the next step is to use a utility application for final assembly of the executable application.”¹⁴⁴ Cooperman also states that code resources are stored separately from application, i.e., software, code.¹⁴⁵ A POSITA would have understood that Cooperman’s compiled and assembled application code is stored in memory. And as I discuss regarding claim 12’s preamble, it has been standard practice to store computer code—such as Cooperman’s software code—in memory. In fact, a POSITA would have had no option but to store this software code in memory, as this is required in computer programming.

232. During the original prosecution, Patent Owner confirmed that such a teaching disclosed by Cooperman meets element 12.1. For example, Patent Owner’s 05/14/2012 Appeal Brief states that element 12.1 is taught by: “When code and data resources are compiled and assembled into

¹⁴² Cooperman at 3:32-37; *see also* Cooperman at 4:1-6, 6:5-9, 13:23-46, 14:4-9.

¹⁴³ Cooperman at 3:13-16, 14:37-15:18, claim 7.

¹⁴⁴ Cooperman at 10:8-11; *see also* Cooperman at 7:1-21.

¹⁴⁵ Cooperman at 7:26-30.

a precursor of an executable program the next step is to use a utility application for final assembly of the executable application.”¹⁴⁶ Cooperman includes this same teaching.

233. Thus, each limitation of element 12.1 is disclosed by Cooperman.

c) Element 12.2

234. The second element of claim 12 reads: “wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system.” I refer to this as Element 12.2 throughout this declaration.

235. Cooperman discloses element 12.2. Cooperman explains that its software code provides certain functionality to the user upon determining that a proper license key has been entered. Cooperman specifies its code has multiple code resources that include a first code resource.¹⁴⁷ And Cooperman teaches that its software code includes the code resources and provides an underlying functionality when installed on the computer.¹⁴⁸ For example, Cooperman states: “The basic premise for this scheme is that there are a certain sub-set of executable code resources, that comprise an application and that are ‘essential’ to the proper function of the application.”¹⁴⁹

236. As another example, Cooperman discloses that software applications include code resources providing functionalities specified in the application:

The memory address of the first instruction in one of these sub-objects is called the "entry point" of the function or procedure. The rest of the instructions comprising

¹⁴⁶ ‘842 Prosecution History at 578 (original claim 59 issued as claim 12); *see also* Cooperman at 415-16 (original claim 61, which issued as claim 13, includes the same limitation “wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system”).

¹⁴⁷ Cooperman at 10:11-29, 11:13-33; *see also* Cooperman at Abstract, 7:26-30, 9:10-21, 13:31-36, claim 6.

¹⁴⁸ Cooperman at 7:19-36, 11:24-37; *see also* Cooperman at 8:30-33, 10:11-29.

¹⁴⁹ Cooperman at 8:30-33.

that sub-object immediately follow from the entry point. Some systems may prefix information to the entry point which describes calling and return conventions for the code which follows, an example is the Apple Macintosh Operating System (MacOS). These sub-objects can be packaged into what are referred to in certain systems as "code resources," which may be stored separately from the application, or shared with other applications, although not necessarily. Within an application there are also data objects, which consist of some data to be operated on by the executable code. These data objects are not executable. That is, they do not consist of executable instructions. The data objects can be referred to in certain systems as "resources."¹⁵⁰

The '842 Patent refers to sub-objects and a memory scheduler as examples of code resources.¹⁵¹

In this additional way, a POSITA would have understood that Cooperman's code includes a first code resource.

237. During the original prosecution, Patent Owner confirmed that such teachings disclosed by Cooperman meets element 12.2. For example, Patent Owner's 05/14/2012 Appeal Brief states that element 12.2 is taught by: "The basic premise for this scheme is that there are a certain subset of executable code resources, that comprise an application and that are 'essential' to the proper function of the application."¹⁵² As another example, Patent Owner's 05/14/2012 Appeal Brief states this element is taught by:

The memory address of the first instruction in one of these sub-objects is called the "entry point" of the function or procedure. The rest of the instructions comprising that sub-object immediately follow from the entry point. Some systems may prefix information to the entry point which describes calling and return conventions for the code which follows, an example is the Apple Macintosh Operating System (MacOS). These sub-objects can be packaged into what are referred to in certain systems as "*code resources*," which may be stored separately from the application, or shared with other applications, although not necessarily. Within an application there are also data objects, which consist of some data to be operated on by the executable code. These data objects are not executable. That is, they do not consist

¹⁵⁰ Cooperman at 7:19-36.

¹⁵¹ '842 Patent at 11:55-65, 15:36-42.

¹⁵² '842 Prosecution History at 578 (original claim 59 issued as claim 12).

of executable instructions. The data objects can be referred to in certain systems as "resources."¹⁵³

Cooperman includes these same teachings.

238. Thus, each limitation of element 12.2 is disclosed by Cooperman.

d) Element 12.3

239. The third element of claim 12 reads: "encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key encoded software code." I refer to this as Element 12.3 throughout this declaration.

240. Cooperman discloses element 12.3. Cooperman teaches encoding its software code to form a first license key encode software code.¹⁵⁴ Cooperman describes that this encoding uses a first license key and an encoding algorithm.¹⁵⁵ For instance, Cooperman details that "[t]he assembly utility can be supplied with a key generated from a license code generated for the license in question."¹⁵⁶ And Cooperman states: "The utility will choose one or several essential code resources, and encode them into one or several data resources using the stegacipher process."¹⁵⁷

241. During the original prosecution, Patent Owner confirmed that such a teaching disclosed by Cooperman meets element 12.3. For instance, Patent Owner's 05/14/2012 Appeal Brief states

¹⁵³ '842 Prosecution History at 579-80 (original claim 61, which issued as claim 13, includes the same limitation "wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system").

¹⁵⁴ Cooperman at 10:28-35, 11:6-15; *see also* Cooperman at 2:27-31, 3:24-31, 12:13-23, claim 6.

¹⁵⁵ Cooperman at 10:13-16, 11:9-11, claim 6.

¹⁵⁶ Cooperman at 11:9-11.

¹⁵⁷ Cooperman at 10:13-16; *see also* Cooperman at claim 6.

that “encoding, by said computer using at least a first license key and an encoding algorithm, said software code” is taught by:

The assembly utility can be supplied with a key generated from a license code generated *for the license in question*.

* * * *

The utility will choose one or several essential code resources, and encode them into one or several data resources using the stegacipher process.¹⁵⁸

242. As another example, Patent Owner’s 05/14/2012 Appeal Brief states that “to form a first license key encoded software code” is taught by:

The purpose of this scheme is to make a particular licensed copy of an application distinguishable from any other. It is not necessary to distinguish every instance of an application, merely every instance of a license.

* * * *

3) Once it has the license code, it can then generate the proper decoding key to access the essential code resources.¹⁵⁹

Cooperman includes this same teaching.

243. Moreover, during the original prosecution, Patent Owner specified that “[e]ncoding using a key and an algorithm is known.”¹⁶⁰ Thus, a POSITA would have understood that Cooperman’s encoding technique necessarily includes a first license key and an encoding algorithm to form a first license key encoded software code.

244. Thus, each limitation of element 12.3 is disclosed by Cooperman.

e) Element 12.4

245. The fourth element of claim 12 reads: “wherein, when installed on a computer system, said first license key encoded software code will provide said specified underlying functionality

¹⁵⁸ ’842 Prosecution History at 578 (emphasis in original) (original claim 59 issued as claim 12).

¹⁵⁹ ’842 Prosecution History at 578-79 (original claim 59 issued as claim 12).

¹⁶⁰ ’842 Prosecution History at 519.

only after receipt of said first license key.” I refer to this as Element 12.4 throughout this declaration.

246. Cooperman discloses element 12.4. Cooperman discloses that its first license key encoded software code provides the specified underlying functionality only after receipt of the first license key.¹⁶¹ For instance, Cooperman explains: “Once it has the license code, it can then generate the proper decoding key to access the essential code resources. Note that the application...must contain the license code issued to the licensed owner, to access its essential code resources.”¹⁶² Cooperman describes that these essential code resources correspond to the underlying functionalities of the software program installed on the computer.¹⁶³

247. Therefore, each limitation of element 12.4 is disclosed by Cooperman. And as I explain above, Cooperman discloses all the other elements of claim 12. Thus, in my opinion, claim 12 is anticipated by Cooperman.

3. Cooperman Anticipates Independent Claim 13.

a) Claim 13’s Preamble

248. The preamble of claim 13 reads: “A method for encoding software code using a computer having a processor and memory, comprising.”

249. I understand that a claim’s preamble generally does not limit the scope of the claim under the broadest reasonable interpretation applied during reexamination. Nevertheless, Cooperman discloses claim 13’s preamble.

¹⁶¹ Cooperman at 10:28-35, 11:6-15; *see also* Cooperman at 2:27-31, 3:24-31, 12:13-23, claim 6.

¹⁶² Cooperman at 11:31-37.

¹⁶³ Cooperman at 5:35-6:9, 11:6-8, 11:31-37, 12:10-16; *see also* Cooperman at 6:26-30, 7:1-5, 8:25-37, 9:14-21.

250. Claim 13's preamble is the same as claim 12's preamble. As I explain above, Cooperman teaches a method for encoding software using a computer with a processor and memory. Thus, Cooperman discloses this preamble.

b) Element 13.1

251. The first element of claim 13 reads: "storing a software code in said memory." I refer to this as Element 13.1 throughout this declaration.

252. Element 13.1 is identical to element 12.1, which I discuss above. For the same reasons as I explain above, Cooperman teaches each limitation of element 13.1.

c) Element 13.2

253. The second element of claim 13 reads: "wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system." I refer to this as Element 13.2 throughout this declaration.

254. Element 13.2 is identical to element 12.2, which I discuss above. For the same reasons as I explain above, Cooperman discloses each limitation of element 13.2.

255. And during the original prosecution, Patent Owner confirmed that such a teaching disclosed by Cooperman meets element 13.2. For instance, Patent Owner's 05/14/2012 Appeal Brief states that element 13.2 is taught by:

The memory address of the first instruction in one of these sub-objects is called the "entry point" of the function or procedure. The rest of the instructions comprising that sub-object immediately follow from the entry point. Some systems may prefix information to the entry point which describes calling and return conventions for the code which follows, an example is the Apple Macintosh Operating System (MacOS). These sub-objects can be packaged into what are referred to in certain systems as "*code resources*," which may be stored separately from the application, or shared with other applications, although not necessarily. Within an application there are also data objects, which consist of some data to be operated on by the executable code. These data objects are not executable. That is, they do not consist

of executable instructions. The data objects can be referred to in certain systems as "resources."¹⁶⁴

As I explain with respect to element 12.2, Cooperman includes this same teaching.

d) Element 13.3

256. The third element of claim 13 reads: “modifying, by said computer, using a first license key and an encoding algorithm, said software code, to form a modified software code; and wherein said modifying comprises encoding said first code resource to form an encoded first code resource.” I refer to this as Element 13.3 throughout this declaration.

257. Cooperman discloses element 13.3. Cooperman teaches modifying its software code using a license key and an encoding algorithm.¹⁶⁵ And Cooperman’s modification includes encoding the first code resource to form an encoded first code resource. For instance, Cooperman describes code modification using a “digital watermarking” process to encode a code resource: “The first method of the present invention described involves hiding necessary ‘parts’ or code ‘resources’ in digitized sample resources using a ‘digital watermarking’ process, such as that described in the ‘Steganographic Method and Device’ patent application.”¹⁶⁶ Cooperman further discloses “watermarking with ‘keys’ derived from license codes... and using the watermarks encoded with such keys to hide an essential subset for the application code resources.”¹⁶⁷ A POSITA would have understood that such modification results in a modified software code.

¹⁶⁴ ’842 Prosecution History at 579-80 (original claim 61 was issued as claim 13).

¹⁶⁵ Cooperman at 3:10-31, 8:25-30, 10:8-31; *see also* Cooperman at 2:19-37, 4:7-17, 11:6-24, claim 6.

¹⁶⁶ Cooperman at 8:25-30; *see also* Cooperman at 2:34-37, 4:7-17 (incorporating by reference U.S. Patent Application No. 08/489,172 entitled “Steganographic Method and Device”).

¹⁶⁷ Cooperman at 5:15-22.

258. During the original prosecution, Patent Owner confirmed that such a teaching disclosed by Cooperman meets element 13.3. For instance, Patent Owner's 05/14/2012 Appeal Brief states that element 13.3 is taught by: "The first method of the present invention described involves hiding necessary 'parts' or code 'resources' in digitized sample resources using a 'digital watermarking' process, such as that described in the 'Steganographic Method and Device' patent application."¹⁶⁸ Cooperman includes this same teaching.

259. Moreover, during the original prosecution, Patent Owner stated that "[e]ncoding using a key and an algorithm is known."¹⁶⁹ Thus, a POSITA would have understood that Cooperman's encoding technique necessarily includes a first license key and an encoding algorithm to form a modified encoded first code resource.

260. Thus, each limitation of element 13.3 is disclosed by Cooperman.

e) Element 13.4

261. The fourth element of claim 13 reads: "wherein said modified software code comprises said encoded first code resource, and a decode resource for decoding said encoded first code resource." I refer to this as Element 13.4 throughout this declaration.

262. Cooperman discloses element 13.4. Cooperman details that its modified software code includes a decode resource for decoding the encoded first code resource.¹⁷⁰ For instance, Cooperman explains the modified application code has a decoding resource: "Note further that the application contains a code resource which performs the function of decoding an encoded code resource from a data resource."¹⁷¹ And Cooperman further teaches that "[o]nce [the

¹⁶⁸ '842 Prosecution History at 580 (original claim 61 issued as claim 13).

¹⁶⁹ '842 Prosecution History at 519.

¹⁷⁰ Cooperman at 11:17-20, claim 6; *see also* Cooperman 11:31-33, claim 5.

¹⁷¹ Cooperman at 11:17-20.

application] has the license code, it can then generate the proper decoding key to access the essential code resources.”¹⁷²

263. During the original prosecution, Patent Owner confirmed that such a teaching disclosed by Cooperman meets element 13.4. For instance, Patent Owner’s 05/14/2012 Appeal Brief states that element 13.4 is taught by: “Note further that the application contains a code resource which performs the function of decoding an encoded code resource from a data resource.”¹⁷³

Cooperman includes this same teaching.

264. Thus, each limitation of element 13.4 is disclosed by Cooperman.

f) Element 13.5

265. The last element of claim 13 reads: “wherein said decode resource is configured to decode said encoded first code resource upon receipt of said first license key.” I refer to this as Element 13.5 throughout this declaration.

266. Cooperman discloses element 13.5. Cooperman teaches that its decode resource decodes the encoded first code resource upon receipt of the license key:

The application must also contain a data resource which specifies in which data resource a particular code resource is encoded. This data resource is created and added at assembly time by the assembly utility. The application can then operate as follows:

- 1) when it is run for the first time, after installation, it asks the user for personalization information, which includes the license code. This can include a particular computer configuration;
- 2) it stores this information in a personalization data resource;
- 3) Once it has the license code, it can then generate the proper decoding key to access the essential code resources.¹⁷⁴

¹⁷² Cooperman at 11:31-33.

¹⁷³ ’842 Prosecution History at 580 (original claim 61 issued as claim 13).

¹⁷⁴ Cooperman at 11:20-33; *see also* Cooperman at claims 5 and 6.

267. During the original prosecution, Patent Owner confirmed that such a teaching disclosed by Cooperman meets element 13.5. For instance, Patent Owner's 05/14/2012 Appeal Brief states that element 13.5 is taught by:

The application must also contain a data resource which specifies in which data resource a particular code resource is encoded. This data resource is created and added at assembly time by the assembly utility. The application can then operate as follows:

- 1) when it is run for the first time, after installation, it asks the user for personalization information, which includes the license code. This can include a particular computer configuration;
- 2) it stores this information in a personalization data resource;
- 3) Once it has the license code, it can then generate the proper decoding key to access the essential code resources.¹⁷⁵

Cooperman includes this same teaching.

268. Therefore, each limitation of element 13.5 is disclosed by Cooperman. And as I explain above, Cooperman teaches all the other elements of claim 13. Thus, in my opinion, claim 13 is anticipated by Cooperman.

4. Cooperman Anticipates Independent Claim 14.

a) Claim 14's Preamble

269. The preamble of claim 14 reads: "A method for encoding software code using a computer having a processor and memory, comprising."

270. I understand that a claim's preamble generally does not limit the scope of the claim under the broadest reasonable interpretation applied during reexamination. Nevertheless, Cooperman discloses claim 14's preamble.

¹⁷⁵ '842 Prosecution History at 580-81 (original claim 61 issued as claim 13).

271. Claim 14's preamble appears to be the same as each of claim 12 and 13's preamble. As I explain above, Cooperman teaches a method for encoding software using a computer with a processor and memory. Thus, Cooperman discloses this preamble.

b) Element 14.1

272. The first element of claim 14 reads: "storing a software code in said memory." I refer to this as Element 14.1 throughout this declaration.

273. Element 14.1 is identical to element 12.1, which I discuss above. For the same reasons as I explain above, Cooperman discloses each limitation of element 14.1.

c) Element 14.2

274. The second element of claim 14 reads: "wherein said software code defines software code interrelationships between code resources that result in a specified underlying functionality when installed on a computer system." I refer to this as Element 14.2 throughout this declaration.

275. Cooperman discloses element 14.2. Cooperman teaches that its software provides certain functionality to the user upon determining that a proper license key has been entered. In doing so, Cooperman's code includes interrelationships between code resources.¹⁷⁶ For instance, Cooperman explains that its software code includes a special code resource, such a memory scheduler, that knows the code interrelationships of all other code resources:

Under the present invention, the application contains a special code resource which knows about all the other code resources in memory. During execution time, this special code resource, called a "memory scheduler," can be called periodically, or at random or pseudo random intervals, at which time it intentionally shuffles the other code resources randomly in memory, so that someone trying to analyze snapshots of memory at various intervals cannot be sure if they are looking at the same code or organization from one "break" to the next. This adds significant complexity to their job. The scheduler also randomly relocates itself when it is finished. In order to do this, the scheduler would have to first copy itself to a new location, and then specifically modify the program counter and stack frame, so that

¹⁷⁶ Cooperman at 14:35-15:17.

it could then jump into the new copy of the scheduler, but return to the correct calling frame. Finally, the scheduler would need to maintain a list of all memory addresses which contain the address of the scheduler, and change them to reflect its new location.¹⁷⁷

276. Cooperman further details its software code as including sub-objects that are code resources that provide entries point to the software's various functions:

The memory address of the first instruction in one of these sub-objects is called the "entry point" of the function or procedure. The rest of the instructions comprising that sub-object immediately follow from the entry point. Some systems may prefix information to the entry point which describes calling and return conventions for the code which follows, an example is the Apple Macintosh Operating System (MacOS). These sub-objects can be packaged into what are referred to in certain systems as "code resources," which may be stored separately from the application, or shared with other applications, although not necessarily.¹⁷⁸

277. And Cooperman teaches that these code resources will be fixed once installed on the computer: "Once the code resources of a program are loaded into memory, they typically remain in a fixed position."¹⁷⁹

278. During the original prosecution, Patent Owner confirmed that such a teaching disclosed by Cooperman meets element 14.2. For example, Patent Owner's 02/28/11 Remarks state that element 14.2 is taught by:

Under the present invention, the application contains a special code resource which knows about all the other code resources in memory.

* * * *

During execution time, this special code resource, called a "memory scheduler," can be called periodically, or at random or pseudo random intervals, at which time it intentionally shuffles the other code resources randomly in memory, so that someone trying to analyze snapshots of memory at various intervals cannot be sure if they are looking at the same code or organization from one "break" to the next. This adds significant complexity to their job. The scheduler also randomly relocates itself when it is finished. In order to do this, the scheduler would have to first copy

¹⁷⁷ Cooperman at 14:35-15:17.

¹⁷⁸ Cooperman at 7:19-30.

¹⁷⁹ Cooperman at 13:31-32.

itself to a new location, and then specifically modify the program counter and stack frame, so that it could then jump into the new copy of the scheduler, but return to the correct calling frame. Finally, the scheduler would need to maintain a list of all memory addresses which contain the address of the scheduler, and change them to reflect its new location.¹⁸⁰

279. And as another example, Patent Owner's 05/14/2012 Appeal Brief explain that element 14.2 is taught by:

The memory address of the first instruction in one of these sub-objects is called the "entry point" of the function or procedure. The rest of the instructions comprising that sub-object immediately follow from the entry point. Some systems may prefix information to the entry point which describes calling and return conventions for the code which follows, an example is the Apple Macintosh Operating System (MacOS). These sub-objects can be packaged into what are referred to in certain systems as "code resources," which may be stored separately from the application, or shared with other applications, although not necessarily.

* * * *

Once the code resources of a program are loaded into memory, they typically remain in a fixed position.¹⁸¹

Cooperman includes these same teachings.

280. Moreover, during the original prosecution, Patent Owner stated that "interrelationships between code resource are not that which is novel."¹⁸² The Patent Owner continues by conceding:

What the examiner has implied by alleging that the "specification ... fails to teach or mention 'software code interrelationships'" is that software code interrelationships were somehow unknown in the art, which clearly is not the case. **As admitted, in the specification at the beginning of paragraph [0051], an "application" comprises "sub-objects" whose "order in the computer memory is of vital importance" in order to perform an intended function.** And as admitted further in paragraph [0051], "When a program is compiled, then, it

¹⁸⁰ '842 Prosecution History at 416 (original claim 62 issued as claim 14) *see also* '842 Prosecution History at 669-71 (Patent Owner explaining that element 14.2 is met by teachings corresponding to Cooperman at 5:18-22, 6:30-7:36).

¹⁸¹ '842 Prosecution History at 581-82 (emphasis in original) (original claim 62 issued as claim 14).

¹⁸² '842 Prosecution History at 519.

consists of a collection of these sub-objects, whose exact order or arrangement in memory is not important, so long as any sub-object which uses another sub-object knows where in memory it can be found." **Paragraph [0051] of course refers to conventional applications. Accordingly, that is admittedly a discussion of what is already know by one skilled in the art.** Accordingly, the examiner's statement that the specification lacks written description support for "software code interrelationships" is inconsistent with the fact that such **interrelationships were explained in paragraphs [0051] and [0052] as a fundamental basis of pre-existing modem computer programs.**¹⁸³

281. Based on the Patent Owner's concession, it is clear that a POSITA would have understood that Cooperman's code resources necessarily define code interrelationships resulting in specific application functionalities once installed on a computer.

282. Therefore, each limitation of element 14.2 is taught by Cooperman.

d) Element 14.3

283. The third element of claim 14 reads: "encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key encoded software code." I refers to this as Element 14.3 throughout this declaration.

284. Element 14.3 is identical to element 12.3, which I address above. For the same reasons I detail above, Cooperman discloses each limitation of element 14.3.

285. And during the original prosecution, Patent Owner confirmed that such a teaching disclosed by Cooperman meets element 14.3. For example, Patent Owner's 05/14/2012 Appeal Brief states that element 14.3 is taught by:

The assembly utility can be supplied with a key generated from a license code generated *for the license in question.*

* * * *

The utility will choose one or several essential code resources, and encode them into one or several data resources using the stegacipher process.

* * * *

¹⁸³ '842 Prosecution History at 519.

The purpose of this scheme is to make a particular licensed copy of an application distinguishable from any other. It is not necessary to distinguish every instance of an application, merely every instance of a license.

* * * *

3) Once it has the license code, it can then generate the proper decoding key to access the essential code resources.¹⁸⁴

286. As I explain with respect to element 12.3, Cooperman includes these same teachings.

287. Moreover, during the original prosecution, Patent Owner stated that “[e]ncoding using a key and an algorithm is known” and that “an interrelationship in software code is necessarily defined by digital data, and digital data can obviously be encoded by an encoding process.”¹⁸⁵

Therefore, a POSITA would have understood that Cooperman’s encoding technique necessarily includes a first license key and an encoding algorithm to form a first license key encoded software code.

e) Element 14.4

288. The fourth element of claim 14 reads: “in which at least one of said software code interrelationships are encoded.” I refer to this as Element 14.4 throughout this declaration.

289. Cooperman discloses element 14.4. Cooperman teaches that its encoding technique results in the encoding of a software code interrelationship. Cooperman, for instance, explains that the software code includes a data resource that specifies where in the code the code resource is encoded:

¹⁸⁴ ’842 Prosecution History at 582 (original claim 62 issued as claim 14); *see also* ’842 Prosecution History at 416 (Patent Owner explaining that element 14.3 is met by teachings corresponding to Cooperman at 10:7-20).

¹⁸⁵ ’842 Prosecution History at 519.

The application must also contain a data resource which specifies in which data resource a particular code resource is encoded. This data resource is created and added at assembly time by the assembly utility.¹⁸⁶

290. And Cooperman further discloses that one of the code resources, such a memory scheduler, is encoded to include the software code interrelationships:

Under the present invention, the application contains a special code resource which knows about all the other code resources in memory. During execution time, this special code resource, called a "memory scheduler," can be called periodically, or at random or pseudo random intervals, at which time it intentionally shuffles the other code resources randomly in memory, so that someone trying to analyze snapshots of memory at various intervals cannot be sure if they are looking at the same code or organization from one "break" to the next.¹⁸⁷

291. During the original prosecution, Patent Owner confirmed that such teachings disclosed by Cooperman meets element 14.4. For instance, Patent Owner's 05/14/2012 Appeal Brief states that element 14.4 is taught by:

The application must also contain a data resource which specifies in which data resource a particular code resource is encoded. This data resource is created and added at assembly time by the assembly utility.

* * * *

Under the present invention, the application contains a special code resource which knows about all the other code resources in memory. During execution time, this special code resource, called a "memory scheduler," can be called periodically, or at random or pseudo random intervals, at which time it intentionally shuffles the other code resources randomly in memory, so that someone trying to analyze snapshots of memory at various intervals cannot be sure if they are looking at the same code or organization from one "break" to the next.¹⁸⁸

Cooperman includes this same teaching.

¹⁸⁶ Cooperman at 11:20-24

¹⁸⁷ Cooperman at 14:35-15:8.

¹⁸⁸ '842 Prosecution History at 577 (original claim 58 was issued as claim 11).

292. Therefore, each limitation of element 14.4 is disclosed by Cooperman. And as I explain above, Cooperman teaches all the other elements of claim 14. Thus, in my opinion, claim 14 is anticipated by Cooperman.

D. Claims 11, 12, 13, and 14 are Anticipated by Hasebe.

1. Hasebe Anticipates Independent Claim 11.

a) Claim 11's Preamble

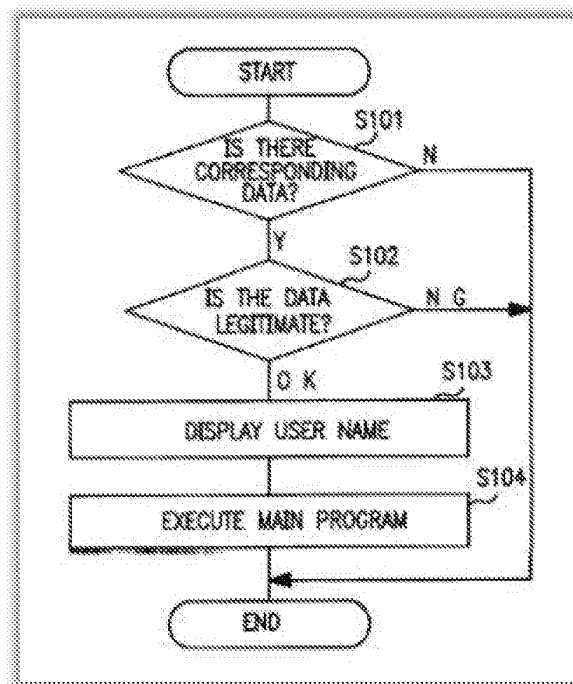
293. The preamble of claim 11 reads: "A method for licensed software use, the method comprising."

294. I understand that a claim's preamble generally does not limit the scope of the claim under the broadest reasonable interpretation applied during reexamination. Nevertheless, Hasebe discloses claim 11's preamble.

295. Hasebe discloses a method of providing software to a user in a non-executable form as well as separate license information.¹⁸⁹ And Hasebe teaches that the user uses the license information to convert the software into an executable form.¹⁹⁰ Hasebe's Figure 6 illustrates this method for licensed software use:

¹⁸⁹ Hasebe at Abstract, 2:47-3:15.

¹⁹⁰ Hasebe at Abstract, 2:47-3:15.



296. Hasebe teaches the steps of this method as follows:

When this software is actuated, as shown in FIG. 6, the CPU, first of all, by checking the contents ID in the license file, decides whether or not data corresponding to the software that is being actuated is present in the license file (step S101). Then, if the corresponding data exists (step S101: Y), the CPU performs a check of the legitimacy of the corresponding data (step S102). In this step, the CPU encodes the information consisting of contents ID and user name stored in the license file using the signature key that is set as data in license display routine 25, and if the result of this encoding agrees with the signature information, decides that the data is legitimate.

If it is legitimate (step S102: OK), the CPU displays the user name which is read from the license file (step S103), and commences operation in accordance with the main program (step S104).

Also, if the corresponding data is not present in the license file (step S101: N) or if the content of the license file is found to be not legitimate (step S102: NG), i.e. if the content of the license file is found to be different from the result of the compilation performed by license file compilation unit 23, the CPU terminates operation without displaying the user name or executing the main program.¹⁹¹

¹⁹¹ Hasebe at 7:61-8:16.

297. As I detail below, Hasebe teaches the remaining steps that comprise the method.

b) *Element 11.1*

298. The first element of claim 11 reads: “loading a software product on a computer, said computer comprising a processor, memory, an input, and an output, so that said computer is programmed to execute said software product.” I refer to this as Element 11.1 throughout this declaration.

299. Hasebe discloses element 11.1. Hasebe teaches a user’s computer having a processor and memory.¹⁹² For instance, Hasebe’s system includes a user terminal with a computer having a “CPU [that operates] when the software that is the subject of the present license system is actuated.”¹⁹³ And Hasebe’s computer includes memory for storing software:

The user terminal comprises a storage unit, a conversion unit, and license file creating unit. In more detail, **the storage unit is employed for storing the license file and software converted to executable form.** The license information, which is generated by the license information generating unit in the management center, is given to the conversion unit. The conversion unit then converts the software to executable form using the license information and installs it in the storage unit. The license file creating unit creates the license file which contains the user identification information contained in the license information, and stores the license file in the storage unit.¹⁹⁴

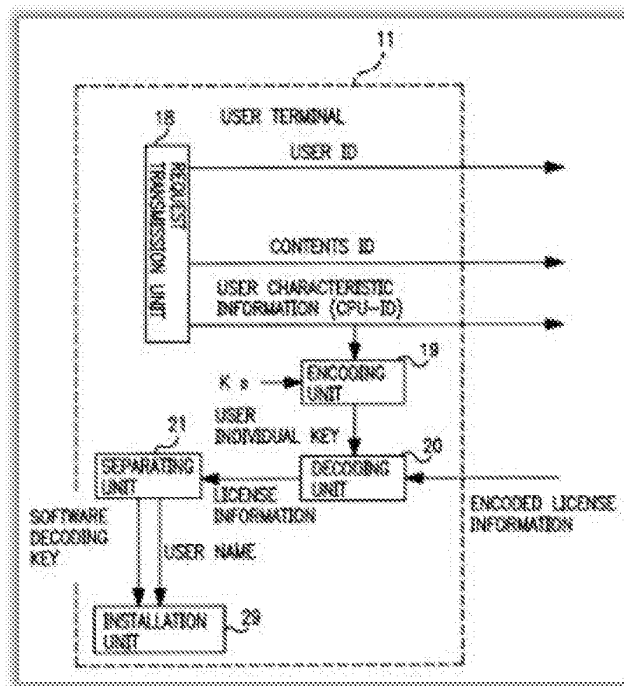
300. Moreover, Hasebe’s computer includes an input (e.g., a keyboard) and output (e.g., a display).¹⁹⁵ As depicted below, Hasebe illustrates the user’s terminal in Figure 7:

¹⁹² Hasebe at 3:62-67, 6:21-25, 7:50-53.

¹⁹³ Hasebe at 7:50-53; *see also* Hasebe at 6:21-25, 7:7-10, 7:61-8:16, 9:6-9.

¹⁹⁴ Hasebe at 2:66-3:10; *see also* Hasebe at 3:62-67 (“convert[ing] the software to executable form using the license information stored in the license file and expands it into memory, and commences operation”), 8:53-59, claims 3, 14.

¹⁹⁵ Hasebe at 7:1-10, 8:47-53, claim 5; *see also* Hasebe at Abstract, 7:54-60, 8:6-21, 8:38-43, 9:33-39, Figs. 6, 9.



301. Hasebe further details loading a software product on the user's computer wherein the computer is programmed to execute the program. For instance, Hasebe discloses that its "software in non-executable form is presented to a user, and license information for converting the software into executable form is informed to the user on condition of payment of a charge, and the software is converted into executable form using this license information."¹⁹⁶ And Hasebe further teaches loading the software onto the user's memory for execution,¹⁹⁷ a process that would undoubtedly require a processor and memory. As shown above, Hasebe's Figure 6 shows executing software loaded onto the user's computer using the license information.

¹⁹⁶ Hasebe at 2:47-54; *see also* Hasebe at claim 1.

¹⁹⁷ Hasebe at 3:28-34, 3:57-67, 8:47-52; *see also* Hasebe at 3:11-15, 8:17-23, Figs. 6, 9.

c) *Element 11.2*

302. The second element of claim 11 reads: “said software product outputting a prompt for input of license information.” I refer to this as Element 11.2 throughout this declaration for convenience.

303. Hasebe discloses element 11.2. Hasebe explains that its software product requests that the user input “license information” into the computer via the keyboard before the product can be used.¹⁹⁸ For instance, Hasebe teaches that the software product prompts the user to input license information: “[N]otification of the contents ID etc to the management center and notification of the encoded license information to the user terminal were performed by another information transmission unit, such as the post... The user terminal is constituted such that installation is effected using encoded license information input from the keyboard.”¹⁹⁹

304. Moreover, Hasebe explains the use of a prompt to enter user ID information which management center 12 uses to generate the encoded-version of the license information:

Request transmission unit 18 commences operation when the keyboard (not shown) of user terminal 11 is operated in accordance with a prescribed procedure that is predetermined as the procedure for request of information for removal of functional restrictions. This request procedure includes keyboard input of the user ID and contents ID; request transmission unit 18 transmits to management center 12 the keyboard input information and the user's characteristic information, which is constituted by the ID of the CPU which is employed in user terminal 11.²⁰⁰

A POSITA would have understood Hasebe’s request for the user ID and contents ID for removal of functional restrictions corresponds to the software outputting a prompt to input license information.

¹⁹⁸ Hasebe at 7:1-10, 8:34-42.

¹⁹⁹ Hasebe at 8:34-42.

²⁰⁰ Hasebe at 7:1-10; *see also* Hasebe at 6:60-7:10, claims 1-2.

305. Thus, each limitation of element 11.2 is disclosed by Hasebe.

d) Element 11.3

306. The third element of claim 11 reads: “said software product using license information entered via said input in response to said prompt in a routine designed to decode a first license code encoded in said software product.” I refer to this as Element 11.3 throughout this declaration for convenience.

307. Hasebe discloses element 11.3. Hasebe teaches that the user’s computer receives “encoded license information” from management center 12:

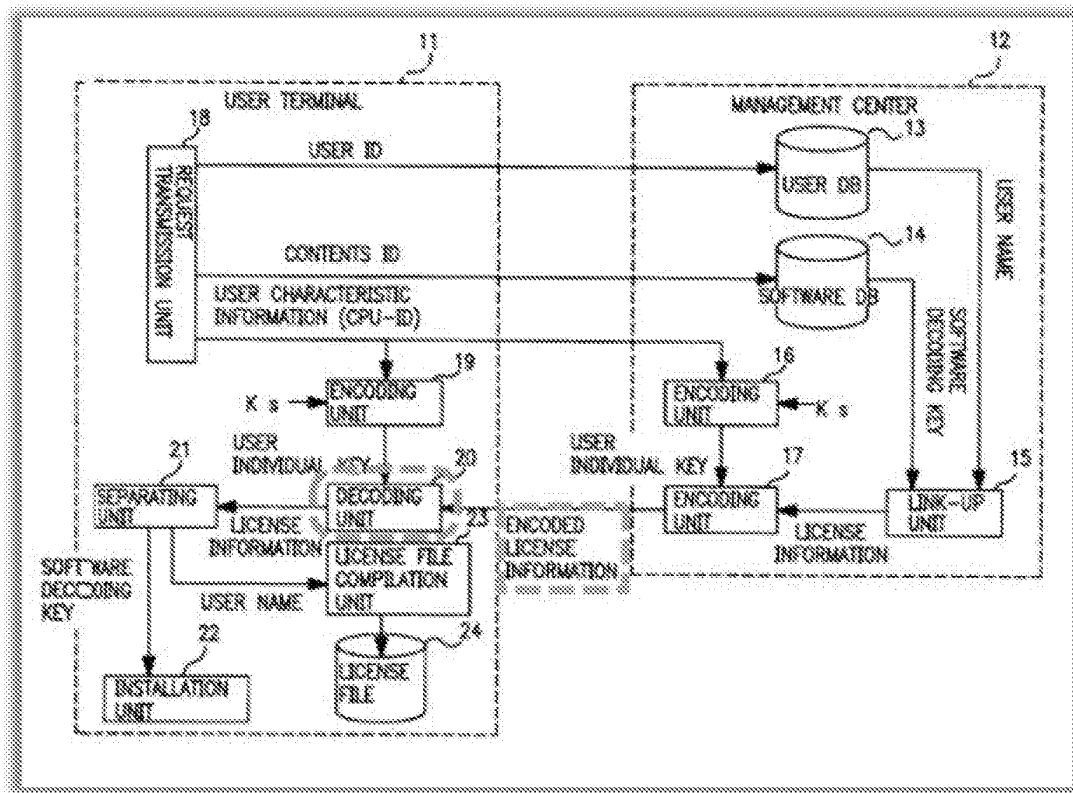
When a request for information for removal of functional restrictions is received from user terminal 11, management center 12 sends to user terminal 11 encoded license information. As a result, after request transmission unit 18 has been operated, user terminal 11 receives encoded license information from management center 12.²⁰¹

308. And Hasebe describes that decoding unit 20 decodes a license code encoded in the software via a decode routine that uses the encoded license information.²⁰² For instance, Hasebe details that its system will “make the software that is presented to the user encoded, and to make the conversion information for decoding the encoded software. Also, ... it is possible to employ information, as license information, which is the result of encoding the conversion information and user identification information, combined in integrated manner.”²⁰³ As illustrated below in annotated Figure 1, Hasebe’s system includes the input of “encoded license information” (dashed box) into the user’s computer 11 which is used to decode the encoded software via decoding unit 20 (dashed oval):

²⁰¹ Hasebe at 7:11-16; *see also* Hasebe at 4:39-58, 6:42-50, Figs. 1, 7.

²⁰² Hasebe at 7:17-31, 9:19-35.

²⁰³ Hasebe at 4:48-58; *see also* Hasebe at 9:29-36.

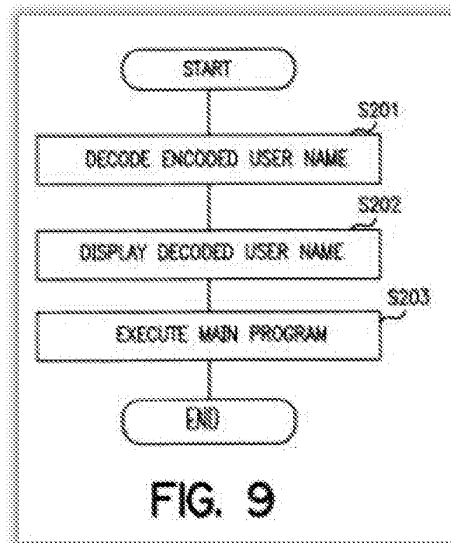
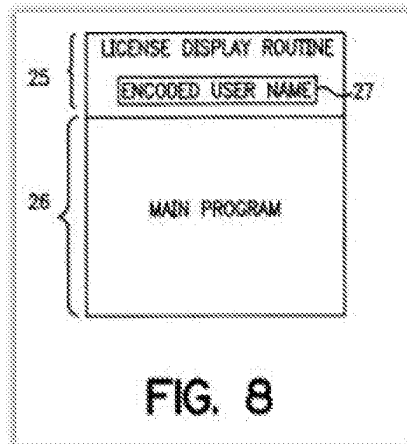


309. Moreover, Hasebe teaches the decoding routine as:

- (a) decoding the license information, which includes the key and user name,
- (b) installing the encoded software using the decoded key,
- (c) writing the user name into the license display routine 25,
- (d) displaying the user name, and
- (e) executing the main portion of software program.²⁰⁴

310. Hasebe's Figure 8 illustrates the license code (routine 25) encoded into the software and main routine 26, and Figure 9 diagrams the decode routine that uses the license information to decode the license code:

²⁰⁴ Hasebe at 9:19-39.



311. A POSITA would have understood that Hasebe's routine 25, with the encoded user name 27, is a license code because it is encoded into the software program and controls the accessibility of the program. And as I explain regarding element 11.2, Hasebe teaches that the user enters license information via an input in response to the prompt.

312. Therefore, each limitation of element 11.3 is disclosed by Hasebe. And as I explain above, Hasebe discloses all the other elements of claim 11. Thus, in my opinion, claim 11 is anticipated by Hasebe.

2. Hasebe Anticipates Independent Claim 12.

a) Claim 12's Preamble

313. The preamble of claim 12 reads: "A method for encoding software code using a computer having a processor and memory, the method comprising"

314. I understand that a claim's preamble generally does not limit the scope of the claim under the broadest reasonable interpretation applied during reexamination. Nevertheless, Hasebe discloses claim 12's preamble.

315. Claim 12 recites both a “computer” and a “computer system.” It is unclear whether those elements refer to the same computing device or separate computing devices. When analyzing claim 12 using the broadest reasonable interpretation, I interpret the “computer” recited in the preamble to be a device separate from the term “computer system.”

316. Hasebe discloses a method for encoding software code using a computer with a processor and memory. Hasebe details that management center 32 generates the software code provided to the user via CD-ROM.²⁰⁵ Alternatively, Hasebe’s software code may be downloaded from the management center.²⁰⁶ And Hasebe describes that the link-up unit 15 of the management center performs “processing” reversed by separating unit 21.²⁰⁷ Thus, A POSITA would have understood that the management center includes a processor and memory to create these CD-ROMs and to provide the downloading capability. Indeed, for as long as computers have been around, it has been standard practice to store the computer code that executes programs—such as the software code used for Hasebe’s invention—in memory. In fact, a POSITA would have had no option but to store Hasebe’s software code in memory, as this is required in computer programming. Similarly, it has been standard practice to execute such programs using a processor in the computer.

317. As I detail below, Hasebe teaches the remaining steps that comprise the method.

b) Element 12.1

318. The first element of claim 12 reads: “storing a software code in said memory.” I refer to this as Element 12.1 throughout this declaration.

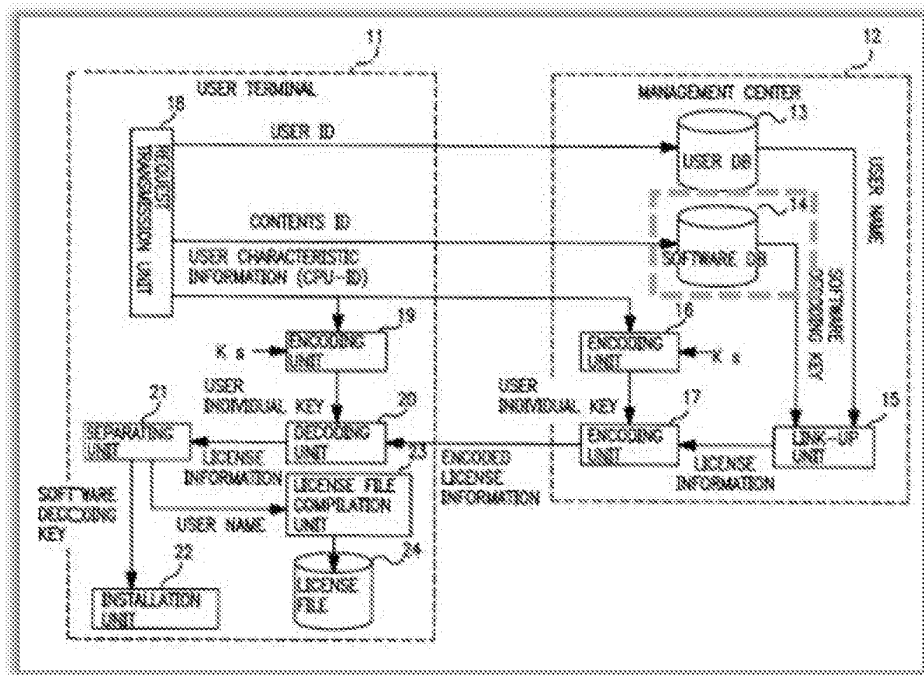
²⁰⁵ Hasebe at 1:9-14, 6:9-13, 9:22-26.

²⁰⁶ Hasebe at 9:60-64.

²⁰⁷ Hasebe at 7:23-26, Fig. 1.

319. Hasebe discloses element 12.1. As explained with respect to claim 12's preamble, Hasebe's management center 32 either generates a CD-ROM containing the software code or provides downloadable versions of the software code.²⁰⁸ A POSITA thus would have understood that Hasebe's management center stores the software code in its memory for CD-ROM generation or user downloading. And as I discuss regarding claim 12's preamble, it has been standard practice to store computer code—such as Hasebe's software code—in memory. In fact, a POSITA would have had no option but to store this software code in memory, as this is required in computer programming.

320. As depicted in Hasebe's Figure 1, as annotated below, management center 14 includes a software database 14 (dashed box) capable of software storage:



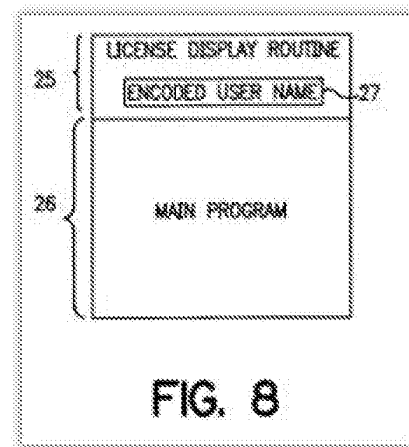
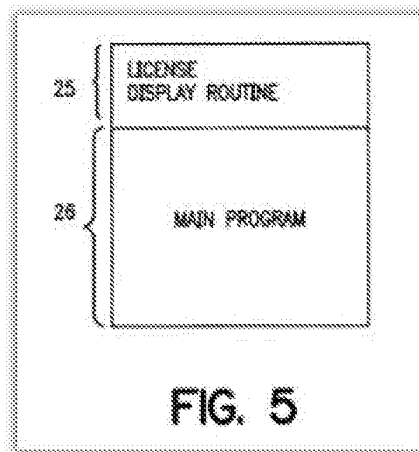
321. Therefore, each limitation of element 12.1 is disclosed by Hasebe.

²⁰⁸ Hasebe at 6:9-13, 9:22-26, 9:60-64.

c) *Element 12.2*

322. The second element of claim 12 reads: “wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system.” I refer to this as Element 12.2 throughout this declaration.

323. Hasebe discloses element 12.2. Hasebe teaches that its software code provides certain functionality to the user upon determining that a proper license key has been entered. Hasebe specifies that its software code includes a license display routine 25.²⁰⁹ And Hasebe explains that routine 25 determines whether the user’s license information is legitimate and, if so, permits access to the main program routine 26.²¹⁰ In doing so, routine 25 refers to one or more code resources. For instance, Hasebe details: “In the main program there are defined the operating procedures relating to the proper functions of this software, in license display routine 25, there is defined the content to be executed prior to execution of main program 26.”²¹¹ Hasebe illustrates routines 25 and 26 of the software code in Figures 5 and 8:



²⁰⁹ Hasebe at 7:55-8:9, 9:25-35, Figs. 5, 8.

²¹⁰ Hasebe at 7:65-8:9.

²¹¹ Hasebe at 7:55-60.

324. The '842 Patent refers to sub-objects, a memory scheduler, and data as examples of code resources.²¹² Hasebe's routine 25 consists of software code that controls access to the underlying functionality of the software's main program, or sub-objects. In this additional way, a POSITA would have understood that Hasebe's routine 25 contains a first code resource.

325. Moreover, Hasebe's software code provides underlying functionalities when installed on the user's computer system (terminal 31). Hasebe, for example, discloses that the code's routine 25 provides access to the main program module 26 upon verification of the user's license information.²¹³

326. Thus, each limitation of element 12.2 is disclosed by Hasebe.

d) Element 12.3

327. The third element of claim 12 reads: "encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key encoded software code." I refer to this as Element 12.3 throughout this declaration.

328. Hasebe discloses element 12.3. As I discuss with respect to element 12.1, Hasebe's management center 32 provides the user the software code via CD-ROM or download from the seller.²¹⁴ Hasebe teaches that the management center 32 encodes the software code:

[I]t is also possible to make the software that is presented to the user encoded, and to make the conversion information for decoding the encoded software. Also, it is possible to employ, in such a licensee notification system, license information containing the user identification information in a form that cannot be separated without special information. For example, it is possible to employ information, as license information, which is the result of encoding the conversion information and user identification information, combined in integrated manner.²¹⁵

²¹² '842 Patent at 11:55-65, 15:36-42.

²¹³ Hasebe at 7:65-8:9, 9:20-36.

²¹⁴ Hasebe at 6:9-13, 9:22-26, 9:60-64.

²¹⁵ Hasebe at 4:48-58; *see also* Hasebe at 7:32-38, 9:22-26.

It is also possible to constitute the system such that, instead of the user name and signature information, information representing the user name in encoded form is stored in the license file, and, when the installed software is executed, the information in the license file is decoded by the software and displayed.²¹⁶

329. With respect to the code illustrated in Figure 9, Hasebe discloses that the customer's computer system "effects installation by decoding the software in the CD ROM using the software decoding key, and generates the user name in encoded form by encoding the user name."²¹⁷

330. Further, Hasebe teaches its encoding technique uses a license key and an encoding algorithm. For instance, Hasebe describes its system includes: "**a DES (data encryption standard) algorithm** [] employed for encoding and decoding."²¹⁸ And Hasebe details that the system uses a license key to encode the software code: "generat[ing] license information including user identification information encoded with **a characteristic key of the software**."²¹⁹

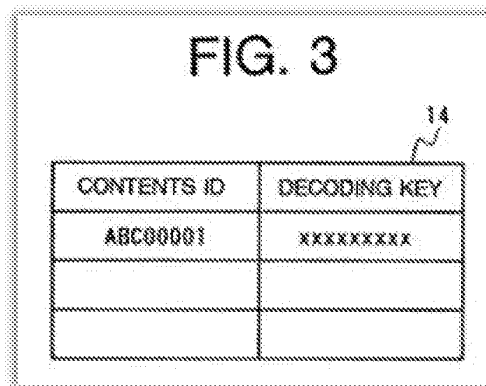
Figure 3, for example, illustrates a license key in the management center's software database 14 used to encode the software:

²¹⁶ Hasebe at 8:47-53.

²¹⁷ Hasebe at 9:22-26.

²¹⁸ Hasebe at 6:48-50.

²¹⁹ Hasebe at 4:40-43; *see also* Hasebe at 6:33-47, 7:33-38, 9:19-26.



Thus, a POSITA would have understood that Hasebe's encoded software code, utilizing the encoded license information, creates the claimed "first license key encoded software code."

331. Moreover, during the original prosecution, Patent Owner stated that "[e]ncoding using a key and an algorithm is known."²²⁰ Therefore, a POSITA would have understood that Hasebe's encoding technique necessarily includes a first license key and an encoding algorithm to form a first license key encoded software code.

332. Thus, each limitation of element 12.3 is disclosed by Hasebe.

e) Element 12.4

333. The fourth element of claim 12 reads: "wherein, when installed on a computer system, said first license key encoded software code will provide said specified underlying functionality only after receipt of said first license key." I refer to this as Element 12.4 throughout this declaration.

334. Hasebe discloses element 12.4. Hasebe teaches the installation of the software code upon verification of the first license key by the user's computer.²²¹ For instance, Hasebe describes the

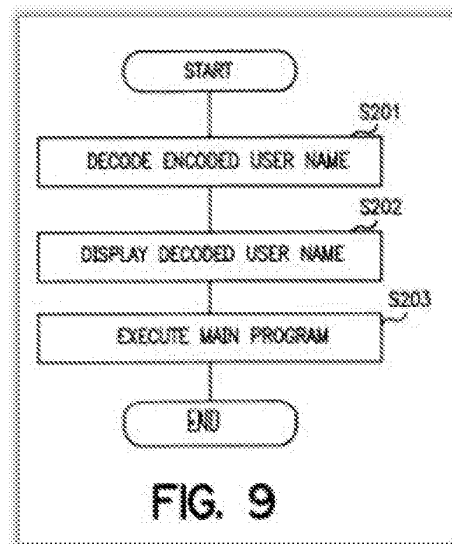
²²⁰ 842 Prosecution History at 519.

²²¹ Hasebe at 3:5-15, 3:30-38, 9:19-39; *see also* Hasebe at 7:32-38, 8:47-53.

software code will provide access to specified underlying functionality of the code contained in main program routine 26 only after receipt of the first license key in license display routine 25:

- (a) decoding the license information, which includes the key and user name,
- (b) installing the encoded software using the decoded key,
- (c) writing the user name into the license display routine 25,
- (d) displaying the user name, and
- (e) executing the main portion routine 26 of software program.²²²

335. And Hasebe's Figure 9 illustrates the user's computer providing the underlying functionality of the main program routine 26 after the receipt and decoding of the first license key:



A POSITA would have understood that Hasebe's main program routine 26 includes specified underlying functionality of the first license key encoded software code accessible via confirmation of the encoded license key.

²²² Hasebe at 9:19-39.

336. Therefore, each limitation of element 12.4 is disclosed by Hasebe. And as I explain above, Hasebe teaches all the other elements of claim 12. Thus, in my opinion, claim 12 is anticipated by Hasebe.

3. Hasebe Anticipates Independent Claim 13.

a) Claim 13's Preamble

337. The preamble of claim 13 reads: "A method for encoding software code using a computer having a processor and memory, comprising."

338. I understand that a claim's preamble generally does not limit the scope of the claim under the broadest reasonable interpretation applied during reexamination. Nevertheless, Hasebe discloses claim 13's preamble.

339. Claim 13's preamble appears to be the same as claim 12's preamble. And as I explain above, Hasebe teaches a method for encoding software using a computer with a processor and memory. Thus, Hasebe discloses this preamble.

b) Element 13.1

340. The first element of claim 13 reads: "storing a software code in said memory." I refer to this as Element 13.1 throughout this declaration.

341. Element 13.1 is identical to element 12.1, which I discuss above. For the same reasons as I explain above, Hasebe discloses each limitation of element 13.1.

c) Element 13.2

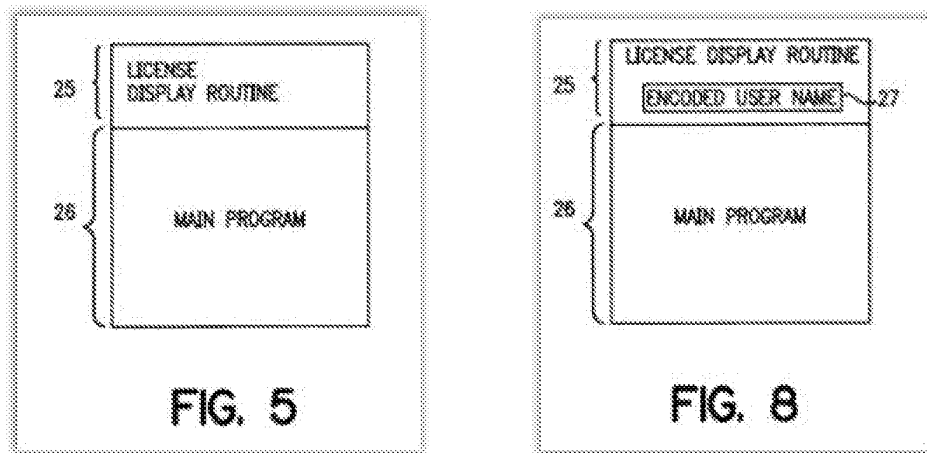
342. The second element of claim 13 reads: "wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system." I refer to this as Element 13.2 throughout this declaration.

343. Element 13.2 is identical to element 12.2, which I discuss above. For the same reasons as I explain above, Hasebe discloses each limitation of element 13.2.

d) *Element 13.3*

344. The third element of claim 13 reads: “modifying, by said computer, using a first license key and an encoding algorithm, said software code, to form a modified software code, and wherein said modifying comprises encoding said first code resource to form an encoded first code resource.” I refer to this as Element 13.3 throughout this declaration.

345. Hasebe discloses element 13.3. As described with respect to element 12.2, Hasebe’s system includes multiple code resources (e.g., license display routine 25) used to access software functionality.²²³ Hasebe illustrates routine 25 and main program routine 26 of the software code in Figures 5 and 8:



346. And as taught with respect to element 12.3, Hasebe’s computer²²⁴ in management center 12 modifies the software code to form an encoded first code resource.²²⁵ For instance, Hasebe’s

²²³ Hasebe at 7:55-8:9, 9:25-35, Figs. 5, 8.

²²⁴ Hasebe at 6:21-24.

²²⁵ Hasebe at 4:48-58, 8:47-53; *see also* Hasebe at 7:32-38, 9:22-26.

software code is modified to include routine 25 used to verify the user's license information and permit execution of the software code.²²⁶

347. Hasebe specifies its code modification uses a license key and an encoding algorithm, as described with respect to element 12.3.²²⁷ Moreover, during the original prosecution, Patent Owner stated that "[e]ncoding using a key and an algorithm is known."²²⁸ Thus, a POSITA would have understood that Hasebe's encoding technique necessarily includes a first license key and an encoding algorithm to form an encoded first code resource.

348. Thus, each limitation of element 13.3 is disclosed by Hasebe.

e) Element 13.4

349. The fourth element of claim 13 reads: "wherein said modified software code comprises said encoded first code resource, and a decode resource for decoding said encoded first code resource." I refer to this as Element 13.4 throughout this declaration.

350. Hasebe discloses element 13.4. As described with respect to element 13.3, Hasebe's modified software code includes the encoded first code resource. And Hasebe teaches that user terminal 11 includes decode unit 20 and separating unit 21 to produce the decoding key for the relevant software code.²²⁹ Hasebe's user terminal sends the decoding key to the software installation unit (Fig. 1's unit 22 or Fig. 7's unit 29), and "[i]nstallation unit 29 effects installation by decoding the software in the CD ROM using the software decoding key, and generates the user name in encoded form by encoding the user name."²³⁰ As depicted below in

²²⁶ Hasebe at 4:48-58, 8:47-53.

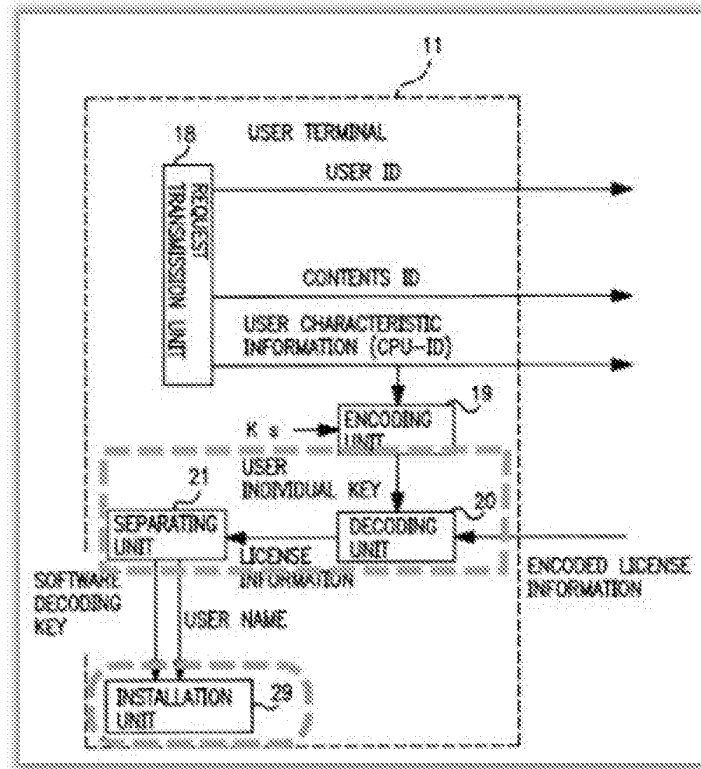
²²⁷ Hasebe at 6:48-50, 4:40-43, Fig. 3; *see also* Hasebe at 6:33-47, 7:33-38, 9:19-26.

²²⁸ '842 Prosecution History at 519.

²²⁹ Hasebe at 7:17-31.

²³⁰ Hasebe at 7:27-39, 9:22-26.

annotated Figure 7, Hasebe's user terminal 11 includes a decode resource including the separating and decoding units 20, 21 (dashed box) and installation unit 22 (dashed oval) to decode the encoded code resource for software execution:



Therefore, Hasebe teaches a decoding resource for decoding the encoded first code resource.

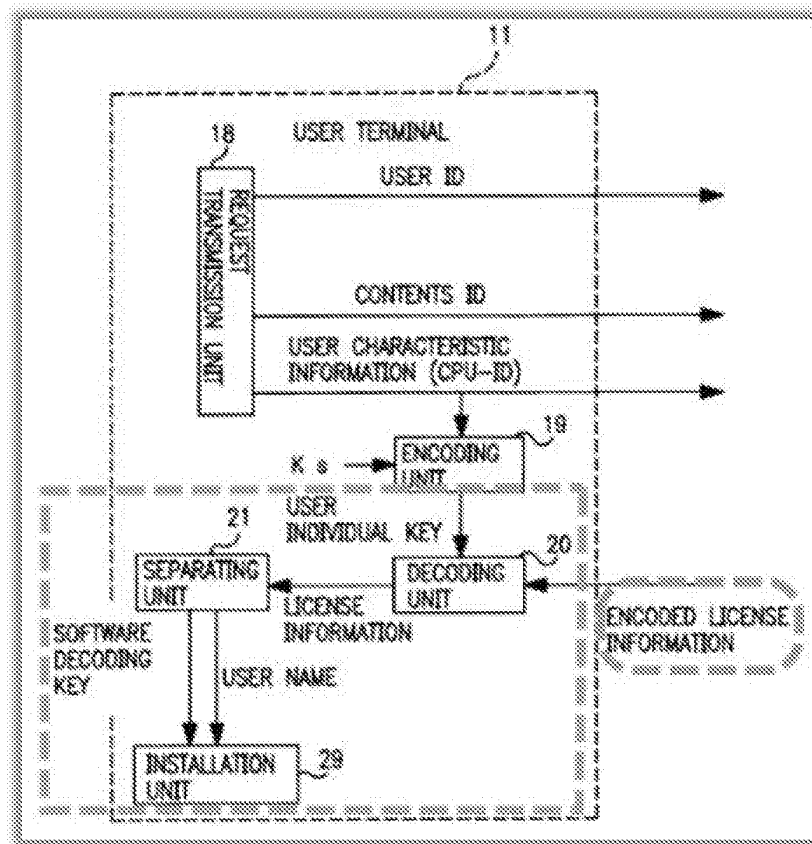
351. Thus, each limitation of element 13.4 is disclosed by Hasebe.

f) Element 13.5

352. The last element of claim 13 reads: "wherein said decode resource is configured to decode said encoded first code resource upon receipt of said first license key." I refer to this as Element 13.5 throughout this declaration.

353. Hasebe discloses element 13.5. As described with respect to element 12.3, Hasebe describes that the system uses a license key to encode the software code: "generat[ing] license

information including user identification information encoded with a **characteristic key of the software.**"²³¹ And Hasebe specifies that its decode resource decodes the encoded first code resource upon receipt of the license key. For instance, Hasebe details that the user terminal receives the encoded license information at decoding unit 20, decodes the information to produce the decoding key, and decodes the encode first code resource (routine 25) "by decoding the software in the CD ROM using the software decoding key."²³² Figure 7, as annotated below, shows the decode resource (dashed box) receiving the first license key (dashed oval) to decode the encoded software—including the encoded first code resource:



²³¹ Hasebe at 4:40-43; *see also* Hasebe at 6:33-47, 7:33-38, 9:19-26, Fig. 3.

²³² Hasebe at 7:27-39, 9:22-26.

354. Therefore, each limitation of element 13.5 is disclosed by Hasebe. And as I explain above, Hasebe teaches all the other elements of claim 13. Thus, in my opinion, claim 13 is anticipated by Hasebe.

4. Hasebe Anticipates Independent Claim 14.

a) Claim 14's Preamble

355. The preamble of claim 14 reads: "A method for encoding software code using a computer having a processor and memory, comprising."

356. I understand that a claim's preamble generally does not limit the scope of the claim under the broadest reasonable interpretation applied during reexamination. Nevertheless, Hasebe discloses claim 14's preamble.

357. Claim 14's preamble appears to be the same as each of claim 12 and 13's preamble. As I explain above, Hasebe teaches a method for encoding software using a computer with a processor and memory. Thus, Hasebe discloses this preamble.

b) Element 14.1

358. The first element of claim 14 reads: "storing a software code in said memory." I refer to this as Element 14.1 throughout this declaration.

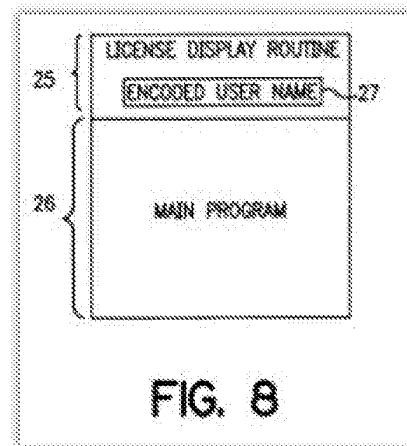
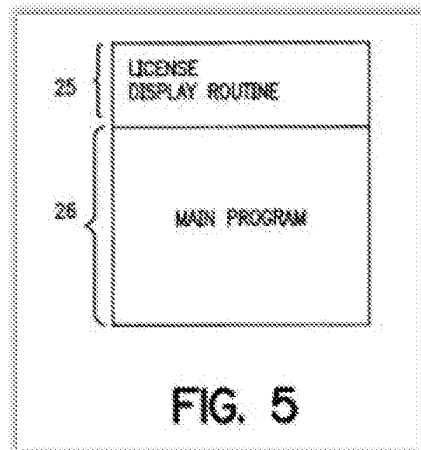
359. Element 14.1 is identical to element 12.1, which I discuss above. For the same reasons as I explain above, Hasebe discloses each limitation of element 14.1.

c) Element 14.2

360. The second element of claim 14 reads: "wherein said software code defines software code interrelationships between code resources that result in a specified underlying functionality when installed on a computer system." I refer to this as Element 14.2 throughout this declaration.

361. Hasebe discloses element 14.2. Hasebe teaches that its software code provides certain functionality to the user upon determining that a proper license key has been entered. In doing

so, Hasebe's code includes interrelationships between code resources, such as interrelating license display routine 25 and main program routine 26.²³³ For instance, Hasebe explains that its software code includes routine 25 which permits access to the main program routine 26 upon validation of user's license information.²³⁴ Hasebe discloses: "In the main program there are defined the operating procedures relating to the proper functions of this software; in license display routine 25, there is defined the content to be executed prior to execution of main program 26."²³⁵ Hasebe illustrates routines 25 and 26 of the software code in Figures 5 and 8:



362. The '842 Patent refers to sub-objects and a memory scheduler as examples of code resources.²³⁶ Hasebe's routine 25 contains a sub-object of the software code because it controls access to the underlying functionality of the software's main program. And Hasebe teaches routine 25 "directly rewrite[es]" the software code when the software code is decoded.²³⁷ In this

²³³ Hasebe at 7:55-8:9, Figs. 5, 8, 9.

²³⁴ Hasebe at 7:65-8:9.

²³⁵ Hasebe at 7:55-60.

²³⁶ '842 Patent at 11:55-65, 15:36-42.

²³⁷ Hasebe at 5:10-32, 9:22-39.

additional way, a POSITA would have understood that Hasebe's routines 25 and 26 contain code resources and that the software code defines software code interrelationships between the code resources. And a POSITA would have understood that the interrelationship between Hasebe's routines 25 and 26 result in a specified underlying functionality upon code installation.

363. Moreover, during the original prosecution, Patent Owner stated that "interrelationships between code resource are not that which is novel."²³⁸ The Patent Owner continues by conceding:

What the examiner has implied by alleging that the "specification ... fails to teach or mention 'software code interrelationships'" is that software code interrelationships were somehow unknown in the art, which clearly is not the case. **As admitted, in the specification at the beginning of paragraph [0051], an "application" comprises "sub-objects" whose "order in the computer memory is of vital importance" in order to perform an intended function.** And as admitted further in paragraph [0051], "When a program is compiled, then, it consists of a collection of these sub-objects, whose exact order or arrangement in memory is not important, so long as any sub-object which uses another sub-object knows where in memory it can be found." **Paragraph [0051] of course refers to conventional applications. Accordingly, that is admittedly a discussion of what is already know by one skilled in the art.** Accordingly, the examiner's statement that the specification lacks written description support for "software code interrelationships" is inconsistent with the fact that such **interrelationships were explained in paragraphs [0051] and [0052] as a fundamental basis of pre-existing modem computer programs.**²³⁹

364. Based on the Patent Owner's concession, it is clear that a POSITA would have understood that Hasebe's code resources necessarily define code interrelationships resulting in specific underlying functionality once installed on a computer.

365. Thus, each limitation of element 14.2 is disclosed by Hasebe.

²³⁸ '842 Prosecution History at 519.

²³⁹ '842 Prosecution History at 519.

d) Element 14.3

366. The third element of claim 13 reads: “encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key encoded software code.” I refers to this as Element 14.3 throughout this declaration.

367. Element 14.3 is identical to element 12.3, which I address above. For the same reasons I explain above, Hasebe discloses each limitation of element 14.3.

368. Moreover, during the original prosecution, Patent Owner stated that “[e]ncoding using a key and an algorithm is known” and that “an interrelationship in software code is necessarily defined by digital data, and digital data can obviously be encoded by an encoding process.”²⁴⁰ Thus, a POSITA would have understood that Hasebe’s encoding technique necessarily includes a first license key and an encoding algorithm to form a first license key encoded software code.

e) Element 14.4

369. The fourth element of claim 14 reads: “in which at least one of said software code interrelationships are encoded.” I refer to this as Element 14.4 throughout this declaration.

370. Hasebe discloses element 14.4. As described with respect to element 14.2, Hasebe teaches that its software code defines code interrelationships between code resources and routine control certain underlying software functionality. Hasebe further explains that its software code is encoded:

[I]t is also possible to make the software that is presented to the user encoded, and to make the conversion information for decoding the encoded software. Also, it is possible to employ, in such a licensee notification system, license information containing the user identification information in a form that cannot be separated without special information. For example, it is possible to employ information, as

²⁴⁰ ‘842 Prosecution History at 519.

license information, which is the result of encoding the conversion information and user identification information, combined in integrated manner.²⁴¹

It is also possible to constitute the system such that, instead of the user name and signature information, information representing the user name in encoded form is stored in the license file, and, when the installed software is executed, the information in the license file is decoded by the software and displayed.²⁴²

371. And Hasebe teaches that the software code includes the code interrelationships between routines 25 and 26, all of which would be encoded as part of the software code.²⁴³

372. Therefore, each limitation of element 14.4 is disclosed by Hasebe. And as I explain above, Hasebe teaches all the other elements of claim 14. Thus, in my opinion, claim 14 is anticipated by Hasebe.

I declare that all statements made herein of my own knowledge are true, and that all statements made on information and belief are believed to be true; and that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code.

Dated: May 15, 2018

By: 
Claudio T. Silva

²⁴¹ Hasebe at 4:48-58; *see also* Hasebe at 7:32-38, 9:22-26.

²⁴² Hasebe at 8:47-53.

²⁴³ Hasebe at 7:55-8:9, Figs. 5, 8, 9.

Exhibit 10

Cláudio T. Silva

Professor of Computer Science and Engineering and Data Science

Tandon School of Engineering
New York University
Six MetroTech Center
Brooklyn, NY 11201
csilva@nyu.edu

phone: (646) 997-4093

<http://engineering.nyu.edu/people/claudio-silva>

Professional Preparation

- Post-doc, Applied Mathematics and Statistics 1996-7
State University of New York at Stony Brook
Concentration Area: Computational Geometry
Mentor: Distinguished Professor Joseph S.B. Mitchell
- Ph.D., Computer Science December 1996
State University of New York at Stony Brook
Dissertation Title: "Parallel Volume Rendering of Irregular Grids"
Advisor: Distinguished Professor Arie E. Kaufman
- M.S., Computer Science May 1993
State University of New York at Stony Brook
- B.S., Mathematics July 1990
Universidade Federal do Ceará (Brazil)

Professional Experience

- Computer Science & Engineering, School of Engineering, New York University
 - Professor (July 2011–)
 - Research Professor (October 2010–June 2011)
 - Engineer-in-Residence, Incubator (December 2012–)
- Center for Data Science, New York University
 - Interim Director, (September 2016–August 2017)
 - Associated Faculty, (September 2013–)
- Center for Urban Science and Progress, New York University
 - Head of Disciplines (September 2012–August 2015)
- Department of Computer Science, Courant Institute of Mathematical Sciences, NYU
 - Affiliated Faculty (December 2011–)
- Major League Baseball (MLB) Advanced Media

- Consultant (February 2012–December 2017)
- Fisch Sigler LLP
 - Expert Witness (June 2017–August 2017)
- Kirkland & Ellis LLP
 - Expert Witness (February 2016–August 2016)
 - Expert Witness (September 2017–December 2017)
- Modelo, Inc.
 - Co-founder (2011)
- School of Computing, University of Utah
 - Adjunct Professor (July 2011–)
 - Professor (July 2010–June 2011)
 - Associate Professor (October 2003–June 2010)
- Guest Professor, Linköping University, Sweden, (January 2010–December 2012)
- Scientific Computing and Imaging (SCI) Institute, University of Utah
 - Associate Director (January 2008–May 2009)
 - Faculty Member (October 2003–June 2011)
- Visiting Researcher, ETH Zurich, (November 2010)
- VisTrails, Inc. (2007) [University of Utah startup company: www.vistrails.com]
 - Co-founder
 - Chief Scientist
- Participating Guest Researcher (April 2003–), Lawrence Livermore National Laboratory.
- Faculty Scholar (January 2003–March 2003), Lawrence Livermore National Laboratory.
- Associate Professor (September 2002–April 2006; on leave starting October 2003), Department of Computer Science & Engineering, OGI School of Science & Engineering, Oregon Health & Science University.
- Information Visualization Research Department, AT&T Labs-Research.
 - Principal Member of Technical Staff (April 2002–September 2002)
 - Senior Member of Technical Staff (July 1999–April 2002)
- Adjunct Assistant Professor, Department of Applied Mathematics and Statistics, State University of New York at Stony Brook, July 1998–July 2000.
- Research Staff Member, Visual and Geometric Computing, IBM T. J. Watson Research Center, December 1997–July 1999.

- Research Associate, Computational Geometry Lab (Joseph S.B. Mitchell, Director). Department of Applied Mathematics and Statistics, State University of New York at Stony Brook, September 1996–December 1997.
- Researcher, Visualization Group, Sandia National Laboratories, May 1995–December 1997.
- Teaching and Research Assistant, Visualization Lab (Arie Kaufman, Director). Department of Computer Science, State University of New York at Stony Brook, 1991–1995.
- Summer Intern, Brookhaven National Laboratories, 1992.
- Summer Intern, Philips Laboratories, 1991.

Honors, Distinctions, and Achievements

- 2018 Technology & Engineering Emmy Award from the National Academy of Television Arts & Sciences (NATAS) for MLB Advanced Media’s Statcast player tracking system
- Best demo honorable mention award – SIBGRAPI 2017
- Best demo honorable mention award – SIGMOD 2017
- (student award) 2017 Henning Biermann Award from the Courant Institute
advisee: Dr. Bowen Yu (2017)
- Best paper honorable mention award – IEEE Data Science and Advanced Analytics, 2016
- Elected Chair of IEEE Technical Committee on Visualization and Computer Graphics (2015–2017)
- (student award) Pearl Brownstein Doctoral Research Award (for PhD thesis),
advisee: Dr. Nivan Ferreira (2015)
- (student award) Courant’s Matthew Smosna Prize for excellence in computer science (for MS thesis),
advisee: Yunzhe Jia (2015)
- Alpha Award for Best Analytics Innovation/Technology for MLB Advanced Media’s Statcast player tracking system, 2015 MIT Sloan Sports Analytics Conference
- 2014 IEEE VGTC Visualization Technical Achievement Award “in recognition of seminal advances in geometric computing for visualization and for contributions to the development of the VisTrails data exploration system.”
- Outstanding Partnership, Federal Laboratory Consortium for Technology Transfer for Ultrascale Visualization Climate Data Analysis Tools (UV-CDAT), 2014
- (student award) VPG Best Dissertation Finalist,
advisee: Dr. Tiago Etienne (2013)
- IBM Faculty Award, 2013.
- Best paper honourable mention award – EuroVis 2013
- 2013 IEEE Fellow “for contributions to geometric computing and visualization.”
- Best paper award – SIBGRAPI 2012

- Best panel award – IEEE VisWeek 2011
- Best paper award – 2nd prize, EuroVis 2011
- Best paper award, ACM Eurographics Symposium on Parallel Graphics and Visualization 2011.
- Finalist, Executable Paper Grand Challenge, 2011.
- 2011 IEEE Computer Society, Certificate of Appreciation “for outstanding service and performance as Co-Chairman of VisWeek 2010.”
- Best paper award, EUROGRAPHICS 2010 Educator Program.
- Best poster award, 24th Brazilian Symposium On Databases (SBBDB 2009)
- 2009 Utah Innovation Awards, VisTrails Provenance Plugin for Autodesk Maya.
- IEEE Senior Member (since 2008).
- Best paper award, IEEE Shape Modeling International 2008.
- Best paper award, IEEE Visualization 2007.
- Best paper finalist, IEEE Shape Modeling International 2007.
- Dean’s Teaching Commendation, Spring 2007.
- IBM Faculty Award, 2007.
- IBM Faculty Award, 2006.
- IBM Faculty Award, 2005.
- Best paper finalist, IEEE Visualization 2001.
- Best paper finalist, IEEE Visualization 1999.
- IBM First Plateau Invention Award, 1999.
- IBM Research Division “accomplishment list” for MPEG-4 3D Model Coding, 1998.
- National Science Foundation Post-Doctoral CISE Associateship Award, 1996–1997.
- Best paper finalist, ACM/IEEE Volume Visualization 1996.
- Doctoral Fellowship – Brazilian Research Council (CNPq – Brazil), 1991–1995.
- 1st place, Entrance exam, Mathematics, Federal University of Ceara, Brazil.

Media Coverage (partial)

- New York Times (online): Mapping the Shadows of New York City: Every Building, Every Block; <https://goo.gl/dToiem>
- New York Times (print and online): To Create a Quieter City, Theyre Recording the Sounds of New York; <https://goo.gl/oimnsK>
- Economist (print and online): Listen to the music of the traffic in the city; <https://goo.gl/jIfvc2>
- Economist (print and online): Every step they take; <https://goo.gl/pEZNGj>
- Vice Sports, Future of the game: The era of wearables (video); <http://goo.gl/D6XGRC>
- Vice Sports, Future of the game: Baseball's latest statistical revolution (video); <http://goo.gl/N4f3sh>
- NetworkWorld, How the cloud gives Major League Baseball a new world of stats; <http://goo.gl/1uJfk0>
- Interview at archspeech (in Russian); <http://goo.gl/sBquL0>
- Claudio Silva: The future of the interdisciplinary approach, capable of solving complex problems of cities (in Russian); <http://goo.gl/vngUOI>
- Do not spoil the unsuccessful city buildings (in Russian); <http://goo.gl/zolzs8>
- (ABC News, USA Today, Sun Times, ...), Data Deluge: MLB Rolls out Statcast Analytics on Tuesday; <http://goo.gl/m0HXEm>
- PR Newswire, McGraw-Hill Education Takes Important Step in Open Technology, Enabling Educators to Build Personalized Learning Experiences; <http://goo.gl/wak1JU>
- MLB News, Statcast wins prestigious Alpha Award for innovation; <http://goo.gl/u0745S>
- Newsweek, Can baseball get more interesting to watch with Big Data?; <http://goo.gl/vwK5jm>
- Wall Street Journal, Billy Beane Expects Big Things from MLB's Big Data Play; <http://goo.gl/mGrBj9>
- MLB.com, Statcast interview (video); <http://goo.gl/TVQ9Hy>
- MLB News, MLBAM introduces new way to analyze every play; <http://goo.gl/DW52zW>

Publications

Google Scholar h-index: 57; total citations: 14,823 (date: 2/12/18)

Book (1)

- [1] *An Introduction to Verification of Visualization Techniques*, T. Etienne, R. Kirby and C. Silva, Morgan & Claypool Publishers, 2015.

Journal Publications (115)

- [2] *GPU Rasterization for Real-Time Spatial Aggregation over Arbitrary Polygons*, E. Tzirita Zacharitou, H. Doraiswamy, A. Ailamaki, C. Silva, and J. Freire, PVLDB 2017/2018, to appear.
- [3] *Wavelet-based Visual Analysis of Dynamic Networks*, A. Dal Col, P. Valdivia, F. Petronetto, F. Dias, C. Silva, and L.G. Nonato, IEEE Transactions on Visualization and Computer Graphics, to appear.
- [4] *ARIES: Enabling Visual Exploration and Organization of Art Image Collections*, L. Crissaff, L. Ruby, S. Deutch, L. DuBois, J.-D. Fekete, J. Freire, and C. Silva, IEEE Computer Graphics and Applications, accepted.
- [5] *TopKube: A Rank-Aware Data Cube for Real-Time Exploration of Spatiotemporal Data*, F. Miranda, L. Lins, J.T. Klosowski, C. Silva, IEEE Transactions on Visualization and Computer Graphics, to appear.
- [6] *TopoAngler: Interactive Topology-based Extraction of Fishes*, A. Bock, H. Doraiswamy, A. Summers, and C. Silva, IEEE Transactions on Visualization and Computer Graphics (Proceedings of SCIVIS 2017), 24(1):812–821, 2018.
- [7] *Wavelet-Based Visual Analysis for Data Exploration*, A. Dal Col, P. Valdivia, F. Petronetto, F. Dias, C. Silva, L.G. Nonato, Computing in Science & Engineering 19 (5), 85-91, 2017.
- [8] *Mocap: Large-scale inference of transcription factor binding sites from chromatin accessibility*, X. Chen, B. Yu, N. Carriero, C. Silva, and R. Bonneau, Nucleic Acids Research, 45(8):4315–4329, 2017.
- [9] *Dynamic Scene Graph: Enabling Scaling, Positioning, and Navigation in the Universe*, E. Axelsson, A. Bock, J. Costa, C. Emmart, C. Silva, and A. Ynnerman, Computer Graphics Forum (Proceedings of EuroVis 2017), 36(3):459–468, 2017.
- [10] *STaRS: Simulating Taxi Ride Sharing at Scale*, M. Ota, H. Vo, C. Silva, and J. Freire, IEEE Transactions on Big Data, 3(3):349–361, 2017.
- [11] *A Survey of Surface Reconstruction from Point Clouds*, M. Berger, A. Tagliasacchi, L. Seversky, P. Alliez, G. Guennebaud, J. Levine, A. Sharf and C. Silva, Computer Graphics Forum, 36(1):301–329, 2017.
- [12] *Urban Pulse: Capturing the Rhythm of Cities*, F. Miranda, H. Doraiswamy, M. Lage, K. Zhao, B. Gonçalves, L. Wilson, M. Hsieh, and C. Silva, IEEE Transactions on Visualization and Computer Graphics (Proceedings of SCIVIS 2016), 23(1): 791-800 (2017).
- [13] *VisFlow - Web-based Visualization Framework for Tabular Data with a Subset Flow Model*, B. Yu and C. Silva, IEEE Transactions on Visualization and Computer Graphics (Proceedings of VAST 2016), 23(1): 251-260 (2017).
- [14] *Bijjective Maps from Simplicial Foliations*, M. Campen, C. Silva, and D. Zorin, ACM Transactions on Graphics (SIGGRAPH 2016), 35(4):74, 2016.
- [15] *Statcast Dashboard: Exploration of Spatiotemporal Baseball Data*, M. Lage J. Piazzentin Ono, D. Cervone, J. Chiang, C. Dietrich, and C. Silva, IEEE Computer Graphics and Applications, 36(5): 28–37, 2016.
- [16] *Visual Analysis of Bike-Sharing Systems*, G. Oliveira, J. Sotomayor, R. Torchelsen, C. Silva, and J. Comba, Computers & Graphics, 60: 119-129, 2016.

- [17] *Reducing the Analytical Bottleneck for Domain Scientists: Lessons from a Climate Data Visualization Case Study*, A. Dasgupta, J. Poco, E. Bertini, and C. Silva, *Computing in Science and Engineering*, 18(1): 92–100, 2016.
- [18] *Visually Exploring Transportation Schedules*, C. Palomo, Z. Guo, C. Silva, and J. Freire, *IEEE Transactions on Visualization and Computer Graphics*, 22(1):170–179, 2016.
- [19] *Topology-based Catalogue Exploration Framework for Identifying View-Enhanced Tower Designs*, H. Doraiswamy, N. Ferreira, M. Lage, H. Vo, L. Wilson, H. Werner, M. Park, and C. Silva, *ACM Transactions on Graphics*, 34(6):230, 2015.
- [20] *Exploring Traffic Dynamics in Urban Environments Using Vector-Valued Functions*, J. Poco, H. Doraiswamy, H. Vo, J. Comba, J. Freire, and C. Silva, *Computer Graphics Forum*, 34(3):161–170, 2015.
- [21] *Bridging Theory with Practice: An Exploratory Study of Visualization Use and Design for Climate Model Comparison*, A. Dasgupta, J. Poco, Y. Wei, B. Cook, E. Bertini and C. T. Silva, *IEEE Transactions on Visualization and Computer Graphics*, 21(9):996–1014, 2015.
- [22] *Riding from Urban Data to Insight Using New York City Taxis*, J. Freire, C. Silva, Huy T. Vo, H. Doraiswamy, N. Ferreira, J. Poco, *IEEE Data Eng. Bull.* 37(4):43–55, 2014.
- [23] *Structured Open Urban Data: Understanding the Landscape*, L. Barbosa, K. Pham, C. Silva, M. Vieira, and J. Freire, *Big Data Journal*, 2:(3), 144–154, 2014.
- [24] *Using Physically Based Rendering to Benchmark SL Scanners*, E. Medeiros, H. Doraiswamy, M. Berger, and C. Silva, *Computer Graphics Forum (Proceedings of Pacific Graphics 2014)*, 33(7):71–80, 2014.
- [25] *Visual Reconciliation of Alternative Similarity Spaces in Climate Modeling*, J. Poco, A. Dasgupta, Y. Wei, W. Hargrove, C. Schwalm, D. Huntzinger, R. Cook, E. Bertini, and C. Silva, *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1923–1932, 2014.
- [26] *Genotet: An Interactive Web-based Visual Exploration Framework to Support Validation of Gene Regulatory Networks*, B. Yu, H. Doraiswamy, X. Chen, E. Miraldi, M. Arrieta-Ortiz, C. Hafemeister, A. Madar, R. Bonneau, and C. Silva, *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1903–1912, 2014.
- [27] *Using Topological Analysis to Support Event-Guided Exploration in Urban Data*, H. Doraiswamy, N. Ferreira, T. Damoulas, J. Freire, and C. Silva, *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2634–2643, 2014.
- [28] *A Weighted Delaunay Triangulation Framework for Merging Triangulations in a Connectivity Oblivious*, L.F. Silva, L.F. Scheidegger, T. Etienne, J. Comba, L. Nonato, and C. Silva, *Computer Graphics Forum*, 33(6):18–30, 2014.
- [29] *SimilarityExplorer: A Visual Inter-comparison Tool for Multifaceted Climate Data*, J. Poco, A. Dasgupta, Y. Wei, W. Hargrove, C. Schwalm, R. Cook, E. Bertini, and C. Silva, *Computer Graphics Forum*, 33(3):341–350, 2014.
- [30] *Fast Adaptive Blue Noise on Polygonal Surfaces*, Esdras medeiros, Lis Ingrid, Sinesio Pesco, and Claudio Silva, *Graphical Models*, 76(1):17–29, 2014.

- [31] *Verifying Volume Rendering Using Discretization Error Analysis*, Tiago Etienne, D. Jonsson, T. Ropinski, C. Scheidegger, J. Comba, L. G. Nonato, R. M. Kirby, A. Ynnerman, and C. T. Silva, IEEE Transactions on Visualization and Computer Graphics, 20(1):140–154, 2014.
- [32] *Visual Exploration of Big Spatio-Temporal Urban Data: A Study of New York City Cab Trips*, Nivan Ferreira, Jorge Poco, Huy T. Vo, Juliana Freire and Claudio Silva, IEEE Transactions on Visualization and Computer Graphics (Proceedings of VAST), 19(12):2149–2158, 2013.
- [33] *Practical considerations on Marching Cubes 33 topological correctness*, Lis Custodio, Tiago Etienne, Sinesio Pesco, Claudio T. Silva, Computers & Graphics 37(7):840–850, 2013.
- [34] *Vector Field k-Means: Clustering Trajectories by Fitting Multiple Vector Fields*, N. Ferreira, J.T. Klosowski, C. Scheidegger, and C. Silva, Computer Graphics Forum (Proceedings of EuroVis 2013). **Best paper honourable mention award.**
- [35] *Ultrascale Visualization of Climate Data*, D. Williams, T. Bremer, C. Doutriaux, J. Patchett, S. Williams, G. Shipman, R. Miller, D. Pugmire, B. Smith, C. Steed, E. Wes Bethel, H. Childs, H. Krishnan, P. Prabhath, M. Wehner, C. Silva, E. Santos, D. Koop, T. Ellqvist, J. Poco, B. Geveci, A. Chaudhary, A. Bauer, A. Pletzer, D. Kindig, G. Potter, and T. Maxwell, IEEE Computer, 46(9): 68-76, 2013.
- [36] *UV-CDAT: Analyzing Climate Datasets from a User’s Perspective*, E. Santos, J. Poco, Y. Wei, S. Liu, B. Cook, D. Williams and C. Silva, Computing in Science & Engineering, 15(1):94–103, 2013.
- [37] *VisTrails SAHM: visualization and workflow management for species habitat modeling*, J.T. Morissette, C.S. Jarnevich, T.R. Holcombe, C.B. Talbert, D. Ignizio, M.K. Talbert, C. Silva, D. Koop, A. Swanson, and N.E. Young, Ecography, 36(2):129-135, 2013.
- [38] *A Benchmark for Surface Reconstruction*, M. Berger, J. Levine, L. G. Nonato, G. Taubin, and C. Silva, ACM Transactions on Graphics, 32(2):20, 2013.
- [39] *Quad-Mesh Generation and Processing: a survey*, D. Bommers, B. Lévy, N. Pietroni, E. Puppo, C. Silva, M. Tarini, and D. Zorin, Computer Graphics Forum (Proceedings of Eurographics 2012), 32(6):51-76, 2013.
- [40] *Nonrigid Matching of Undersampled Shapes via Medial Diffusion*, M. Berger and C. Silva. Computer Graphics Forum (Proceedings of Symposium on Geometry Processing 2012), 31(5):1587–1596, 2012.
- [41] *Making Computations and Publications Reproducible with VisTrails*, J. Freire and C. Silva, Computing in Science and Engineering, 14(4):18–25, 2012.
- [42] *Medial Kernels*, M. Berger and C. Silva. Computer Graphics Forum (Proceedings of Eurographics 2012), 31(2):795–804, 2012.
- [43] *ISP: An Optimal Out-Of-Core Image-Set Processing Streaming Architecture for Parallel Heterogeneous Systems*, L. Ha, J. Krueger, J. Comba, C. Silva, and S. Joshi, IEEE Transactions on Visualization and Computer Graphics), 18(6):838–851, 2012.
- [44] *Simple and Efficient Mesh Layout with Space-filling Curves*, H. Vo, L. Scheidegger, V. Pascucci, and C. Silva, Journal of Graphics Tools, GPU, and Game Tools, 16(1):25–39, 2012.
- [45] *Interactive Quadrangulation with Reeb Atlases and Connectivity Textures*, J. Tierny, J. Daniels II, L. G. Nonato, V. Pascucci and C. Silva, IEEE Transactions on Visualization and Computer Graphics), 18(10):1650–1663, 2012.

- [46] *HyperFlow and ITK v4 Integration: Exploring the use of a modern parallel dataflow architecture in ITK*, H. Vo, L. Lins, and C. Silva, *The Insight Journal*, 04-2012.
- [47] *Inspired Quadrangulation*, J. Tierny, J. Daniels II, L. G. Nonato, V. Pascucci and C. Silva, *Computer-Aided Design (Proceedings of SIAM Conference on Geometric and Physical Modeling)*, 43(11):1516–1526, 2011.
- [48] *Efficient Probabilistic and Geometric Anatomical Mapping using Particle Mesh Approximation on GPUs*, L. Ha, M. Prastawa, G. Gerig, J. Gilmore, C. Silva and S. Joshi, *International Journal of Biomedical Imaging*, 2011.
- [49] *Template-Based Quadrilateral Mesh Generation from Imaging Data*, M. Lizier, M. Siqueira, J. Daniels II, C. Silva and L. Nonato, *The Visual Computer*, 27(10):887–903, 2011.
- [50] *Managing Data for Visual Analytics: Opportunities and Challenges*, J.-D. Fekete and C. Silva, *IEEE Data Eng. Bull.* 35(3): 27-36, 2012.
- [51] *Topology Verification for Isosurface Extraction*, T. Etienne, L. Nonato, C. Scheidegger, J. Tierny, T. Peters, V. Pascucci, R. M. Kirby, and C. Silva, *IEEE Transactions on Visualization and Computer Graphics*, 18(6):952–965, 2012. **Spotlight paper.**
- [52] *BirdVis: Visualizing and Understanding Bird Populations*, N. Ferreira, L. Lins, D. Fink, S. Kelling, C. Wood, J. Freire, and C. Silva, *IEEE Transactions on Visualization and Computer Graphics (Proceedings of InfoVIS 2011)*, 17(12):2374-2383, 2011.
- [53] *Streaming-Enabled Parallel Data Flow Framework in the Visualization ToolKit*, H. Vo, J. Comba, B. Geveci, and C. Silva, *Computing in Science and Engineering*, 13(3):72-83, 2011.
- [54] *A User Study of Visualization Effectiveness Using EEG and Cognitive Load*, E. Anderson, K. Potter, L. Matzen, J. Shepherd, G. Preston, and C. Silva, *Computer Graphics Forum (Proceedings of EuroVis 2011)*. **Best paper award – 2nd prize.**
- [55] *Template-based Quadrilateral Meshing*, J. Daniels II, M. Lizier, M. Siqueira, C. Silva and L.G. Nonato, *Computers and Graphics (Proceedings of Shape Modeling International 2011)*, 35(3), 2011.
- [56] *The ALPS project release 2.0: Open source software for strongly correlated systems*, B. Bauer, L. D. Carr, H.G. Evertz, A. Feiguin, J. Freire, S. Fuchs, L. Gamper, J. Gukelberger, E. Gull, S. Guertler, A. Hehn, R. Igarashi, S.V. Isakov, D. Koop, P.N. Ma, P. Mates, H. Matsuo, O. Parcollet, G. Pawlowski, J.D. Picon, L. Pollet, E. Santos, V.W. Scarola, U. Schollwck, C. Silva, B. Surer, S. Todo, S. Trebst, M. Troyer, M.L. Wall, P. Werner, S. Wessel, *Journal of Statistical Mechanics: Theory and Experiment (JSTAT)*, 5:P05001, 2011.
- [57] *Using VisTrails and Provenance for Teaching Scientific Visualization*, C. Silva, E. Anderson, E. Santos, and J. Freire, *Computer Graphics Forum*, 30(1):75–84, 2011. (Presented at EUROGRAPHICS 2010 Educator Program, 2010). **Best paper award.**
- [58] *PedVis: A Structured, Space Efficient Technique for Pedigree Visualization*, C. Tuttle, L. G. Nonato, and C. Silva. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Information Visualization 2010)*, 16(6):1063–1072, 2010.
- [59] *Two-Phase Mapping for Projecting Massive Data Sets*, F. V. Paulovich, L. G. Nonato, and C. Silva. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization 2010)*, 16(6):1281-1290, 2010.

- [60] *Interactive Vector Field Feature Identification*, J. Daniels, E. W. Anderson, L. G. Nonato, and C. Silva. IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization 2010), 16(6):1560–1568, 2010.
- [61] *Using Python for Signal Processing and Visualization*. E. Anderson, G. Preston, and C. Silva. IEEE Computing in Science and Engineering 12(4) pp 90–95, 2010.
- [62] *Fiedler Trees for Multiscale Surface Analysis*, M. Berger, L. G. Nonato, V. Pascucci, and C. Silva, Computer & Graphics (Proceedings of IEEE International Conference on Shape Modeling and Applications (SMI) 2010).
- [63] *Streaming-Enabled Parallel Dataflow Architecture for Multicore Systems*, H. Vo, B. Summa, D. Osmani, J. Comba, V. Pascucci, and C. Silva, Computer Graphics Forum (Proceedings of EuroVis 2010).
- [64] *Effects of 10Hz rTMS on the neural efficiency of working memory*, G. A. Preston, E. W. Anderson, E. Wassermann, T. Goldberg, and C. Silva, Journal of Cognitive Neuroscience, 22(3):447–456, 2010.
- [65] *Verifiable Visualization for Isosurface Extraction*, T. Etienne, C. Scheidegger, L. G. Nonato, R. M. Kirby, and C. Silva. IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization 2009).
- [66] *VisMashup: Streamlining the Creation of Custom Visualization Applications*, E. Santos, L. Lins, J. Ahrens, J. Freire, and C. Silva. IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization 2009).
- [67] *Semi-Regular Quadrilateral Remeshing from Simplified Base Domains*, J. Daniels, E. Cohen, and C. Silva. Computer Graphics Forum (Proceedings of Symposium on Geometry Processing 2009), 28(5):1427–1435, 2009.
- [68] *Localized Quadrilateral Coarsening*, J. Daniels, E. Cohen, and C. Silva. Computer Graphics Forum (Proceedings of Symposium on Geometry Processing 2009), 28(5):1436–1444, 2009.
- [69] *Robust Topology-Based Multiscale Analysis of Scientific Data*, A. Gyulassy, L. G. Nonato, P.-T. Bremer, C. Silva, and Valerio Pascucci. Computing in Science and Engineering, 11(5):88–95, 2009.
- [70] *Fast 4-way parallel radix sorting on GPUs*, L. Ha, J. Krueger, and C. Silva. Computer Graphics Forum, 28(8):2368–2378, 2009.
- [71] *Image-Space Acceleration for Direct Volume Rendering of Unstructured Grids using Joint Bilateral Upsampling*, S. P. Callahan and C. Silva, Journal of Graphics, GPU, & Game Tools, 14(1):115, 2009.
- [72] *Bandwidth Selection and Reconstruction Quality in Point-Based Surfaces*, H. Wang, C. E. Scheidegger, and C. Silva, IEEE Transactions on Visualization and Computer Graphics, 15(4):572–582, 2009.
- [73] *Marching Cubes without Skinny Triangles*, C. Dietrich, J. Comba, L. Nedel, C. Scheidegger, J. Schreiner, and C. Silva. Computing in Science and Engineering, 11(2):82–87, 2009.
- [74] *Improving Mesh Quality of Marching Cubes Using Edge Transformations*, C. Dietrich, J. Comba, L. Nedel, C. Scheidegger, J. Schreiner, and C. Silva. IEEE Transactions on Visualization and Computer Graphics, 15(1):150–159, 2009.
- [75] *Quadrilateral Mesh Simplification*, J. Daniels, C. Silva, J. Shepherd, and E. Cohen, ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia 2008).

- [76] *The Need for Verifiable Visualization*, R. M. Kirby and C. Silva. IEEE Computer Graphics and Applications, 28(5):78–83, 2008.
- [77] *Interactive Transfer Function Specification for Direct Volume Rendering of Disparate Volumes*, F. Bernardon, L. Ha, S. Callahan, J. Comba, and C. Silva. Computing in Science and Engineering, 10(6):82–89, 2008.
- [78] *VisComplete: Automating Suggestions for Visualization Pipelines*, D. Koop, C. Scheidegger, S. Callahan, J. Freire, and C. Silva. IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization 2008), 14(6):1691–1698, 2008.
- [79] *Edge Groups: A New Approach to Understanding the Mesh Quality of Marching Methods*, C. Dietrich, J. Comba, L. Nedel, C. Scheidegger, and C. Silva. IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization 2008), 14(6):1651–1658, 2008.
- [80] *Revisiting Histograms and Isosurface Statistics*, C. Scheidegger, J. Schreiner, B. Duffy, H. Carr and C. Silva. IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization 2008), 14(6):1659–1666, 2008.
- [81] *Spline-Based Feature Curves from Point-Sampled Geometry*, J. Daniels, T. Ochotta, L. Ha, and C. Silva. The Visual Computer, 24(6):449–462, 2008.
- [82] *Scientific Exploration in the Era of Ocean Observatories*, A. Baptista, B. Howe, J. Freire, D. Maier, and C. Silva. Computing in Science and Engineering, 10(3):53–58, 2008.
- [83] *Provenance for Computational Tasks: A Survey*, J. Freire, D. Koop, E. Santos, and C. Silva. Computing in Science and Engineering, 10(3):11–21, 2008.
- [84] *Provenance in Comparative Analysis: A Study in Cosmology*, E. W. Anderson, J. Ahrens, K. Heitmann, S. Habib, and C. Silva. Computing in Science and Engineering, 10(3):30–37, 2008.
- [85] *Robust Soft Shadow Mapping with Depth Peeling*, L. Bavoil, S. Callahan, and C. Silva. Journal of Graphics Tools, 13(1):19–30, 2008.
- [86] *Tackling the Provenance Challenge One Layer at a Time*, C. Scheidegger, D. Koop, E. Santos, H. Vo, S. Callahan, J. Freire, and C. Silva. Concurrency And Computation: Practice And Experience, 20(5):473–483, 2008.
- [87] *Direct Volume Rendering: A 3D Plotting Technique for Scientific Data*, S. P. Callahan, J. H. Callahan, C. E. Scheidegger, and C. Silva, Computing in Science and Engineering, 10(1):88–92, 2008.
- [88] *Special Issue: The First Provenance Challenge*, L. Moreau et al., Concurrency and Computation: Practice and Experience, 20(5):409–418, 2008.
- [89] *Provenance for Visualization: Reproducibility and Beyond*, C. Silva, J. Freire, and S. P. Callahan, Computing in Science and Engineering, 9(5):82–89, 2007.
- [90] *Querying and Creating Visualizations by Analogy*, C. E. Scheidegger, H. T. Vo, D. Koop, J. Freire, and C. Silva. IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization 2007), 13(6):1560–1567. **Best paper award.**
- [91] *An Adaptive Framework for Visualizing Unstructured Grids with Time-Varying Scalar Fields*, F. Bernardon, S. Callahan, J. Comba, and C. Silva. Parallel Computing, 33(6):391–405, 2007.

- [92] *Streaming Simplification for Tetrahedral Meshes*, H. Vo, S. Callahan, P. Lindstrom, V. Pascucci, and C. Silva. IEEE Transactions on Visualization and Computer Graphics, 13(1):145-155, 2007.
- [93] *GPU-based Tiled Ray Casting using Depth Peeling*, F. Bernardon, C. Pagot, J. Comba, and C. Silva, Journal of Graphics Tools, 11(4):1–16, 2006.
- [94] *High-Quality Extraction of Isosurfaces from Regular and Irregular Grids*, J. Schreiner, C. Scheidegger, and C. Silva. IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization 2006), 12(5):1205–1212, 2006.
- [95] *Progressive Volume Rendering of Large Unstructured Grids*, S. Callahan, L. Bavoil, V. Pascucci, and C. Silva. IEEE Transactions on Visualization and Computer Graphics (Proceedings of IEEE Visualization 2006), 12(5):1307-1314, 2006.
- [96] *Direct (Re)Meshing for Efficient Surface Processing*, J. Schreiner, C. Scheidegger, S. Fleishman, and C. Silva. Computer Graphics Forum (Proceedings of Eurographics 2006), 25(3):527–536, 2006.
- [97] *A Survey of GPU-Based Volume Rendering of Unstructured Grids*, C. Silva, J. Comba, S. Callahan, and F. Bernardon, Brazilian Journal of Theoretic and Applied Computing (RITA), 12(2):9–29, 2005.
- [98] *Image-Space Visibility Ordering for Cell Projection Volume Rendering of Unstructured Data*, R. Cook, N. Max, C. Silva, and P. Williams, IEEE Transactions on Visualization and Computer Graphics, 10(6):695–707, 2004.
- [99] *Computing and Rendering Point Set Surfaces*, M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. Silva, 9(1):3–15, IEEE Transactions on Visualization and Computer Graphics, 2003.
- [100] *Out-Of-Core Sort-First Parallel Rendering for Cluster-Based Tiled Displays*, W. Corrêa, J. Klosowski, and C. Silva, Parallel Computing, Vol 29, pp. 325–338, 2003.
- [101] *Robust Moving Least-squares Fitting with Sharp Features*, S. Fleishman, D. Cohen-Or, and C. Silva. ACM Transactions on Graphics (Proceedings of SIGGRAPH 2005), 24(3):544–552, 2005.
- [102] *Hardware-Assisted Visibility Sorting for Unstructured Volume Rendering*, S. Callahan, M. Ikits, J. Comba, and C. Silva, IEEE Transactions on Visualization and Computer Graphics, 11(3):285–295, 2005.
- [103] *Progressive Point Set Surfaces*, S. Fleishman, M. Alexa, D. Cohen-Or, and C. Silva, ACM Transactions on Graphics, 22(4):997–1011, 2003.
- [104] *A Survey of Visibility for Walkthrough Applications*, D. Cohen-Or, Y. Chrysanthou, C. Silva, and F. Durand. 9(3):412-431, IEEE Transactions on Visualization and Computer Graphics, 2003.
- [105] *Modeling and Rendering of Real Environments*, W. Corrêa, M. Oliveira, C. Silva, and J. Wang, 9(2):127–156, Brazilian Journal of Theoretic and Applied Computing (RITA), 2002.
- [106] *Efficient Conservative Visibility Culling Using The Prioritized-Layered Projection Algorithm*, J. Klosowski and C. Silva, 7(4):365–379, IEEE Transactions on Visualization and Computer Graphics, 2001.
- [107] *Out-Of-Core Rendering of Large Unstructured Grids*, R. Farias and C. Silva, 21(4):42–50, IEEE Computer Graphics and Applications, 2001.

- [108] *Surface Reconstruction using Lower Dimensional Incremental Delaunay Triangulation*, M. Gopi, S. Krishnan, and C. Silva, Computer Graphics Forum (Proceedings of Eurographics 2000), 19:467–478, 2000.
- [109] *Visualization Research with Large Displays*, B. Wei, C. Silva, E. Koutsofios, S. Krishnan, and S. North, 20(4):50–54, IEEE Computer Graphics and Applications, 2000.
- [110] *Approximate Volume Rendering for Curvilinear and Unstructured Grids by Hardware-Assisted Polyhedron Projection*, N. Max, P. Williams, and C. Silva, 11:53–61, International Journal of Imaging Systems and Technology, 2000.
- [111] *Fast Polyhedral Cell Sorting for Interactive Rendering of Unstructured Grids*, J. Comba, J. Klosowski, N. Max, J. Mitchell, C. Silva, and P. Williams, Computer Graphics Forum (Proceedings of Eurographics 1999), 18:367–376, 1999.
- [112] *The Ball-Pivoting Algorithm for Surface Reconstruction*, F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin, 5(4):349–359, IEEE Transactions on Visualization and Computer Graphics, 1999.
- [113] *Efficient Compression of Non-Manifold Polygonal Meshes*, A. Gueziec, F. Bossen, G. Taubin, and C. Silva, 14(1-3):137–166, Computational Geometry: Theory and Applications, 1999.
- [114] *The Prioritized-Layered Projection Algorithm for Visible Set Estimation*, J. Klosowski and C. Silva, 6(2):108–123, IEEE Transactions on Visualization and Computer Graphics, 2000.
- [115] *The Lazy Sweep Ray Casting Algorithm for Rendering Irregular Grids*, C. Silva and J. Mitchell, 3(2):142–157, IEEE Transactions on Visualization and Computer Graphics, 1997.
- [116] *PVR: High Performance Volume Rendering*, C. Silva, A. Kaufman, and C. Pavlakos, pp. 18–28, IEEE Computational Science and Engineering (Special Issue on Visual Supercomputing), Winter 1996.

Conference Publications (93)

- [117] *Data visualization tool for monitoring transit operation and performance*, A. Kurkcu, F. Miranda, K. Ozbay, and C. Silva, 2017 5th IEEE International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS), pp. 598–603, 2017.
- [118] *Querying and Exploring Polygamous Relationships in Urban Spatio-Temporal Data Sets*, Y.-Y. Chan, F. Chirigati, H. Doraiswamy, C. Silva and J. Freire, ACM SIGMOD 2017, pp. 1643–1646, 2017.
- [119] *Using Change-Sets to Achieve a Bounded Undo and Make Tutorials in 3D Version Control Systems*, R. Vieira, J. B. Cavalcante Neto, C. Vidal, G. Vialaneix and C. Silva, 29th SIBGRAPI Conference on Graphics, Patterns and Images, SIBGRAPI 2016, pp. 144–151, 2016.
- [120] *Anonymizing NYC Taxi Data: Does It Matter?*, M. Douriez, H. Doraiswamy, C. Silva and J. Freire, In Proceedings of IEEE International Conference on Data Science and Advanced Analytics (DSAA) pp. 140–148, 2016.
- [121] *A GPU-Based Index to Support Interactive Spatio-Temporal Queries over Historical Data*, H. Doraiswamy, H. Vo, C. Silva, and J. Freire. In Proceedings of IEEE International Conference on Data Engineering (ICDE), pp. 1086–1097, 2016.

- [122] *A Scalable Approach for Data-Driven Taxi Ride-Sharing Simulation*, M. Ota, H. Vo, C. Silva, and J. Freire. In Proceedings of IEEE BigData 2015, pp. 888–897, 2015.
- [123] *Visualizing the Evolution of Module Workflows*, M. Hlawatsch, M. Burch, F. Beck, J. Freire, C. Silva, and D. Weiskopf. In Proceedings of the IEEE International Conference on Information Visualisation, pp. 40–49, 2015.
- [124] *Using Maximum Topology Matching to Explore Differences in Species Distribution Models*, J. Poco, H. Doraiswamy, M. Talbert, J. Morissette, and C. Silva, Proceedings of SciVis 2015, pp. 9–16, 2015.
- [125] *Wavelet-based visualization of time-varying data on graphs*, P. Valdivia, F. Dias, F. Petronetto, C. Silva, and L. Nonato, Proceedings of VAST 2015, pp. 1–8, 2015.
- [126] *Urbane: A 3D Framework to Support Data Driven Decision Making in Urban Development*, Nivan Ferreira, Marcos Lage, Harish Doraiswamy, Huy Vo, Luc Wilson, Heidi Werner, Muchan Park, and C. Silva, Proceedings of VAST 2015, pp. 97–104, 2015.
- [127] *An Urban Data Profiler*, D. Ribeiro, H. Vo, J. Freire, and C. Silva, WWW 2015 Companion Volume, 2015:1389–1394, 2015.
- [128] *Visualization and Analysis of Parallel Dataflow Execution with Smart Traces*, Daniel K. Osmari, Huy T. Vo, Cláudio T. Silva, João L. D. Comba, and Lauro Lins, Proceedings of SIBGRAPI 2014, pp. 165–172, 2014.
- [129] *Baseball4D: A Tool for Baseball Game Reconstruction & Visualization*, Carlos Dietrich, David Koop, Huy Vo, Claudio Silva, Proceedings of VAST 2014, pp. 23–32, 2014.
- [130] *Discovering and Visualizing Patterns in EEG Data*, E. W. Anderson, C. Chong, G. Preston, C. Silva, Proceedings of IEEE 6th Symposium of Pacific Visualization 2013, pp. 57–64, 2013.
- [131] *Visual Summaries for Graph Collections*, D. Koop, J. Freire, and C. Silva, Proceedings of IEEE 6th Symposium of Pacific Visualization 2013, pp. 105–112, 2013.
- [132] *HyperFlow: A Heterogeneous Dataflow Architecture*, H. Vo, D. Osmari, J. Comba, P. Lindstrom, and C. Silva, Proceedings of Eurographics Symposium on Parallel Graphics and Visualization (EGPGV), pp. 1–10, 2012.
- [133] *Connectivity Oblivious Merging of Triangulations*, L.F. Silva, L.F. Scheidegger, T. Etienne, C. Silva, L.G. Nonato, and J. Comba, Conference on Graphics, Patterns and Images (SIBGRAPI 2012), pp. 118–125, 2012. **Best paper award.**
- [134] *A wildland fire modeling and visualization environment*, J. Mandel, J. D. Beezley, A. K. Kochanski, V. Y. Kondratenko, L. Zhang, E. Anderson, J. Daniels II, C. Silva, and Christopher R. Johnson, Proceedings of the Ninth Symposium on Fire and Forest Meteorology, 2011.
- [135] *VisCareTrails: Visualizing Trails in the Electronic Health Record with Timed Word Trees, a Pancreas Cancer Use Case*, L. Lins, M. Heilbrun, J. Freire and C. Silva, Workshop on Visual Analytics in Healthcare (VAHC 2011), 2011.
- [136] *Parallel Large-data Visualization with Display Walls*, L. Scheidegger, H. Vo, J. Kruger, C. Silva and J. Comba. Proceedings IS&T/SPIE Electronic Imaging 2012, Visualization and Data Analysis (VDA), 2012.

- [137] *Parallel Visualization on Large Clusters using MapReduce*, H. Vo, J. Bronson, B. Summa, J. Comba, J. Freire, B. Howe, V. Pascucci, and C. Silva, IEEE Symposium on Large-Scale Data Analysis and Visualization, 2011.
- [138] *CrowdLabs: Social Analysis and Visualization for the Sciences*, P. Mates, E. Santos, J. Freire, and C. Silva, Statistical and Scientific Database Management (SSDBM), 2011.
- [139] *Massive Image Editing on the Cloud*, B. Summa, H. Vo, V. Pascucci and C. Silva, Proceedings of the IASTED International Conference on Computational Photography (CPhoto), 2011.
- [140] *A Provenance-Based Infrastructure for Creating Executable Papers*, D. Koop, E. Santos, P. Mates, H. T. Vo, P. Bonnet, B. Bauer, B. Surer, M. Troyer, D. N. Williams, J. E. Tohline, J. Freire, and C. Silva. Procedia Computer Science, 2011. ICCS 2011. **Grand Challenge Finalist.**
- [141] *Optimal Multi-Image Processing Streaming Framework on Parallel Heterogeneous Systems*, L. Ha, C. Silva, J. Krueger, J. Comba, and S. Joshi, 11th Eurographics Workshop on Parallel Graphics and Visualization (EGPGV 2011), 2011. **Best paper award.**
- [142] *Template-based Remeshing for Image Decomposition*, M. Lizier, M. Siqueira, J. Daniels II, C. Silva, and L. G. Nonato. SIBGRAPI 2010 – Brazilian Symposium on Computer Graphics and Image Processing, 2010. (Selected as one of the best papers, invited for journal submission.)
- [143] *Image Registration Driven by Combined Probabilistic and Geometric Descriptors*, Linh Ha, Marcel Prastawa, Guido Gerig, John H. Gilmore, Claudio T. Silva, Sarang Joshi, Proceedings of MICCAI 2010.
- [144] *Collaborative Monitoring and Analysis for Simulation Scientists*, R. Tchoua, S. Klasky, N. Podhorszki, B. Grimm, A. Khan, E. Santos, C. Silva, P. Mouallem, and M. Vouk. Proceedings of The 2010 International Symposium on Collaborative Technologies and Systems (CTS 2010).
- [145] *The Provenance of Workflow Upgrades*, D. Koop, C. Scheidegger, J. Freire, and C. Silva, 3rd International Provenance and Annotation Workshop (IPAW) 2010.
- [146] *Bridging Workflow and Data Provenance using Strong Links*, D. Koop, E. Santos, B. Bauer, M. Troyer, J. Freire, and C. Silva, Statistical and Scientific Database Management (SSDBM), 2010.
- [147] *Fast Parallel Unbiased Diffeomorphic Atlas Construction on Multi-Graphics Processing Units*, L. K. Ha, J. Krueger, P. T. Fletcher, S. Joshi and C. Silva, 9th Eurographics Workshop on Parallel Graphics and Visualization (EGPGV 2009), 2009.
- [148] *Enabling Advanced Visualization Tools in a Simulation Monitoring System*, E. Santos, J. Tierny, A. Khan, B. Grimm, L. Lins, J. Freire, V. Pascucci, C. Silva, S. Klasky, R. Barreto, N. Podhorszki, IEEE International Conference on e-Science 2009, pp. 358–365, 2009.
- [149] *Using Workow Medleys to Streamline Exploratory Tasks*, E. Santos, D. Koop, H. Vo, E. Anderson, J. Freire, and C. Silva, pp. 292–301, Statistical and Scientific Database Management (SSDBM), 2009.
- [150] *Using Mediation to Achieve Provenance Interoperability*, T. Ellkvist, D. Koop, J. Freire, C. Silva, and L. Strömbäck, IEEE International Conference on Scientific Workflows 2009.
- [151] *End-to-End eScience: Integrating Workflow, Query, Visualization, and Provenance at an Ocean Observatory*, B. Howe, P. Lawson, R. Bellinger, E. Anderson, E. Santos, J. Freire, C. Scheidegger, A. Baptista, and C. Silva, IEEE International Conference on e-Science 2008.

- [152] *Effects of Texture and Color on the Perception of Medical Images*, I. Cheng, A. Badalov, C. Silva, and A. Basu. 30th IEEE Engineering in Medicine and Biology Society, 2008.
- [153] *A First Study on Clustering Collections of Workflow Graphs*, E. Santos, L. Lins, J. P. Ahrens, J. Freire, and C. Silva. Second International Provenance and Annotation Workshop (IPAW) 2008.
- [154] *Towards Provenance-Enabling ParaView*, S. P. Callahan, J. Freire, C. E. Scheidegger, C. Silva, and Huy T. Vo. Second International Provenance and Annotation Workshop (IPAW) 2008.
- [155] *Using Provenance to Support Real-Time Collaborative Design of Workflows*, T. Ellkvist, D. Koop, E. W. Anderson, J. Freire, and C. Silva. Second International Provenance and Annotation Workshop (IPAW) 2008.
- [156] *Examining Statistics of Workflow Evolution Provenance: A First Study*, L. Lins, D. Koop, E. W. Anderson, S. P. Callahan, E. Santos, C. E. Scheidegger, J. Freire, and C. T. Silva. Statistical and Scientific Database Management (SSDBM), 2008.
- [157] *Optimal Bandwidth Selection for MLS Surfaces*, H. Wang, C. E. Scheidegger, and C. Silva, IEEE International Conference on Shape Modeling and Applications (SMI), 2008. **Best paper award.**
- [158] *Querying and Re-Using Workflows with VisTrails*, C. E. Scheidegger, H. T. Vo, D. Koop, J. Freire, and C. Silva, ACM SIGMOD 2008.
- [159] *Quality Improvement and Boolean-Like Cutting Operations in Hexahedral Meshes*, J.F. Shepherd, Y. Zhang, C. Tuttle, and C. Silva, Proceedings of the 10th Conference of the International Society of Grid Generation, 2007.
- [160] *Hardware-Assisted Point-Based Volume Rendering of Tetrahedral Meshes*, E. Anderson, S. Callahan, C. Scheidegger, J. Schreiner, and C. Silva. SIBGRAPI 2007 – Brazilian Symposium on Computer Graphics and Image Processing, 2007.
- [161] *iRun: Interactive Rendering of Large Unstructured Grids*, H. Vo, S. Callahan, N. Smith, C. Silva, W. Martin, D. Owen, D. Weinstein. 7th Eurographics Workshop on Parallel Graphics and Visualization (EGPGV 2007), pages 93–100, 2007.
- [162] *Robust Smooth Feature Extraction from Point Clouds*, J. Daniels, L. Ha, T. Ochotta, and C. Silva. Shape Modeling International 2007, pages 123–133, 2007. **Best paper finalist.**
- [163] *Towards Development of a Circuit Based Treatment for Impaired Memory: A Multidisciplinary Approach*, E. Anderson, G. Preston, and C. Silva. IEEE Engineering in Medicine and Biology Conference (EMBS) 2007, 2007.
- [164] *Multi-Fragment Effects on the GPU using the k-Buffer*, L. Bavoil, S.P. Callahan, A. Lefohn, J.L.D. Comba, and C. Silva. ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, pages 97–104, 2007.
- [165] *Volume Rendering of Time-Varying Scalar Fields on Unstructured Meshes*, F. Bernardon, S. Callahan, J. Comba, and C. Silva. 6th Eurographics Workshop on Parallel Graphics and Visualization (EGPGV 2006).
- [166] *Managing the Evolution of Dataflows with VisTrails*, S. P. Callahan, J. Freire, E. Santos, C. E. Scheidegger, C. Silva, and H. T. Vo, IEEE Workshop on Workflow and Data Flow for Scientific Applications (SciFlow) 2006.

- [167] *Visualizing Uncertainty with Uncertainty Multiples*, R. B. Gilbert, F. Tonon, J. Freire, C. Silva, and D. R. Maidment, American Society of Civil Engineers (ASCE) 2006 GeoCongress.
- [168] *VisTrails: Visualization meets Data Management*, S. P. Callahan, J. Freire, E. Santos, C. E. Scheidegger, C. Silva, and H. T. Vo, ACM SIGMOD 2006, pp. 745-747, 2006.
- [169] *Interactive Rendering of Large Unstructured Grids Using Dynamic Level-Of-Detail*, S. Callahan, J. Comba, P. Shirley, and C. Silva. IEEE Visualization 2005, pp. 199–206, 2005.
- [170] *Hardware Accelerated Simulated Radiography*, D. Laney, S. Callahan, N. Max, C. Silva, S. Langer, and R. Frank. IEEE Visualization 2005, pp. 343–350, 2005.
- [171] *Triangulating Point Set Surfaces with Bounded Error*, C. Scheidegger, S. Fleishman, and C. Silva. Eurographics Symposium on Geometry Processing 2005, pp. 63–72, 2005.
- [172] *Simplification of Unstructured Tetrahedral Meshes by Point-Sampling*, D. Uesu, L. Bavoil, S. Fleishman, J. Shepherd, and C. Silva, pp. 157–165, Volume Graphics 2005, pp. 157–165, 2005.
- [173] *Implicit Occluders*, S. Pesco, P. Lindstrom, V. Pascucci, and C. Silva, IEEE Symposium on Volume Visualization and Graphics 2004, pp. 47–54, 2004. (Selected as one of the best papers, invited for journal submission.)
- [174] *VisTrails: Enabling Interactive Multiple-View Visualizations*, L. Bavoil, S. Callahan, P. Crossno, J. Freire, C. Scheidegger, C. Silva, and H. Vo. IEEE Visualization 2005, pp. 135–142, 2005.
- [175] *On the Convexification of Unstructured Grids From A Scientific Visualization Perspective*, J. Comba, J. Mitchell, and C. Silva, Proceedings of Dagstuhl 2003. Scientific Visualization: Extracting Information and Knowledge from Scientific Datasets Editors: G.-P. Bonneau, T. Ertl, G. M. Nielson, Springer-Verlag, 2005.
- [176] *Visibility-Based Prefetching for Interactive Out-Of-Core Rendering*, W. Corrêa, J. Klosowski, and C. Silva, IEEE Parallel & Large-Data Visualization & Graphics Symposium 2003, pp. 1–8, 2003.
- [177] *Visualizing Spatial and Temporal Variability in Coastal Observatories*, W. Herrera-Jimenez, W. Corrêa, C. Silva, and A. Baptista, IEEE Visualization 2003, pp. 269–274, 2003.
- [178] *Volume Rendering for Curvilinear and Unstructured Grids*, N. Max, P. Williams, and C. Silva, Computer Graphics International, 2003.
- [179] *Out-Of-Core Sort-First Parallel Rendering for Cluster-Based Tiled Displays*, W. Corrêa, J. Klosowski, and C. Silva, 4th Eurographics Workshop on Parallel Graphics and Visualization, 2002.
- [180] *A Generic Programming Approach to Multiresolution Spatial Decompositions*, V. Mello, L. Velho, P. Roma, and C. Silva, International Workshop on Visualization and Mathematics 2002, Berlin-Dahlem, Germany, 2002.
- [181] *Towards Point-Based Acquisition and Rendering of Large Real-World Environments*, W. Corrêa, S. Fleishman, and C. Silva, SIBGRAPI 2002 – Brazilian Symposium on Computer Graphics and Image Processing, 2002.
- [182] *Integrating Occlusion Culling with View-Dependent Rendering*, J. El-Sana, N. Sokolovsky, and C. Silva, IEEE Visualization 2001, pp. 371–378, 2001.

- [183] *A Unified Infrastructure for Parallel Out-Of-Core Isosurface and Volume Rendering of Unstructured Grids*, Y.-J. Chiang, R. Farias, C. Silva, and B. Wei, pp. 59–66, IEEE Parallel & Large-Data Visualization & Graphics Symposium 2001.
- [184] *Parallelizing the ZSWEEP algorithm for Distributed-Shared Memory Architectures*, R. Farias, and C. Silva, International Workshop On Volume Graphics 2001.
- [185] *A Hardware-Assisted Visibility-Ordering Algorithm With Applications to Volume Rendering*, S. Krishnan, C. Silva, and B. Wei, pp. 233–242, Data Visualization 2001 Joint Eurographics-IEEE TVCG Symposium on Visualization, 2001.
- [186] *A Memory Insensitive Technique for Large Model Simplification*, P. Lindstrom and C. Silva, pp. 121–126, IEEE Visualization 2001.
- [187] *Point Set Surfaces*, M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. Silva, IEEE Visualization 2001, pp. 21–28, 2001. **Best paper finalist.**
- [188] *Cell Projection of Meshes With Non-Planar Faces*, N. Max, P. Williams, and C. Silva, Proceedings of Dagstuhl 2000.
- [189] *Time-Critical Rendering of Irregular Grids*, R. Farias, J. Mitchell, C. Silva, and B. Wylie, pp. 243–250, SIBGRAPI 2000 – Brazilian Symposium on Computer Graphics and Image Processing, 2000.
- [190] *ZSWEEP: An Efficient and Exact Projection Algorithm for Unstructured Volume Rendering*, R. Farias, J. Mitchell, and C. Silva, pp. 91–99, ACM Volume Visualization and Graphics Symposium, 2000. (72 citations)
- [191] *Rendering on a Budget: A Framework for Time-Critical Rendering*, J. Klosowski and C. Silva, pp. 115–122, IEEE Visualization, 1999. **Best paper finalist.**
- [192] *Efficient Compression of Non-Manifold Polygonal Meshes*, A. Gueziec, F. Bossen, G. Taubin and C. Silva, pp. 73–80, IEEE Visualization, 1999.
- [193] *Optimal Processor Allocation for Sort-Last Compositing under BSP-tree Ordering*, C. R. Ramakrishnan and C. Silva. SPIE Electronic Imaging, Visual Data Exploration and Analysis IV, 1999.
- [194] *Greedy Cuts: An Advancing Front Terrain Triangulation Algorithm*, C. Silva and J. Mitchell, pp. 137–144, ACM Symposium on Geographic Information Systems 1998.
- [195] *An Exact Interactive Time Visibility Ordering Algorithm for Polyhedral Cell Complexes*, C. Silva, J. Mitchell, and P. Williams, pp. 87–94, ACM/IEEE Volume Visualization Symposium, 1998.
- [196] *Simple, Fast, and Robust Ray Casting of Irregular Grids*, P. Bunyk, A. Kaufman, and C. Silva, In “Scientific Visualization”, pp. 30–36, Proceedings of Dagstuhl ’97, H. Hagen, G. Nielson, F. Post, eds., IEEE Computer Society Press, 2000. Also in “Advances in Volume Visualization”, ACM SIGGRAPH 98 Course #24, July 1998.
- [197] *External Memory Techniques for Isosurface Extraction in Scientific Visualization*, Y.-J. Chiang and C. Silva, In “AMS/DIMACS Proceedings of the DIMACS Workshop on External Memory Algorithms and Visualization”, J. Abello and J. Vitter, eds., DIMACS book series, American Mathematical Society, 1998. (Journal version of the presentation given at the workshop.)
- [198] *Interactive Out-Of-Core Isosurface Extraction*, Y.-J. Chiang, C. Silva, and W. Schroeder, pp. 167–174, IEEE Visualization, 1998.

- [199] *I/O Optimal Isosurface Extraction*, Y.-J. Chiang and C. Silva, pp. 293–300, IEEE Visualization, 1997.
- [200] *Wavelet and Entropy Analysis Combination to Evaluate Diffusion and Correlation Behaviors*, R. Chiou, M. Ferreira, C. Silva and A. Kaufman, SIBGRAPI '97 – Brazilian Symposium on Computer Graphics and Image Processing.
- [201] *Fast Rendering of Irregular Grids*, C. Silva, J. Mitchell and A. Kaufman, pp. 15–22, ACM/IEEE Volume Visualization Symposium, 1996. Selected as one of the best papers, invited for special issue.
- [202] *Three Dimensional Visualization of Proteins in Cellular Interactions*, C. Monks, P. Crossno, G. Davidson, C. Pavlakos, A. Kupfer, C. Silva and B. Wylie, pp. 363–366, IEEE Visualization, 1996.
- [203] *Using Wavelets to Extract Information from Volumetric Data*, R. Chiou, M. Ferreira, A. Kaufman, and C. Silva, pp. 576–582, International Conference on Information Systems Analysis and Synthesis, 1996.
- [204] *Tetra-Cubes: An algorithm to generate 3D isosurfaces based upon tetrahedra*, B. Piquet, C. Silva, and A. Kaufman, pp. 205–210, SIBGRAPI '96 – Brazilian Symposium on Computer Graphics and Image Processing, Minas Gerais, Brazil, 1996.
- [205] *Automatic Generation of Triangular Irregular Networks using Greedy Cuts*, C. Silva, J. S. B. Mitchell and A. Kaufman, pp. 201–208, IEEE Visualization, 1995.
- [206] *VolVis: A Diversified Volume Visualization System*, R. Avila, T. He, L. Hong, A. Kaufman, H. Pfister, C. Silva, L. Sobierajski, S. Wang, pp. 31–38, IEEE Visualization, 1994.
- [207] *Parallel Performance Measures for Volume Ray Casting*, C. Silva and A. Kaufman, pp. 196–203, IEEE Visualization, 1994.
- [208] *Flow Surface Probes for Vector Field Visualization*, C. Silva, L. Hong and A. Kaufman, In “Scientific Visualization: Overviews, Methodologies and Techniques”, Dagstuhl '94, G. Nielson, H. Mueller, and H. Hagen, eds., IEEE Computer Society Press, 1997.
- [209] *Minhoca Plus – A Local Area Network for Teaching*, J. Coelho, C. Silva, M. Vieira, and A. Oliveira, VII Brazilian Conference on Computer Networks, UFRGS, March 1989. (In Portuguese.)

Patents (12 granted)

- [210] US patent 8,762,186, *Analogy based workflow identification*, issued to the University of Utah on June 24, 2014.
- [211] US patent 8,190,633, *Enabling provenance management for pre-existing applications*, issued to the University of Utah on May 29, 2012.
- [212] US patent 8,229,967, *Space efficient visualization of pedigree data*, issued to the University of Utah on July 24, 2012.
- [213] US patent 8,060,391, *Automated development of data processing results*, issued to the University of Utah on November 15, 2011.
- [214] US patent 6,968,299, *Method and apparatus for reconstructing a surface using a ball-pivoting algorithm*, issued to IBM on November 22, 2005.

- [215] US patent 6,933,946, *Method for out-of core rendering of large 3D models*, issued to AT&T on August 23, 2005.
- [216] US patent 6,831,636, *System and Process for Level of Detail Selection Based on Approximate Visibility Estimation*, issued to IBM on December 14, 2004.
- [217] US patent 6,801,215, *Hardware-Assisted Visibility-Ordering Algorithm*, issued to AT&T on October 5, 2004.
- [218] US patent 6,452,596, *Methods and Apparatus for the Efficient Compression of Non-manifold Polygonal Meshes*, issued to IBM on September 17th, 2002.
- [219] US patent 6,445,389, *Compression of Polygonal Models with Low Latency Decompression*, issued to IBM on September 3rd, 2002.
- [220] US patent 6,414,680, *System, Program Product And Method Of Rendering A Three Dimensional Image On a Display*, issued to IBM on July 2nd, 2002.
- [221] US patent 6,356,262, *System And Method For Fast Polyhedral Cell Sorting*, issued to IBM on March 12th, 2002.

Book Chapters (9)

- [222] *Programming with Big Data*, H. Vo, and C. Silva, Big Data and Social Science: A Practical Guide to Methods and Tools, pp. 125-144, 2016.
- [223] *Reproducibility using VisTrails*, J. Freire, D. Koop, F. Chirigati, and C. Silva. In Stodden et al., *Implementing Reproducible Research*, 2014.
- [224] *Estimating Species Distributions—Across Space, Through Time, and with Features of the Environment*, Kelling, S., Fink, D., Hochachka, W., Rosenberg, K., Cook, R., Damoulas, T., Silva, C. and Michener, W., *The DATA Bonanza: Improving Knowledge Discovery in Science, Engineering, and Business*, chapter 22, John Wiley & Sons, Inc., 2013.
- [225] *VisTrails*, D. Koop, E. Santos, C. E. Scheidegger, H. T. Vo, C. T. Silva, and J. Freire. In *Architecture of Open-Source Applications*, pp. 377-394, 2011.
- [226] *Multi-scale Unbiased Diffeomorphic Atlas Construction on Multi-GPUs*, L. Ha, J. Krüger, S. Joshi and C. Silva, GPU GEMS volume 1, 2010.
- [227] *Visualization for Data-Intensive Science*, C. Hansen, C. R. Johnson, V. Pascucci, and C. Silva. In *The Fourth Paradigm: Data Intensive Scientific Discovery*, K. Tolle, S. Tansley and T. Hey (Eds), 2010.
- [228] *Scientific Process Automation and Workflow Management*, B. Ludaescher, I. Altintas, S. Bowers, J. Cummings, T. Critchlow, E. Deelman, D. D. Roure, J. Freire, C. Goble, M. Jones, S. Klasky, T. McPhillips, N. Podhorszki, C. Silva, I. Taylor, and M. Vouk. In A. Shoshani and D. Rotem, editors, *Scientific Data Management: Challenges, Existing Technology, and Deployment*, Computational Science Series, chapter 13. Chapman & Hall/CRC, 2009.
- [229] *Modeling Cardiogenesis: The Challenges and Promises of 3D Reconstruction*, J. Pentecost, C. Silva, M. Pescitelli, and K. Thornburg, pp. 115–143, Vol. 56, *Current Topics in Developmental Biology*, 2003.
- [230] *Fast and Simple Occlusion Culling*, W. Corrêa, J. Klosowski, and C. Silva, pp. 353–358, *Game Programming Gems 3*, 2002.

Edited Proceedings (6)

- [231] *Proceedings of Advances in Visual Computing, 5th International Symposium, ISVC 2009, Part I*, G. Bebis, R. D. Boyle, B. Parvin, D. Koracin, Y. Kuno, J. Wang, R. Pajarola, P. Lindstrom, A. Hinkenjann, M. L. Encarnaçã, C. Silva, D. S. Las Vegas, NV, USA, 2009.
- [232] *Proceedings of Advances in Visual Computing, 5th International Symposium, ISVC 2009, Part II*, G. Bebis, R. D. Boyle, B. Parvin, D. Koracin, Y. Kuno, J. Wang, R. Pajarola, P. Lindstrom, A. Hinkenjann, M. L. Encarnaçã, C. Silva, D. S. Las Vegas, NV, USA, 2009.
- [233] *Proceedings of IEEE Visualization 2006*, E. Groeller, A. Pang, C. Silva, J. Stasko, and J. van Wijk, IEEE, ISSN 1077-2626, 2006.
- [234] *Proceedings of IEEE Visualization 2005*, C. Silva, E. Groeller, and H. Rushmeier, IEEE, 0-7803-9462-3, 2005.
- [235] *Proceedings of IEEE/ACM SIGGRAPH Symposium on Volume Visualization and Graphics 2004*, D. Silver, T. Ertl, C. Silva, IEEE, 0-7803-8781-3, 2004.
- [236] *Proceedings of the IEEE Symposium on Parallel and Large-Data Visualization and Graphics 2003*, A. Koning, R. Machiraju, and C. Silva, IEEE, 0-7803-8122-X, 2003.

Journal Editorials (2)

- [237] *Guest Editorial: Special Section on Visualization 2005*, C. Silva, E. Groeller, and H. Rushmeier. IEEE Transactions on Visualization and Computer Graphics, 12(4):419–420, 2006.
- [238] *Guest Editorial: Special Issue on Computational Provenance*, C. Silva and J. Tohline, Computing in Science and Engineering, 10(3):9-10, 2008.

Invited Conference Publications (7)

- [239] *Occam's razor and petascale visual data analysis*, E. W. Bethel, C. Johnson, S. Ahern, J. Bell, P.-T. Bremer, H. Childs, E. Cormier-Michel, M. Day, E. Deines, T. Fogal, C. Garth, C. G. R. Geddes, H. Hagen, B. Hamann, C. Hansen, J. Jacobsen, K. Joy, J. Krger, J. Meredith, P. Messmer, G. Ostrouchov, V. Pascucci, K. Potter, Prabhat, D. Pugmire, O. Rbel, A. Sanderson, C. Silva, D. Ushizima, G. Weber, B. Whitlock, K. Wu, Journal of Physics: Conference Series, SciDAC 2009 Conference, 2009.
- [240] *Software Infrastructure for Exploratory Visualization and Data Analysis: Past, Present and Future*, C. Silva and J. Freire, Journal of Physics: Conference Series, SciDAC 2008 Conference, July 2008.
- [241] *Comparing Techniques for Tetrahedral Mesh Generation*, M. Lizier, J. F. Shepherd, L. G. Nonato, J. Comba, and C. Silva. Inaugural International Conference of the Engineering Mechanics Institute, 2008.
- [242] *SciDAC visualization and analytics center for enabling technology*, E. W. Bethel, C. Johnson, K. Joy, S. Ahern, V. Pascucci, H. Childs, J. Cohen, M. Duchaineau, B. Hamann, C. Hansen, D. Laney, P. Lindstrom, J. Meredith, G. Ostrouchov, S. Parker, C. Silva, A. Sanderson, and X. Tricoche, Journal of Physics: Conference Series, SciDAC 2007 Conference, June 2007.
- [243] *Automation of Network-Based Scientific Workflows*, M. Vouk, I. Altintas, R. Barreto, J. Blondin, Z. Cheng, T. Critchlow, A. Khan, S. Klasky, J. Ligon, B. Ludaescher, P. A. Moullem, S. Parker, N.

Podhorszki, A. Shoshani, C. Silva, International Federation for Information Processing (IFIP), Volume 239, Grid-Based Problem Solving Environments, 2007.

- [244] *Managing Rapidly-Evolving Scientific Workflows*, J. Freire, C. Silva, S. P. Callahan, E. Santos, C. E. Scheidegger and H. T. Vo, Proceedings of the International Provenance and Annotation Workshop (IPAW), pp. 10-18, 2006. **Invited paper corresponding to Keynote Talk.**
- [245] *VACET: Proposed SciDAC2 Visualization and Analytics Center for Enabling Technologies*, E. Wes Bethel, C. Johnson, C. Hansen, S. Parker, A. Sanderson, C. Silva, X. Trichoche, V. Pascucci, H. Childs, J. Cohen, M. Duchaineau, D. Laney, P. Lindstrom, S. Ahern, J. Meredith, G. Ostouchov, K. Joy, B. Hamann, Journal of Physics: Conference Series, SciDAC 2006 Conference, Denver CO, 2006.

Invited Posters (1)

- [246] *Meet the Proposed SciDAC2 Visualization and Analytics Center for Enabling Technologies*, E. Wes Bethel, C. Johnson, C. Hansen, S. Parker, A. Sanderson, C. Silva, X. Trichoche, V. Pascucci, H. Childs, J. Cohen, M. Duchaineau, D. Laney, P. Lindstrom, S. Ahern, J. Meredith, G. Ostouchov, K. Joy, B. Hamann. Poster, 2006 SciDAC program meeting, Denver, CO.

Refereed Posters, SIGGRAPH Sketches, and Presentations (14)

- [247] *Desenvolvimento de Estruturas de Controle Explícito para o SGWfC VisTrails*, F. Seabra Chirigati, R. Dahis, S. Manuel Serra da Cruz, J. Freire, C. Silva, and M. Mattoso, 24th Brazilian Symposium On Databases (SBBD 2009). **Best poster award.**
- [248] *Simplifying the Design of Workflows for Large-Scale Data Exploration and Visualization*, J. Freire and C. Silva. In Proceedings of the Microsoft eScience Workshop, 2008.
- [249] *Using Mediation to Achieve Provenance Interoperability*, T. Ellkvist, D. Koop, J. Freire, C. Silva, and L. Strömbäck, IEEE International Conference on e-Science 2008.
- [250] *Enhanced neuronal efficiency and 10-12Hz spectral dynamics: Results from a concurrent EEG-TMS study*, G. A. Preston, E. W. Anderson, E. Wassermann, T. Goldberg, and C. Silva. 1st North American Symposium on TMS and Neuroimaging in Cognition and Behaviour, 2008.
- [251] *Towards Enabling Social Analysis of Scientific Data*, J. Freire and C. Silva, CHI Social Data Analysis Workshop, 2008.
- [252] *VisTrails: Using Provenance to Streamline Data Exploration*, E. W. Anderson, S. P. Callahan, D. A. Koop, E. Santos, C. E. Scheidegger, H. T. Vo, J. Freire, and C. Silva. Post Proceedings of the International Workshop on Data Integration in the Life Sciences (DILS) 2007. Invited for oral presentation.
- [253] *Effects of 10 Hz rTMS on Alpha Spectral Dynamics and Working Memory Performance*, G. A. Preston, E. W. Anderson, E. Wassermann, T. Goldberg, and C. Silva. Proceedings of Neuroscience Poster Session 2007.
- [254] *Real-Time Soft Shadows with Cone Culling*, L. Bavoil and C. Silva. ACM SIGGRAPH 2006 Sketches Program.
- [255] *Progressive Volume Rendering of Unstructured Grids on Modern GPUs*, S. Callahan, L. Bavoil, V. Pascucci, and C. Silva. ACM SIGGRAPH 2006 Sketches Program.

- [256] *Efficient Acquisition of Web Data Through Restricted Query Interfaces*, S. Byers, J. Freire, and C. Silva, WWW10, poster, 2001.
- [257] *Curvature-Based Estimation of Surface Sampling*, C. Silva and G. Taubin, SIAM Conference on Geometric Design, 1999.
- [258] *External Memory Techniques for Isosurface Extraction in Scientific Visualization*, Y.-J. Chiang and C. Silva, Third CGC Workshop on Computational Geometry, 1998.
- [259] *Lazy Sweep Ray Casting: A Fast Scanline Algorithm for Rendering Irregular Grids*, C. Silva and J. Mitchell, Second CGC Workshop on Computational Geometry, 1997.
- [260] *Automatic Generation of Triangular Irregular Networks using Greedy Cuts*, C. Silva, J. S. B. Mitchell and A. Kaufman, Fifth MSI-Stony Brook Workshop on Computational Geometry, 1995.

Other Publications (2)

- [261] *Through a New Looking Glass: Mathematically Precise Visualization*, K. E. Jordan, R. M. Kirby, C. Silva, and T. J. Peters, SIAM News, Vol. 43, Number 5, June 2010.
- [262] *DOE's SciDAC Visualization and Analytics Center for Enabling Technologies - Strategy for Petascale Visual Data Analysis Success*, E. Bethel, C. Johnson, C. Aragon, Prabhat, O. Rbel, G. Weber, V. Pascucci, H. Childs, P.-T. Bremer, B. Whitlock, S. Ahern, J. Meredith, G. Ostrouchov, K. Joy, B. Hamann, C. Garth, M. Cole, C. Hansen, S. Parker, A. Sanderson, C. Silva, X. Tricoche, CTWatch Quarterly, Volume 3, Number 4, November 2007.

Selected Technical Reports (7)

- [263] *DEFOG: A System for Data-Backed Visual Composition*, L. Lins, D. Koop, J. Freire, and C. Silva. SCI Technical Report, No. UUSCI-2011-003, University of Utah, 2011.
- [264] *A Unified Projection Operator for Moving Least Squares Surfaces*, T. Ochotta, C. Scheidegger, J. Schreiner, R. Kirby, and C. Silva. SCI Institute Technical Report, No. UUSCI-2007-006, 2007.
- [265] *Visualization in Radiation Oncology: Towards Replacing the Laboratory Notebook*, E. W. Anderson, S. P. Callahan, G. T.Y. Chen, J. Freire, E. Santos, C. E. Scheidegger, C. Silva, and H. T. Vo, SCI Institute Technical Report UUSCI-2006-17, 2006.
- [266] *Simplification of Unstructured Tetrahedral Meshes by Point-Sampling*, D. Uesu, L. Bavoil, S. Fleishman, and C. Silva, SCI Institute Technical Report UUSCI-2004-005, 2004.
- [267] *Out-Of-Core Algorithms for Scientific Visualization and Computer Graphics*, C. Silva, Y.-J. Chiang, W. Corrêa, J. El-Sana, and P. Lindstrom, LLNL Technical Report UCRL-JC-150434-REV-1, 2003.
- [268] *iWalk: Interactive Out-Of-Core Rendering of Large Models*, W. Corrêa, J. Klosowski, and C. Silva, Technical Report TR-653-02, Princeton University, 2002.
- [269] *Final Report for the Tera Computer TTI CRADA*, G. Davidson, C. Pavlakos, and C. Silva, Sandia Report SAND97-0134, Sandia National Laboratories, 1997.
- [270] *Parallel Volume Rendering of Irregular Grids*. C. Silva, Ph.D. thesis, Department of Computer Science, State University of New York at Stony Brook, 1996.

Research Funding

- [1] NVIDIA AI Lab, K. Cho (PI), C. Silva, R. Fergus, Y. LeCun and J. Li, US\$ 100K, 2017.
- [2] DARPA, *Streamlining Model Design, Comparison and Curation*, Juliana (PI), H. Doraiswamy (co-PI), Claudio Silva (co-PI), K. Cho (co-PI), and E. Bertine (co-PI), US\$ 3.8M, 2017-21.
- [3] National Science Foundation, *MRI: Development of Experiential Supercomputing: Developing a Transdisciplinary Research and Innovation Holodeck*, CNS-1626098, W. Burlison (PI), K. Perlin (Co-PI), M. Shelley (Co-PI), Jan Plass (Co-PI), A. Roginska (Co-PI), L. DuBois (co-I), C. Silva (co-I) US\$ 2.9M, 2016–2021.
- [4] National Science Foundation, *II-New: An Infrastructure of Display Devices to Study Visual Analytics Beyond the Desktop Recommended*, CNS-1730396, E. Bertini, C. Silva (Co-PI), US\$ 273K, 2017-2020.
- [5] MLB Advanced Media, *Sports Analytics Research*, C. Silva (PI), US\$ 335K, 2016-2017.
- [6] Moore and Sloan Foundations, *Moore-Sloan Data Science Initiative*, Joint effort by New York University, the University of California, Berkeley and the University of Washington. US\$ 37.8 million. 2013–2018.
- [7] National Science Foundation, *CPS: Frontier: SONYC: A Cyber-Physical System for Monitoring, Analysis and Mitigation of Urban Noise Pollution*, CPS-1544753, J. Bello (PI), R. DuBois (Co-PI), C. Silva (Co-PI), O. Nov (Co-PI), A. Arora (Co-PI), US\$ 4.6M, 2016-2021.
- [8] National Aeronautics and Space Administration (NASA), *OpenSpace: An Engine for Dynamic Visualization of Earth and Space Science for Informal Education and Beyond*, C. Silva (NYU PI), US\$ 1.2M, 2015-2020. (total grant: US\$ 6.2M)
- [9] NVIDIA AI Lab, K. Cho (PI), C. Silva, R. Fergus, Y. LeCun and J. Li, US\$ 100K, 2016.
- [10] National Science Foundation, *AitF: FULL: Collaborative Research: Provably Efficient GPU Algorithms*, CCF-1533564, J. Iacono (PI) and C. Silva (co-PI), US\$ 500K, 2015-19.
- [11] National Science Foundation, *MRI: Acquisition of an infrastructure for prototyping next-generation algorithms for large-scale visualization, data processing and analysis*, CNS-1229185, C. Silva (PI) and H. Vo, J. Freire, J. Iacono, and T. Suel, US\$ 800K (2012–2017).
- [12] State of New York, *Develop Data Storage and Access Platform for MTA Bustime Data*, C. Silva (PI) with K. Ozbay, US\$ 86K, 2015-16.
- [13] National Aeronautics and Space Administration (NASA) (through a USGS sub-contract), *Using the USGS Resource for Advanced modeling*, US\$ 444K, 2012-2016.
- [14] McGraw-Hill Education, *Education Data Algorithms and Visualizations*, US\$ 250K, C. Silva (PI), with J. Plass, L. Dubois, E. Bertini, and B. Ubell, 2015.
- [15] MLB Advanced Media, *Sports Analytics Research*, C. Silva (PI), US\$ 192K, 2015-2016.
- [16] AT&T Virtual University Research Initiative (VURI), C. Silva (PI), US\$ 25K, 2015.
- [17] Lawrence Livermore National Labs, *Ultrascale Visualization Climate Data Analysis Tools (UV- CDAT)*, C. Silva (PI), US\$ 100K, 2015.

- [18] Department of Energy, *Accelerated Climate Modeling For Energy*, C. Silva (PI), US\$ 225K, 2014-2017.
- [19] Kitware (Department of Energy subcontract), *ClimatePipes: User-Friendly Data Access, Data Manipulation, Data Analysis and Visualization of Community Climate Models*, C. Silva (PI), US\$ 500K, 2012–2014.
- [20] Sloan Foundation. *Computational Reproducibility: Understanding the Requirements and Building the Necessary Infrastructure*, J. Freire (PI), Co-PIs: C. Silva and D. Shasha. US\$74K. 2012–2014.
- [21] Kitware (National Institutes of Health subcontract), *ITK v4*, C. Silva (PI) and H. Vo (co-PI), US\$ 120K (2011-2012).
- [22] Department of Energy, *Ultra-scale Visualization Climate Data Analysis Tools (UV-CDAT)*, C. Silva (PI) US\$ 1.2M (2010-1014).
- [23] National Science Foundation, *III: Medium: Provenance Analytics: Exploring Computational Tasks and their History*, J. Freire (PI) and C. Silva (co-PI). US\$ 957K (2009-2012).
- [24] National Science Foundation, *II-NEW: The Utah Acquisition and Rapid Prototyping Laboratory*, A. Bargeil (PI), E. Cohen (co-PI), R. M. Kirby (co-PI), and C. Silva (co-PI). US\$ 391K (2009-2012).
- [25] Department of Energy. *SBIR Phase I and Phase II: Provenance-Enabling DOE Visualization Applications*. D. Koop (PI); Co-PIs: J. Freire and C. Silva. US\$ 850,000 (2008–2011).
- [26] National Science Foundation, *Where the Ocean Meets the Cloud: Ad Hoc Longitudinal Analysis and Collaboration Over Massive Mesh Data*, C. Silva (PI) and J. Freire (co-PI). US\$ 190K. (2009-2011) (This is a collaborative proposal with B. Howe, University of Washington.)
- [27] National Science Foundation, *CDI-Type II: Collaborative Research: The Open Wildland Fire Modeling E-community: a virtual organization accelerating research, education, and fire management technology*, ATM-0835821, C. Johnson (PI) and C. Silva (co-PI). US\$ 641,790 (Utah portion out of a total project budget of US\$ 1.65M). (2008-2012). Collaborative proposal with NCAR and University of Colorado at Denver.
- [28] National Science Foundation, *CRI: IAD A Service-Oriented Architecture for The Computation, Visualization, and Management of Scientific Data*, CNS-0751152, C. Silva (PI), J. Freire, S. Joshi, R. M. Kirby (co-PIs). US\$ 500,000 (2008-2011).
- [29] National Science Foundation, *Science and Technology Center for Coastal Margin Observation and Prediction*, OCE-0424602, A. Baptista (PI, OHSU), J. Freire and C. Silva (Utah co-PIs), Total: (approx) US\$ 20,000,000; Utah portion: US\$ 478,563 (2006-11).
- [30] Department of Energy, *Scientific Data Management Enabling Technology Center*, DOE SciDAC II. A. Shoshani (PI, LBNL), C. Silva (Utah PI), Total: (approx) US\$ 16,500,000; Utah portion: US\$ 910,000 (2006-11).
- [31] Department of Energy, *VACET: Visualization and Analytics Center for Enabling Technologies*, DOE SciDAC II. C. Johnson, C. Hansen, C. Silva, S. Parker, A. Sanderson, X. Tricoche (Utah Team), Total: (approx) US\$ 11,000,000; Utah portion: US\$ 2,790,726 (2006-11).
- [32] Department of Energy, *Towards a multi-threaded data-driven streaming execution model for VTK*, C. Silva (PI), C. Hansen, and V. Pascucci. US\$ 126K (2009).

- [33] ExxonMobil, *Imaging, Visualization, and Modeling Research Center*, R. Whitaker (PI), C. Hansen (co-PI), V. Pascucci (co-PI), and C. Silva (co-PI). US\$ 2.2M (2008–2013).
- [34] Department of Energy, *Supporting Pipelines of Retrieval, Analysis and Visualization of Web Data*, J. Freire (PI) and C. Silva (co-PI) US\$ 103K (2009-10).
- [35] Department of Energy, *Provenance Analytics Tools to Improve the Measurement of Usability and Insight in Visualization Applications*, C. Silva (PI) and J. Freire (co-PI). US\$ 100K (2009-10).
- [36] National Institutes of Health, *NCRR ARRA Administrative Supplement - Translational*, D. A. McClain (PI), L. Cannon-Albright, P. Renshaw, C. Silva, J. Freire, D. A. Yurgelun-Todd. US\$ 998,137 (2009-2011).
- [37] *National Science Foundation*, SBIR Phase I and IB: A Collaborative Architecture to Support Large-Scale Exploratory Workflows, IIP-0712592, S. P. Callahan (PI); Co-PIs: J. Freire and C. Silva. US\$ 150,000 (2007).
- [38] *State of Utah, Centers of Excellence*. Center for Software Process Automation and Exploratory Data Mining. G. Jones, J. Freire and C. Silva. US\$ 200,000 (2008–2009).
- [39] Department of Energy, *Integrating VisIt and VisTrails Software*, C. Silva. US\$ 53,209 (2009).
- [40] National Science Foundation, *MSPA-MCS: Collaborative Research: New Methods for Robust, Feature-Preserving Surface Reconstruction*, CCF-0528201 and CCF-0528209, C. Silva (lead PI, Utah), J. Mitchell (PI, Stony Brook). Total: US\$ 480,686 (2005-8); Utah portion: US\$ 275,599 (2005-8).
- [41] National Science Foundation, *SEIII: Managing Complex Visualizations*, IIS-0513692, J. Freire (PI) and C. Silva (co-PI). US\$ 530,252 (2005-8). REU supplement: US\$ 12,000 (2006).
- [42] National Science Foundation, *U.S. Brazil Collaborative Research: 3D Modeling and Visualization*, OISE-0405402, C. Silva (PI), E. Praun (co-PI) and R. Whitaker (co-PI). US\$ 85,000 (2004-6). REU supplements: US\$ 15,000 (2005).
- [43] State of Utah, Centers of Excellence. *Center for Management of Exploratory Workflows-Business Team*, J. Freire (PI) and C. Silva (co-PI), US\$ 50,000 (2007-8).
- [44] Department of Energy, *Topic in Visualization Research*, C. Silva (PI), C. Hansen and J. Freire (co-PIs). US\$ 200,000 (2007-8).
- [45] National Institutes of Health, *High Resolution Mapping Of Placental Gene Expression*, C. Silva (co-I), J. Pentecost (PI, OHSU). Approximately US\$ 210,000 (2005-7).
- [46] National Science Foundation, *Interactive Out-Of-Core Visualization of Large Polygonal Datasets*, CCF-0401498, Cláudio Silva (PI). US\$ 178,488 (2003–6). REU supplements: US\$ 12,000 (2004); US\$ 12,000 (2005).
- [47] Department of Energy, *Using Morse Theory in the Parameterization of Arbitrary 2-Manifolds*, C. Silva (PI). US\$ 35,306 (2006).
- [48] Department of Energy, *Advanced Volume Rendering Techniques*, C. Silva (PI). US\$ 90,000 (2006).
- [49] Department of Energy, *Utah Advanced Visualization Center*, C. Hansen (PI) and C. Silva (co-PI). US\$ 680,000 (2003–6).

- [50] National Science Foundation, *A Cluster Infrastructure to Support Retrieval, Management and Visualization of Massive Amounts of Data*, EIA-0323604, J. Freire (PI) and C. Silva (co-PI). US\$ 110,000 (2003–5). Institutional matching funds: US\$ 55,000.
- [51] Department of Defense (Army STTR), *A Scalable System for Enormous Dataset Volume Visualization on Commodity Hardware* (Phase I), W911NF-05-C-0107, D. Weinstein (PI, Visual Influence), C. Silva (PI, Utah), and J. Freire (co-PI). Phase I: US\$ 94,704 (2005-6).
- [52] Department of Energy, *Studying The Topology of Point-Set Surfaces*, C. Silva (PI). US\$ 37,492 (2005).
- [53] University of Utah Seed Grant, *Digital Geometry Processing Techniques for Spatial Genomics*, C. Silva (PI). US\$ 27,000 (2004-5).
- [54] Department of Energy, *Advanced Scientific Visualization Techniques*, C. Silva (PI). US\$ 56,000 (2004–5).
- [55] Department of Energy, *Rendering of Isosurfaces Using Implicit Occluders*, C. Silva (PI). US\$ 22,919 (2003).
- [56] Department of Energy, *Visualization of Adaptive Mesh Refinement in SAMRAI*, C. Silva (PI). US\$ 22,170 (2003).
- [57] Department of Energy, *Developing Techniques for High-Resolution Interactive Volume Rendering of Large Unstructured Volumetric Grids on Clusters of Commodity PCs*, C. Silva (PI). US\$ 92,984 (2003–4).
- [58] Department of Energy, *High-Performance Visualization*, J. Mitchell (PI) and C. Silva (co-PI). US\$ 303,196 (1996–2001).
- [59] National Science Foundation, *Efficient Geometric Algorithms in Support of Virtual Reality Systems*, Post-Doctoral CISE Associateship Award, CCR-9626370. J. Mitchell (PI) and C. Silva. US\$ 46,000 (1996–98).
- [60] Department of Energy, *Support for Research Assistant – Cláudio T. Silva*, A. Kaufman (PI). (approx) US\$ 50,000. (1995–96).

Advising

Current Post-doctoral Assistants (2 Post-doc)

Alexander Bock (Data Science Fellow, New York University, 2017–)
 Yitzchak Lockerman (Post-doc, New York University, 2016–)

Current Graduate Students (3 Ph.D.)

Jorge Henrique (Ph.D., New York University, since August 2015)
 Fabio Miranda (Ph.D., New York University, since August 2012)
 Bowen Yu (Ph.D., New York University, since August 2013)

Former Graduate Students and Post-doctoral Assistants (12 Post-doc, 17 Ph.D., 9 M.S.)

Marcel Campen (Post-doc, New York University, 2015–2017)
Lhaylla Crissaff (Post-doc, New York University, 2015-16)
Harish Doraiswamy (Post-doc, New York University, 2012–2015; Co-advised with Juliana Freire)
Aritra Dasgupta (Post-doc, New York University, 2012–2015)
Yunzhe (Alvin) Jia (M.S., New York University, 2015)
Nivan Ferreira (Ph.D., New York University, 2015)
Jorge Poco (Ph.D., New York University, 2015)
Guillaume Vialaneix (Post-doc, New York University, 2013–2014)
Lis Custodio Roque (Ph.D., PUC-Rio, 2014; Co-advised with Sinesio Pesco)
Joel Daniels (Post-doc, NYU-Poly, 2009–2011)
Lauro Lins (Post-doc, University of Utah and NYU-Poly 2007–2012; Co-advised with Juliana Freire))
Wendel Silva (M.S., NYU-Poly, 2013)
Daniel K. Osmari (M.S., NYU-Poly, 2013)
Jonathas Costa (M.S., NYU-Poly, 2013)
Tiago Etienne (Ph.D., University of Utah, 2013)
Matt Berger (Ph.D., University of Utah, 2012)
Erik Anderson (Ph.D., University of Utah, 2011)
David Koop (Ph.D., University of Utah, 2011; Co-advised with Juliana Freire)
Huy T. Vo (Ph.D., University of Utah, 2011)
Linh K. Ha (Ph.D., University of Utah, 2011; Co-advised with Sarang Joshi and Jens Krueger)
Caurissa Tuttle (M.S., University of Utah, 2011)
Emanuele Santos (Ph.D., University of Utah, 2010; Co-advised with Juliana Freire)
Hao Wang (M.S.–project option, University of Utah, 2010).
Carlos Scheidegger (Ph.D., University of Utah, 2009)
Joel D. Daniels II (Ph.D., University of Utah, 2009; Co-advised with Elaine Cohen)
Tilo Ochotta (Post-doc, 2008–2009)
John Schreiner (Ph.D., University of Utah, 2008)
Steven P. Callahan (Ph.D., University of Utah, 2008)
Heballa Benan Alzahawi (M.S.–project option, University of Utah, 2008)
Yuan Zhou (Post-doc, University of Utah, 2007–2008)
Louis Bavoil (M.S.–thesis option, University of Utah, 2006)
Steven P. Callahan (M.S.–thesis option, University of Utah, 2005)
Shachar Fleishman (Post-doc, University of Utah, 2004–2005)
Sinesio Pesco (Post-doc, University of Utah, 2003–2004)
Dirce Uesu (Post-doc, University of Utah, 2003–2004)
Wagner Corrêa (Ph.D., Princeton University, 2003; Co-advised with Szymon Rusinkiewicz)
Ricardo Farias (Ph.D., SUNY-Stony Brook, 2001; Co-advised with Joseph Mitchell)
Tsung-Chin Ho (Ph.D., SUNY-Stony Brook, 2001; Co-advised with Joseph Mitchell)

Graduate Thesis Defense Committees (17)

Liang Zhou (Ph.D., University of Utah, 2014)
Otavio Braga (Ph.D., New York University, 2013)
Lei Wang (Ph.D., Stony Brook University, 2013)
Denis Kovacs (Ph.D., New York University, 2013)
Hoa Nguyen (Ph.D., University of Utah, 2011)

Mathias Schott (Ph.D., University of Utah, 2011)
Abe Stephens (Ph.D., University of Utah, 2011)
Andrew Kensner (Ph.D., University of Utah, 2009)
Jelka Stevanovic (Ph.D., University of Utah, 2009)
Luciano Barbosa (Ph.D., University of Utah, 2009)
Thiago Ize (Ph.D., University of Utah, 2009)
Aaron Knoll (Ph.D., University of Utah, 2008)
Guo-Shi Li (Ph.D., University of Utah, 2008)
Miriah D. Meyer (Ph.D., University of Utah, 2008)
Xianming Chen (Ph.D., University of Utah, 2007)
Jason F. Shepherd (Ph.D., University of Utah, 2007)
Joel D. Daniels II (M.S., University of Utah, 2005)

Other Advising and Mentoring

Mentor, J. Comba (Associate Professor, Federal University of Rio Grande do Sul, Brazil), September 2010–August 2011.

Mentor, L. G. Nonato (Associate Professor, University of São Paulo, Brazil), August 2008–July 2010.

P. Hendricks, since July 2009. NSF REU student.

P. Mates, since February 2009. NSF REU student.

C. Brooks, since September 2008. NSF REU student.

Undergraduate advisor, W. Tyler, since September–December 2005. NSF REU student.

Undergraduate advisor, E. Anderson, August 2004–January 2005. NSF REU student.

Undergraduate advisor, H. Vo, May 2004–May 2005. Continuing into Ph.D. program.

Undergraduate advisor, N. Smith, September 2005–March 2007. NSF REU student.

Undergraduate advisor, J. Callahan, Summer 2007–Summer 2008. NSF REU student.

Undergraduate advisor, H. Wang, December 2006–August 2008. Continuing into Ph.D. program.

Research mentor, M. Lizier, November 2007–July 2008.

Research Mentor, T. Ochotta, September 2006–February 2007.

Research mentor, F. Bernardon, June–September 2006.

Research mentor, Y. Lima, February–May, 2006.

Ph.D. advisor, W. Herrera-Jimenez, 01/2003–10/2003. Dropped for personal reasons.

Research mentor (with J. Freire), L. Rocha, April–November 2003.

Research mentor, AT&T Summer Internship Program, W. Corrêa (Princeton University), Summer 2002.

Research mentor, AT&T-Labs Fellowship Program, L. Lloyd, Summer 2002.

Research mentor, AT&T Summer Internship Program, S. Fleishman, Summer 2001.

Research mentor, AT&T URP (Under Represented Minority Program), B. Anthony, Summer 2000.

Selected Tutorials (20)

Provenance-Enabled Data Exploration and Visualization

IEEE Visualization 2009.

Provenance and Scientific Workflows: Supporting Data Exploration and Visualization

IEEE International Conference on e-Science 2008.

Visualization and Data Analysis with VisTrails

SciDAC (Scientific Discovery through Advanced Computing) 2008.

GPU-Based Volume Rendering of Unstructured Grids

SIBGRAPI 2005.
Multi-resolution Modeling, Visualization and Compression of Volumetric Data
Eurographics 2004.
IEEE Visualization 2003.
Out-Of-Core Algorithms for Scientific Visualization and Computer Graphics
IEEE Visualization 2003.
IEEE Visualization 2002.
High-Performance Visualization of Large and Complex Scientific Datasets
ACM/IEEE SC 2002.
Rendering and Visualization in Affordable Parallel Environments
IEEE Visualization 2001.
Eurographics 2001.
ACM SIGGRAPH 2000.
IEEE Visualization 2000.
ACM SIGGRAPH 1999.
Eurographics 1999.
Eurographics 1998.
Visibility, problems, techniques and applications
ACM SIGGRAPH 2001.
ACM SIGGRAPH 2000.
Eurographics 1999.
Advances in Volume Visualization
ACM SIGGRAPH 1998.

Selected Invited Talks

Visualization and Analysis of Urban Data
University of Illinois at Chicago, January 21st, 2016 **Distinguished Lecture Series**
AT&T Data Science, December 7th, 2015
International Symposium on Visual Computing (ISVC) 2015, December 15th, 2015 **Keynote**
EuroVis 2015, May 26th, 2015 **Keynote**

Visualization and The City: Projects in Urban Data Visualization and Sports Analytics
Data Visualization New York Meetup, November 17th, 2015

Building Tools for Urban Data Science
Chesapeake Large-Scale Analytics Conference, October 14th, 2015

Attempts at Building UrbanGIS Infrastructure
The First International Workshop on Smart Cities and Urban Analytics (UrbanGIS) 2015, November 3rd, 2015

Big Data Platforms for Urban Data
Center of Excellence in Wireless and Information Technology (CEWITT) 2015 Conference, October 19th, 2015
National Geospatial Intelligence agency/Argonne National Labs Megacities Workshop, September 10th, 2015

Exploring Big Urban Data
NYU Shanghai Big Data Symposium, November 21, 2014 **Keynote**
Shandong University, November 19th, 2014
ELLIIT Workshop 2014, Linkping University, Sweden, October 23rd, 2014 **Keynote**
American Express, April 7th, 2015

Urban Data Visualization
Dagstuhl, June 3rd, 2014
NII-Japan, March 10th, 2014

Big Data Research at the Center for Urban Science and Progress
IBM Research, February 11th, 2014

Measuring Visualization Correctness and Effectiveness

SIBGRAPI 2011 Brazilian Symposium on Computer Graphics and Image Processing, October 28th, 2011 **Keynote**
Geometry and Topology for Quadrilateral Mesh Processing and Verifiable Visualization
 Symposium on Computational Geometry, June 16th, 2010 **Plenary Talk**
 Federal University of Ceara, June 7th, 2010
 University of Chicago, April 8, 2010

High-Quality Isosurfaces and Surface Re(Meshing)
 Washington University, December 4th, 2009
 Brown University, April 7th, 2009
 Linköping University (Norrköping Campus), January 20th, 2009

Introduction to Computational Provenance
 Workshop on Monte Carlo data evaluation, archiving and provenance, Inst. Theor. Physics, ETH, Nov. 2nd, 2008.

Introduction to VisTrails
 Workshop on Monte Carlo data evaluation, archiving and provenance, Inst. Theor. Physics, ETH, Nov. 2nd, 2008.

VisTrails: Provenance and Data Exploration
 Harvard University, April 9th, 2009
 NIH National Biomedical Computation Resource (NBCR) Summer Institute 2008, August 4th, 2008.

Software Infrastructure for Exploratory Visualization and Data Analysis: Past, Present and Future
 SciDAC (Scientific Discovery through Advanced Computing) 2008 (Featured Speaker), July 17th, 2008.

Visualization at the University of Utah
 Linköping University, January 16th, 2009
 Workshop on Interactive Data Visualization (co-located with SIBGRAPI 2007), October 7th, 2007.

Supporting Data Exploration through Visualization
 Open Grid Forum 19, February 1st, 2007.
 IBM T. J. Watson Research Center, October 27th, 2006.
 International Fall School and Workshops, Universidad Nacional Autonoma de Mexico, October 18th, 2006.

Scalable Techniques for Scientific Visualization,
 IEEE EMBS Chapter talk, University of Alberta, August 1st, 2007.
 CIG Computational Geodynamics and Scientific Computing Workshop, UT-Austin, October 17th, 2006.
 Microsoft eScience 2006, October 14th, 2006.

Surface Re(Meshing) and Applications
 Federal University of Rio Grande do Sul, August 3rd, 2006.
 Ayia Napa Summer Seminar 2006, June 29th, 2006.
 IMPA-Brazil, June 14th, 2006.

VisTrails: Visualization meets Data Management
 University of North Carolina at Chapel Hill, February 2nd, 2007.
 TU-Kaiserlautern, June 23rd, 2006.

Managing Complex Visualizations
 Harmon Pro Group Tech Conference, February 22nd, 2006.

Dynamic Level-Of-Detail Rendering of Unstructured Meshes
 Dagstuhl Scientific Visualization, June 9th, 2005.

Point-Set Surfaces: An Update and Recent Work,
 Lawrence Livermore National Laboratory, May 20th, 2005.

GPU-Based Scientific Visualization
 University of Texas at Dallas, March 28th, 2005.
 IBM T. J. Watson Research Center, November 22nd, 2004.
 Brigham Young University, September 23rd, 2004.

GPU-Based Unstructured Volume Rendering
 Technische Universität München, June 9th, 2004.
 University of Stuttgart, June 8th, 2004.

Using Points for Rendering and Modeling Surfaces
 UC-Davis, March 5th, 2003.

UC-Berkeley, March 20th, 2003.
 Dagstuhl Scientific Visualization, June 5th, 2003.
 DIMACS Surface Reconstruction Workshop , April 30th, 2003.

Massive Polygonal Rendering
 Arctic Region Supercomputer Center, Alaska, August 6th, 2003.

Direct Volume Rendering Techniques for Unstructured Grids
 UC-Santa Barbara, March 13th, 2002.
 Univ. of Pittsburgh, April 4th, 2002.
 University of Miami, March 21st, 2002.
 Rutgers University, March 8th, 2002.
 UMass-Amherst, February 8th, 2002.
 OGI-OHSU, January 31st, 2002.

External Memory Algorithms for Scientific Visualization
 Michigan State University, December 3rd, 2001.

Surface Reconstruction Algorithms
 Princeton University, November 12th, 2001.

The ZSWEEP Algorithm for Rendering Irregular Grids
 LLNL, Livermore, October 12th, 2001.

Point-Set Surfaces
 LLNL, Livermore, October 10th, 2001.

Rendering Irregular Grids
 IMPA-Brazil, February 13th, 2001.

Towards Acquiring and Rendering Real-World Environments
 AT&T Cambridge, Sept. 3rd, 2001.

Challenges in Scientific Visualization
 New York University, New York City, December 17th, 1999.

Sorting Polyhedra and Applications
 Bell Labs, Murray Hill, New Jersey, October 28th, 1999.
 CNUCE – CNR, Pisa, Italy, September 3rd, 1999.
 Rutgers University, New Jersey, April 22nd, 1999.

Service

Internal Service

New York University

- Tenure and Promotion Committee, School of Engineering
 - Chair (2015–)
 - Member (2011–)
- Advisory Committee, NYU Libraries
- Moore-Sloan Data Science Initiative
 - Executive committee
 - Software Tools Workgroup (SWG) leader
- Interim Curriculum Advisory Committee, CUSP (2011)

- Curriculum Advisory Committee, Center for Data Science (2011)

School of Computing, University of Utah

- Director (founding), Graphics and Visualization Track, 2004–2009.
- Curriculum Committee
 - Member, 2004–2010.
 - Co-chair, 2007–2008.
- Member of Graduate Admissions Committee, 2005–2009.
- Member of Faculty Recruiting Committee (Theory sub-committee), 2006.
- Coach, Utah Programming Team, 2004–2006
 - We won 2nd place in the 2006 ACM Rocky Mountain Regional Contest, and best in Utah.
 - We won 2nd place in the 2005 ACM Rocky Mountain Regional Contest, and best in Utah.
 - (With E. Praun.) We won 3rd place in the 2004 ACM Rocky Mountain Regional Contest, and best in Utah.

Scientific Computing and Imaging (SCI) Institute, University of Utah

- Associate Director, January 2008–May 2009.
- Member of Graduate Recruiting Committee, 2005–2010.
- Member of IT Committee, 2005.

External Service

Journal Editorships

- Editorial Board, ACM Transactions on Spatial Algorithms and Systems (TSAS) (2013–).
- Associate Editor, IEEE Transactions on Big Data (2015–).
- Associate Editor, Computer Graphics Forum (2013–).
- Associate Editor, The Visual Computer (2011–).
- Co-Editor, Visualization Corner, Computing in Science and Engineering magazine (2007–2015).
- Associate Editor, Graphical Models (GMOD) (2010–2014).
- Editorial Board, Computer and Graphics (2008–2014).
- Guest Editor: Computing in Science and Engineering theme issue on Computational Provenance, 2008.
- Associate Editor, IEEE Transactions on Visualization and Computer Graphics (2002–2006).

- Guest Editor: IEEE Transactions on Visualization and Computer Graphics issue on IEEE Visualization 2006.
- Guest Editor: IEEE Transactions on Visualization and Computer Graphics issue on IEEE Visualization 2005.

Conference Chairing and Organization

- Best paper award selection committee, IEEE SciVIS 2014.
- Steering Committee, IEEE SciVIS (2013–).
- Best paper award selection committee, LDAH 2013.
- Program Co-chair, LDAH 2013.
- Program Co-chair, SIBGRAPI 2013.
- Program Co-chair, IEEE Symposium on Large-Scale Data Analysis and Visualization (LDAH) 2011.
- General Co-chair, IEEE VisWeek 2010.
- Conference chair, IEEE Visualization 2010.
- Co-organizer, CSCW 2010 workshop on “The Changing Dynamics of Scientific Collaborations”
- Visualization area co-chair, 5th International Symposium on Visual Computing, 2009.
- Co-organizer, CHI 2009 workshop on “The Changing Face of Digital Science: Workshop on New Practices in Scientific Collaborations.”
- Papers Co-chair, IEEE Visualization 2006.
- Papers Co-chair, IEEE Visualization 2005.
- Best paper award selection committee, IEEE Visualization 2006.
- Best paper award selection committee (chair), IEEE Visualization 2005.
- Best paper award selection committee, IEEE Visualization 2004.
- Papers Co-chair, IEEE/SIGGRAPH Symposium on Volume Visualization and Graphics 2004.
- Co-chair, IEEE Parallel & Large-Data Visualization & Graphics Symposium 2003.
- Co-organizer, DIMACS Implementation Challenge on Surface Reconstruction, 2003.
- Co-organizer, DIMACS Workshop on Visualization and Data Mining, 2002.

Reviewing and Other Committee Participation

- Reappointment committee for the Editor-in-Chief of IEEE Transactions on Visualization and Computer Graphics (2015).
- Reappointment committee for the Editor-in-Chief of IEEE/AIP Computing in Science and Engineering (2010).
- Search committee for the Editor-in-Chief of IEEE Computer Graphics and Applications (2009).
- NIH Panelist for Software Maintenance Panel (twice).
- NSF Panelist, 2002-04, 2006, 2007, 2008.
- Member, geometry subcommittee, NIFTI Data Format Working Group of the National Institutes of Health, 2005–.
- Member, MPEG-4 3D Model Coding (3DMC) standardization committee, 1998-9.
- Symposium Committee, ACM/IEEE Volume Visualization 2000.
- Reviewer for: National Science Foundation, MacArthur Fellows Program, Dutch National Science Foundation (NWO), ACM SIGGRAPH (papers and courses), IEEE Visualization, ACM SIGMOD, ACM/IEEE Volume Visualization, IEEE Transactions on Visualization and Computer Graphics, IEEE Computer Graphics and Applications, Eurographics, Visual Computer, IEEE Transactions on Networking, Graphics Interface, Symposium on Interactive 3D, and several other conferences, journals, and funding agencies.
- Book reviewer for Morgan-Kaufmann Publishers, AK Peters.
- Member of ACM, IEEE, Eurographics.

Program Committees (125+)

EG 2017
Pacific Graphics 2016
SPM 2016
SGP 2016
Eurovis 2016
EG 2016 Papers
I3D 2016 Papers
LDAV 2015
VAST 2015
Pacific Graphics 2015
Eurovis 2015
I3D 2015 Papers
SMI 2014
SciVIS 2014 Papers
VAST 2014 Papers
EuroVis Short Papers (EuroVis 2014 conference)
IEEE International Congress on Big Data
SIBGRAPI 2014 (27th Conference on Graphics, Patterns and Images).

PG 2014
 GMP 2014
 EuroVis 2014 Papers
 I3D 2014
 The 2nd International Workshop on Urban Computing (UrbComp 2013)
 IEEE Visualization 2013
 IEEE Big Data 2013
 Third Joint 3DIM/3DPVT Conference (3D Imaging, Modeling, Processing, Visualization & Transmission)
 SIAM Conference on Geometric and Physical Modeling (GD/SPM13)
 Shape Modeling International 2013 (SMI'13)
 EuroVis Workshop on Reproducibility, Verification, and Validation in Visualization (EuroRVVV)
 EuroVis 2013 Short Papers
 Symposium on Geometry Processing 2013
 International Conference on 3D Web Technology (Web3D 2013)
 Pacific Graphics 2013
 Symposium on Interactive 3D Graphics and Games 2013 (I3D 2013)
 Large Data Analysis and Visualization Symposium (LDAV) 2012
 IEEE Visualization 2012
 International Conference on 3D Web Technology (Web3D 2012)
 2012 Symposium on Solid and Physical Modeling
 EuroVis 2012 Short Papers
 Geometric Modeling and Processing (GMP 2012)
 Short papers program – Eurographics 2012
 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D) 2012
 Papers program – Eurographics 2012
 IEEE International Conference on Shape Modeling and Applications (SMI) 2012
 ICDE 2012 “Research – Scientific data and data visualization” track.
 2011 ACM International Web3D Conference
 2011 SIAM/ACM Joint Conference on Geometric and Physical Modeling
 Symposium on Geometry Processing 2011
 3DIMPVT 2011
 EuroVis 2011
 Eurographics Symposium on Parallel Graphics and Visualization 2011 (EGPGV 2011)
 10th Eurographics Parallel Graphics and Visualisation (EGPGV) Symposium
 XXI Brazilian Symp on Computer Graphics and Image Processing (SIBGRAPI) 2010
 2010 SIAM/ACM Joint Conference on Geometric and Physical Modeling
 EuroVis 2010
 International Meeting High Performance Computing for Computational Science (VECPAR'10)
 Symposium on Geometry Processing 2010
 IEEE International Conference on Shape Modeling and Applications (SMI) 2010
 Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT) 2010
 First International Workshop on Semantic Web and Provenance Management 2009 (SWPM)
 XX Brazilian Symp on Computer Graphics and Image Processing (SIBGRAPI) 2009
 1st International Workshop on Provenance in Practice 2009 (PPW09)
 Symposium on Geometry Processing 2009
 VizMining 2009 Workshop at the 2009 SIAM International Conference on Data Mining
 EuroVis 2009
 Eurographics 2009 Symposium on Parallel Graphics and Visualization (EGPGV'09)

2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling
 IEEE International Conference on Shape Modeling and Applications (SMI) 2009
 ACM Multimedia 2008 Technical Demonstrations
 Knowledge-Assisted Visualization (KAV) 2008
 International Symposium on Volume Graphics 2008 (VG08)
 XIX Brazilian Symp on Computer Graphics and Image Processing (SIBGRAPI) 2008
 Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT) 2008
 2nd International Provenance and Annotation Workshop (IPAW 2008)
 ACM SIGGRAPH 2008 Papers Program
 ACM Solid and Physical Modeling Symposium (SPM) 2008
 IEEE International Conference on Shape Modeling and Applications (SMI) 2008
 EuroVis 2008
 International Conference on Computer Animation and Social Agents (CASA) 2008
 Symposium on Geometry Processing 2008
 Knowledge-Assisted Visualization (KAV) 2007
 Pacific Graphics 2007
 IEEE Visualization 2007
 6th International Workshop on Volume Graphics (VG 2007)
 3rd International Symposium on Visual Computing (ISVC 07)
 ACM SIGGRAPH 2007 Sketches & Posters Program
 XVIII Brazilian Symp on Computer Graphics and Image Processing (SIBGRAPI) 2007
 7th Eurographics Workshop on Parallel Graphics and Visualization (EGPGV), 2007
 Symposium on Geometry Processing 2007
 Eurographics 2007
 2nd International Symposium on Visual Computing (ISVC 06)
 5th International Workshop on Volume Graphics (VG 2006)
 3rd Ibero-American Symposium on Computer Graphics (SIACG 2006)
 Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT) 2006
 Computer Graphics International 2006
 Symposium on Point-Based Graphics 2006
 Shape Modelling International 2006
 Symposium on Geometry Processing 2006
 XVIII Brazilian Symp on Computer Graphics and Image Processing (SIBGRAPI) 2006
 6th Eurographics Workshop on Parallel Graphics and Visualization (EGPGV), 2006
 Pacific Graphics 2005
 XVII Brazilian Symp on Computer Graphics and Image Processing (SIBGRAPI) 2005
 Symposium on Point-Based Graphics 2005
 Symposium on Geometry Processing 2005
 International Workshop on Volume Graphics 2005
 Shape Modelling International 2005
 7th Brazilian Symposium on Virtual Reality 2004
 XVII Brazilian Symp on Computer Graphics and Image Processing (SIBGRAPI) 2004
 Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT) 2004
 Pacific Graphics 2004
 Second Symposium on Geometry Processing 2004
 Fifth Eurographics Symposium on Parallel Graphics and Visualization 2004
 Symposium on Point-Based Graphics 2004
 Solid Modelling International 2004

Sixth Brazilian Virtual Reality Symposium (SVR) 2003
IEEE Visualization 2003
Symposium on Geometry Processing, 2003
IEEE Visualization 2002
ACM/IEEE Volume Visualization 2002
Fifth Brazilian Virtual Reality Symposium (SVR) 2002
International Workshop on 3D Digitization (3DD) 2002
Eurographics Workshop on Parallel Graphics and Visualization 2002
1st Ibero-American Symposium on Computer Graphics 2002
IEEE Visualization 2001
IEEE Parallel & Large-Data Visualization & Graphics Symposium 2001
International Workshop On Volume Graphics 2001
IEEE Visualization 2000

INFORMATION DISCLOSURE STATEMENT BY APPLICANT (Not for submission under 37 CFR 1.99)	Application Number		RE of Patent No. 9,104,842	
	Filing Date		2007-08-24	
	First Named Inventor	Scott A. Moskowitz		
	Art Unit	TBD		
	Examiner Name	TBD		
	Attorney Docket Number			

U.S.PATENTS						
Examiner Initial*	Cite No	Patent Number	Kind Code ¹	Issue Date	Name of Patentee or Applicant of cited Document	Pages,Columns,Lines where Relevant Passages or Relevant Figures Appear
	1	5933497		1999-08-03	Robert Carl Beetcher	
	2	5935243		1999-08-10	Takayuki Hasebe	

If you wish to add additional U.S. Patent citation information please click the Add button.

U.S.PATENT APPLICATION PUBLICATIONS						
Examiner Initial*	Cite No	Publication Number	Kind Code ¹	Publication Date	Name of Patentee or Applicant of cited Document	Pages,Columns,Lines where Relevant Passages or Relevant Figures Appear
	1					

If you wish to add additional U.S. Published Application citation information please click the Add button.

FOREIGN PATENT DOCUMENTS								
Examiner Initial*	Cite No	Foreign Document Number ³	Country Code ² i	Kind Code ⁴	Publication Date	Name of Patentee or Applicant of cited Document	Pages,Columns,Lines where Relevant Passages or Relevant Figures Appear	T ⁵
	1	05-334072	JP		1993-12-17	Robert Carl Beetcher		<input type="checkbox"/>
	2	9726732	WO		1997-07-24	Scott A. Moskowitz		<input type="checkbox"/>

INFORMATION DISCLOSURE STATEMENT BY APPLICANT (Not for submission under 37 CFR 1.99)	Application Number	RE of Patent No. 9,104,842
	Filing Date	2007-08-24
	First Named Inventor	Scott A. Moskowitz
	Art Unit	TBD
	Examiner Name	TBD
	Attorney Docket Number	

If you wish to add additional Foreign Patent Document citation information please click the Add button			
NON-PATENT LITERATURE DOCUMENTS			
Examiner Initials*	Cite No	Include name of the author (in CAPITAL LETTERS), title of the article (when appropriate), title of the item (book, magazine, journal, serial, symposium, catalog, etc), date, pages(s), volume-issue number(s), publisher, city and/or country where published.	T ⁵
	1		<input type="checkbox"/>
If you wish to add additional non-patent literature document citation information please click the Add button			
EXAMINER SIGNATURE			
Examiner Signature		Date Considered	
*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609. Draw line through a citation if not in conformance and not considered. Include copy of this form with next communication to applicant.			
<small>¹ See Kind Codes of USPTO Patent Documents at www.USPTO.GOV or MPEP 901.04. ² Enter office that issued the document, by the two-letter code (WIPO Standard ST.3). ³ For Japanese patent documents, the indication of the year of the reign of the Emperor must precede the serial number of the patent document. ⁴ Kind of document by the appropriate symbols as indicated on the document under WIPO Standard ST.16 if possible. ⁵ Applicant is to place a check mark here if English language translation is attached.</small>			

INFORMATION DISCLOSURE STATEMENT BY APPLICANT (Not for submission under 37 CFR 1.99)	Application Number	RE of Patent No. 9,104,842
	Filing Date	2007-08-24
	First Named Inventor	Scott A. Moskowitz
	Art Unit	TBD
	Examiner Name	TBD
	Attorney Docket Number	

CERTIFICATION STATEMENT

Please see 37 CFR 1.97 and 1.98 to make the appropriate selection(s):

That each item of information contained in the information disclosure statement was first cited in any communication from a foreign patent office in a counterpart foreign application not more than three months prior to the filing of the information disclosure statement. See 37 CFR 1.97(e)(1).

OR

That no item of information contained in the information disclosure statement was cited in a communication from a foreign patent office in a counterpart foreign application, and, to the knowledge of the person signing the certification after making reasonable inquiry, no item of information contained in the information disclosure statement was known to any individual designated in 37 CFR 1.56(c) more than three months prior to the filing of the information disclosure statement. See 37 CFR 1.97(e)(2).

See attached certification statement.

The fee set forth in 37 CFR 1.17 (p) has been submitted herewith.

A certification statement is not submitted herewith.

SIGNATURE

A signature of the applicant or representative is required in accordance with CFR 1.33, 10.18. Please see CFR 1.4(d) for the form of the signature.

Signature	/Joseph F. Edell/	Date (YYYY-MM-DD)	2018-05-16
Name/Print	Joseph F. Edell	Registration Number	67,625

This collection of information is required by 37 CFR 1.97 and 1.98. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 1 hour to complete, including gathering, preparing and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. **SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.**

Privacy Act Statement

The Privacy Act of 1974 (P.L. 93-579) requires that you be given certain information in connection with your submission of the attached form related to a patent application or patent. Accordingly, pursuant to the requirements of the Act, please be advised that: (1) the general authority for the collection of this information is 35 U.S.C. 2(b)(2); (2) furnishing of the information solicited is voluntary; and (3) the principal purpose for which the information is used by the U.S. Patent and Trademark Office is to process and/or examine your submission related to a patent application or patent. If you do not furnish the requested information, the U.S. Patent and Trademark Office may not be able to process and/or examine your submission, which may result in termination of proceedings or abandonment of the application or expiration of the patent.

The information provided by you in this form will be subject to the following routine uses:

1. The information on this form will be treated confidentially to the extent allowed under the Freedom of Information Act (5 U.S.C. 552) and the Privacy Act (5 U.S.C. 552a). Records from this system of records may be disclosed to the Department of Justice to determine whether the Freedom of Information Act requires disclosure of these records.
2. A record from this system of records may be disclosed, as a routine use, in the course of presenting evidence to a court, magistrate, or administrative tribunal, including disclosures to opposing counsel in the course of settlement negotiations.
3. A record in this system of records may be disclosed, as a routine use, to a Member of Congress submitting a request involving an individual, to whom the record pertains, when the individual has requested assistance from the Member with respect to the subject matter of the record.
4. A record in this system of records may be disclosed, as a routine use, to a contractor of the Agency having need for the information in order to perform a contract. Recipients of information shall be required to comply with the requirements of the Privacy Act of 1974, as amended, pursuant to 5 U.S.C. 552a(m).
5. A record related to an International Application filed under the Patent Cooperation Treaty in this system of records may be disclosed, as a routine use, to the International Bureau of the World Intellectual Property Organization, pursuant to the Patent Cooperation Treaty.
6. A record in this system of records may be disclosed, as a routine use, to another federal agency for purposes of National Security review (35 U.S.C. 181) and for review pursuant to the Atomic Energy Act (42 U.S.C. 218(c)).
7. A record from this system of records may be disclosed, as a routine use, to the Administrator, General Services, or his/her designee, during an inspection of records conducted by GSA as part of that agency's responsibility to recommend improvements in records management practices and programs, under authority of 44 U.S.C. 2904 and 2906. Such disclosure shall be made in accordance with the GSA regulations governing inspection of records for this purpose, and any other relevant (i.e., GSA or Commerce) directive. Such disclosure shall not be used to make determinations about individuals.
8. A record from this system of records may be disclosed, as a routine use, to the public after either publication of the application pursuant to 35 U.S.C. 122(b) or issuance of a patent pursuant to 35 U.S.C. 151. Further, a record may be disclosed, subject to the limitations of 37 CFR 1.14, as a routine use, to the public if the record was filed in an application which became abandoned or in which the proceedings were terminated and which application is referenced by either a published application, an application open to public inspections or an issued patent.
9. A record from this system of records may be disclosed, as a routine use, to a Federal, State, or local law enforcement agency, if the USPTO becomes aware of a violation or potential violation of law or regulation.



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
 United States Patent and Trademark Office
 Address: COMMISSIONER FOR PATENTS
 P.O. Box 1450
 Alexandria, Virginia 22313-1450
 www.uspto.gov

BIB DATA SHEET

CONFIRMATION NO. 7638

SERIAL NUMBER 90/014,138	FILING or 371(c) DATE 05/16/2018 RULE	CLASS 713	GROUP ART UNIT 3992	ATTORNEY DOCKET NO.		
APPLICANTS						
INVENTORS 9104842, Residence Not Provided; WISTARIA TRADING, LTD., HAMILTON, BERMUDA; FISCH SIGLER, LLP (3RD PTY REQ.), WASHINGTON, DC; FISCH SIGLER, LLP, WASHINGTON, DC						
** CONTINUING DATA ***** This application is a REX of 11/895,388 08/24/2007 PAT 9104842 which is a DIV of 10/602,777 06/25/2003 PAT 7664263 which is a CON of 09/046,627 03/24/1998 PAT 6598162						
** FOREIGN APPLICATIONS *****						
** IF REQUIRED, FOREIGN FILING LICENSE GRANTED **						
Foreign Priority claimed <input type="checkbox"/> Yes <input type="checkbox"/> No		<input type="checkbox"/> Met after Allowance		STATE OR COUNTRY	SHEETS DRAWINGS	TOTAL CLAIMS
35 USC 119(a-d) conditions met <input type="checkbox"/> Yes <input type="checkbox"/> No		Initials				14
Verified and Acknowledged		Examiner's Signature				9
ADDRESS NEIFELD IP LAW, PC 5400 Shawnee Road Suite 310 ALEXANDRIA, VA 22312-2300 UNITED STATES						
TITLE DATA PROTECTION METHOD AND DEVICE						
FILING FEE RECEIVED 12000	FEES: Authority has been given in Paper No. _____ to charge/credit DEPOSIT ACCOUNT No. _____ for following:				<input type="checkbox"/> All Fees <input type="checkbox"/> 1.16 Fees (Filing) <input type="checkbox"/> 1.17 Fees (Processing Ext. of time) <input type="checkbox"/> 1.18 Fees (Issue) <input type="checkbox"/> Other _____ <input type="checkbox"/> Credit	

Patent Assignment Abstract of Title

Total Assignments: 1

Application #: 11895388

Filing Dt: 08/24/2007

Patent #: 9104842

Issue Dt: 08/11/2015

PCT #: NONE

Intl Reg #:

Publication #: US20080016365

Pub Dt: 01/17/2008

Inventor: Scott A. Moskowitz

Title: Data protection method and device

Assignment: 1

Reel/Frame: 036342 / 0953

Received: 08/17/2015

Recorded: 08/17/2015

Mailed: 08/18/2015

Pages: 9

Conveyance: ASSIGNMENT OF ASSIGNORS INTEREST (SEE DOCUMENT FOR DETAILS).

Assignor: MOSKOWITZ, SCOTT A.

Exec Dt: 08/14/2015


Assignee: WISTARIA TRADING LTD

CLARENDON HOUSE, 2 CHURCH STREET
HAMILTON, BERMUDA HM 11

Correspondent: BRUCE T. MARGULIES

4813-B EISENHOWER AVE
ALEXANDRIA, VA 22304

Search Results as of: 05/21/2018 08:42 AM

If you have any comments or questions concerning the data displayed, contact PRD / Assignments at 571-272-3350  v.2.6
Web interface last modified: Jun 26, 2017 v.2.6



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

REEXAM CONTROL NUMBER	FILING OR 371 (c) DATE	PATENT NUMBER
90/014,138	05/16/2018	9104842

FISCH SIGLER, LLP
5301 WISCONSIN AVENUE, NW
FOURTH FLOOR
WASHINGTON, DC 20015

**CONFIRMATION NO. 7638
REEXAMINATION REQUEST
NOTICE**



Date Mailed: 05/22/2018

NOTICE OF REEXAMINATION REQUEST FILING DATE

(Third Party Requester)

Requester is hereby notified that the filing date of the request for reexamination is 05/16/2018, the date that the filing requirements of 37 CFR § 1.510 were received.

A decision on the request for reexamination will be mailed within three months from the filing date of the request for reexamination. (See 37 CFR 1.515(a)).

A copy of the Notice is being sent to the person identified by the requester as the patent owner. Further patent owner correspondence will be the latest attorney or agent of record in the patent file. (See 37 CFR 1.33). Any paper filed should include a reference to the present request for reexamination (by Reexamination Control Number).

cc: Patent Owner
31518
NEIFELD IP LAW, PC
5400 Shawnee Road
Suite 310
ALEXANDRIA, VA 22312-2300

/rbell/

Legal Instruments Examiner
Central Reexamination Unit 571-272-7705; FAX No. 571-273-9900



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

Table with 3 columns: REEXAM CONTROL NUMBER (90/014,138), FILING OR 371 (c) DATE (05/16/2018), PATENT NUMBER (9104842)

31518
NEIFELD IP LAW, PC
5400 Shawnee Road
Suite 310
ALEXANDRIA, VA 22312-2300

CONFIRMATION NO. 7638
REEXAM ASSIGNMENT NOTICE



Date Mailed: 05/22/2018

NOTICE OF ASSIGNMENT OF REEXAMINATION REQUEST

The above-identified request for reexamination has been assigned to Art Unit 3992. All future correspondence to the proceeding should be identified by the control number listed above and directed to the assigned Art Unit.

A copy of this Notice is being sent to the latest attorney or agent of record in the patent file or to all owners of record. (See 37 CFR 1.33(c)). If the addressee is not, or does not represent, the current owner, he or she is required to forward all communications regarding this proceeding to the current owner(s). An attorney or agent receiving this communication who does not represent the current owner(s) may wish to seek to withdraw pursuant to 37 CFR 1.36 in order to avoid receiving future communications. If the address of the current owner(s) is unknown, this communication should be returned within the request to withdraw pursuant to Section 1.36.

NOTICE OF USPTO EX PARTE REEXAMINATION PATENT OWNER STATEMENT WAIVER PROGRAM

The USPTO has implemented a pilot program where, after a reexamination proceeding has been granted a filing date and before the examiner begins his or her review, the patent owner may orally waive the right to file a patent owner's statement. See "Pilot Program for Waiver of Patent Owner's Statement in Ex Parte Reexamination Proceedings," 75 FR 47269 (August 5, 2010). One goal of the pilot program is to reduce the pendency of reexamination proceedings and improve the efficiency of the reexamination process.

Ordinarily when ex parte reexamination is ordered, the USPTO must wait until after the receipt of the patent owner's statement and the third party requester's reply, or after the expiration of the time period for filing the statement and reply (a period that can be as long as 5 to 6 months), before mailing a first determination of patentability. The USPTO's first determination of patentability is usually a first Office action on the merits or a Notice of Intent to Issue Reexamination Certificate (NIRC).

Under the pilot program, the patent owner's oral waiver allows the USPTO to act on the first determination of patentability immediately after determining that reexamination will be ordered, and in a suitable case issue the reexamination order and the first determination of patentability (which could be a NIRC if the claims under reexamination are confirmed) at the same time.

Benefits to the Patent Owner for participating in this pilot program include reduction in pendency.

To participate in this pilot program, Patent Owners may contact the USPTO's Central Reexamination Unit (CRU) at 571-272-7705. The USPTO will make the oral waiver of record in the reexamination file in an interview summary and a copy will be mailed to the patent owner and any third party requester.

cc: Third Party Requester(if any)
FISCH SIGLER, LLP
5301 WISCONSIN AVENUE, NW
FOURTH FLOOR
WASHINGTON, DC 20015

/rbell/

Legal Instruments Examiner
Central Reexamination Unit 571-272-7705; FAX No. 571-273-9900



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

Table with 5 columns: APPLICATION NO., FILING DATE, FIRST NAMED INVENTOR, ATTORNEY DOCKET NO., CONFIRMATION NO.
90/014,138 05/16/2018 9104842 7638

31518 7590 05/22/2018
NEIFELD IP LAW, PC
5400 Shawnee Road
Suite 310
ALEXANDRIA, VA 22312-2300

EXAMINER
BONSHOCK, DENNIS G

ART UNIT PAPER NUMBER
3992

MAIL DATE DELIVERY MODE
05/22/2018 PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.



UNITED STATES PATENT AND TRADEMARK OFFICE

Commissioner for Patents
United States Patents and Trademark Office
P.O.Box 1450
Alexandria, VA 22313-1450
www.uspto.gov

THIRD PARTY REQUESTER'S CORRESPONDENCE ADDRESS

FISCH SIGLER, LLP
5301 WISCONSIN AVENUE, NW
FOURTH FLOOR
WASHINGTON, DC 20015

Date:

MAY 22 2018

EX PARTE REEXAMINATION COMMUNICATION TRANSMITTAL FORM

REEXAMINATION CONTROL NO. : 90014138
PATENT NO. : 9104842
ART UNIT : 3992

Enclosed is a copy of the latest communication from the United States Patent and Trademark Office in the above identified ex parte reexamination proceeding (37 CFR 1.550(f)).

Where this copy is supplied after the reply by requester, 37 CFR 1.535, or the time for filing a reply has passed, no submission on behalf of the ex parte reexamination requester will be acknowledged or considered (37 CFR 1.550(g)).

Ex Parte Reexamination Interview Summary – Pilot Program for Waiver of Patent Owner's Statement	Control No.	Patent For Which Reexamination is Requested
	90/014,138	9,104,842
	Examiner	Art Unit

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address. --

All participants (USPTO official and patent owner):

- (1) Patricia Martin (3)
(2) Richard Neifeld, 35299 (4)

Date of Telephonic Interview: 5/22/18.

The USPTO official requested waiver of the patent owner's statement pursuant to the pilot program for waiver of patent owner's statement in *ex parte* reexamination proceedings.*

- The patent owner **agreed** to waive its right to file a patent owner's statement under 35 U.S.C. 304 in the event reexamination is ordered for the above-identified patent.
- The patent owner **did not agree** to waive its right to file a patent owner's statement under 35 U.S.C. 304 at this time.

The patent owner is not required to file a written statement of this telephone communication under 37 CFR 1.560(b) or otherwise. However, any disagreement as to this interview summary must be brought to the immediate attention of the USPTO, and no later than one month from the mailing date of this interview summary. Extensions of time are governed by 37 CFR 1.550(c).

*For more information regarding this pilot program, see *Pilot Program for Waiver of Patent Owner's Statement in Ex Parte Reexamination Proceedings*, 75 Fed. Reg. 47269 (August 5, 2010), available on the USPTO Web site at <http://www.uspto.gov/patents/law/notices/2010.jsp>.

- USPTO personnel were unable to reach the patent owner.

The patent owner may contact the USPTO personnel at the telephone number provided below if the patent owner decides to waive the right to file a patent owner's statement under 35 U.S.C. 304.

/Patricia Martin/
Paralegal Specialist,
Central Reexamination Unit
Signature and telephone number of the USPTO official who contacted or attempted to contact the patent owner. 571-272-5004

cc: Requester (if third party requester)

Litigation Search Report CRU 3999

Reexam Control No. 90/014,138

TO: Dennis Bonshock
Location: CRU
Art Unit: 3992
Date: May 24, 2018

From: Patricia Martin
Paralegal Specialist
Location: CRU 3999
Phone: (571) 272-7705

U.S. Patent Number: 9,104,842

Search Notes

- 1) I performed a search on the patent in Lexis Court Link for any open dockets or closed cases.
- 2) I performed a Key Cite Search in Westlaw, which retrieves all history on the patent including any litigation.
- 3) I performed a search in Lexis in the Federal Courts and Administrative Materials databases for any cases found.
- 4) I performed a search in Lexis in the IP Journal and Periodicals database for any articles on the patent.
- 5) I performed a search in Lexis in the news databases for any articles about the patent or any articles about litigation on this patent.

Litigation was found involving:

Citing References (17)

Treatment	Title	Date	Type	Depth	Headnote(s)
Examined by	1. Juniper Networks, Inc.'s Answer, Affirmative Defenses, and Counterclaims to Plaintiff Blue Spike, LLC's Amended Complaint <small>Out Of Page</small> BLUE SPIKE, LLC, Plaintiff, v. JUNIPER NETWORKS, INC., Defendant. 2017 WL 3587937, *1+ , E.D.Tex. (Trial Pleading)	July 20, 2017	Petition		—
Examined by	2. First Amended Complaint for Patent Infringement <small>Out Of Page</small> BLUE SPIKE, LLC, Plaintiff, v. JUNIPER NETWORKS, INC., Defendant. 2017 WL 3587934, *1+ , E.D.Tex. (Trial Pleading)	May 12, 2017	Petition		—
Cited by	3. Defendants' Responsive Claim Construction Brief <small>Out Of Page</small> BLUE SPIKE, LLC, Plaintiff, v. BLU PRODUCTS, INC., et al., Defendants. Blue Spike, LLC, Plaintiff, v. Toshiba America Information Systems, Inc. & Tosh... 2017 WL 2773187, *1+ , E.D.Tex. (Trial Motion, Memorandum and Affidavit)	Apr. 21, 2017	Motion		—
Cited by	4. Defendants' Responsive Claim Construction Brief <small>Out Of Page</small> BLUE SPIKE, LLC, Plaintiff, v. BLU PRODUCTS, INC., et al., Defendants. Blue Spike, LLC, Plaintiff, v. Toshiba America Information Systems, Inc. & Tosh... 2017 WL 1830410, *1+ , E.D.Tex. (Trial Motion, Memorandum and Affidavit)	Apr. 10, 2017	Motion		—
Mentioned by	5. Methods, systems and devices for packet watermarking and efficient provisioning of bandwidth LitAlert P2018-19-21	May 04, 2018	Lit Alert		—
Mentioned by	6. Methods, systems and devices for packet watermarking and efficient provisioning of bandwidth LitAlert P2018-18-10	Apr. 27, 2018	Lit Alert		—
Mentioned by	7. Methods, systems and devices for packet watermarking and efficient provisioning of bandwidth LitAlert P2018-17-25	Apr. 20, 2018	Lit Alert		—
Mentioned by	8. Methods, systems and devices for packet watermarking and efficient provisioning of bandwidth LitAlert P2017-02-13	Jan. 06, 2017	Lit Alert		—
—	9. STEGA CIPHER PROTECTION FOR COPY PROTECTION OF COMPUTER PROGRAMS INVOLVES ENCODING SOME ESSENTIAL CODE RESOURCES AS WATERMARKS IN DATA RESOURCES AND USING RANDOM DYNAMIC MEMORY SHUFFLING <small>Out Of Page</small> DWPI 1997-385615	Jan. 17, 1996	DWPI	—	—

List of 17 Citing References for DATA PROTECTION METHOD AND DEVICE

Treatment	Title	Date	Type	Depth	Headnote(s)
—	10. DIGITAL INFORMATION E.G. STILL IMAGE, COPY PROTECTION METHOD FOR USE IN E.G. PLUG-IN DIGITAL PLAYER, INVOLVES GENERATING ENCODED DIGITAL INFORMATION, AND INCLUDING DIGITAL SAMPLE AND ENCODED FORMAT INFORMATION <small>Out Of File</small> DWPI 2008-B91679	Jan. 17, 1996	DWPI	—	—
—	11. RF 036342/0953 <small>Out Of File</small>	Aug. 17, 2015	Assignments	—	—
—	12. Blue Spike, LLC v. Altice USA, Inc.	May 18, 2018	Docket Summaries	—	—
—	13. Blue Spike, LLC v. Charter Communications, Inc.	May 04, 2018	Docket Summaries	—	—
—	14. Blue Spike, LLC v. Comcast Corporation ET AL	Apr. 27, 2018	Docket Summaries	—	—
—	15. Blue Spike, LLC v. Cequel Communications, LLC ET AL	Apr. 20, 2018	Docket Summaries	—	—
—	16. Blue Spike, LLC v. Juniper Networks, Inc.	Jan. 06, 2017	Docket Summaries	—	—
—	17. DATA PROTECTION METHOD AND DEVICE <small>Out Of File</small> US PAT APP 20160004875+ , U.S. PTO Application An apparatus and method for encoding and decoding additional information into a digital information in an integral manner. More particularly, the invention relates to a method and...	Jan. 07, 2016	Patents	—	—

LexisNexis® CourtLink®

Switch Client | My Briefcase | Order Runner Documents | Lexis Advance | Sign Out |
 Welcome, Patricia Martin

Single Search - with Terms and Connectors

Enter keywords - Search multiple dockets & documents Info

-
-
-
-
-
-
-
-
-

Search > Patent Search > Litigation involving patent 9,104,842

Click a docket number below to view a docket.

Patent Search Results

[Edit Search](#)

This search was run on 5/24/2018

Results: 5 cases and their patents, totaling 5 items.

[Printer Friendly List](#)
[Email List](#)
[Customize List](#)

Items 1 to 5 of 5									
<input type="checkbox"/>	Patent	Class	Subclass	Description	Court	Docket Number	Filed	Date Retrieved	
<input checked="" type="checkbox"/>	9,104,842	1	1	Blue Spike, Llc V. Juniper Networks, Inc.	US-DIS-TXED	6:17cv18	01/06/2017	05/21/2018	
<input checked="" type="checkbox"/>	9,104,842	1	1	Blue Spike, Llc V. Suddenlink Communications Et Al	US-DIS-TXED	6:18cv174	04/19/2018	04/21/2018	
<input checked="" type="checkbox"/>	9,104,842	1	1	Blue Spike, Llc V. Comcast Corporation D/B/A Xfinity Et Al	US-DIS-TXED	6:18cv181	04/27/2018	04/27/2018	
<input checked="" type="checkbox"/>	9,104,842	1	1	Blue Spike, Llc V. Charter Communications, Inc.	US-DIS-TXED	6:18cv195	05/04/2018	05/23/2018	
<input checked="" type="checkbox"/>	9,104,842	1	1	Blue Spike, Llc V. Altice Usa, Inc.	US-DIS-TXED	6:18cv223	05/18/2018	05/19/2018	

[Printer Friendly List](#)
[Email List](#)
[Customize List](#)

US District Court Civil Docket

U.S. District - Texas Eastern
(Tyler)

6:17cv16

Blue Spike, Llc v. Juniper Networks, Inc.

This case was retrieved from the court on Thursday, May 24, 2018

Date Filed: 01/06/2017
Assigned To: Magistrate Judge K. Nicole Mitchell
Referred To: Class Code: OPEN
Nature of suit: Patent (830) Closed:
Cause: Patent Infringement Statute: 35:271
Lead Docket: None Jury Demand: Both
Other Docket: 6:18cv00181 Demand Amount: \$0
6:18cv00195 NOS Description: Patent
6:18cv00223
Jurisdiction: Federal Question

Litigants

Attorneys

David Folsom
Mediator

Blue Spike, Llc
Plaintiff

Juniper Networks, Inc.
Defendant

Randall T Garteiser
ATTORNEY TO BE NOTICED
Garteiser Honea PLLC
119 W. Ferguson St.
Tyler , TX 75702
USA
903-705-7420
Fax: 888-908-4400
Email: Rgarteiser@ghiplaw.Com

Alan Michael Fisch
LEAD ATTORNEY; ATTORNEY TO BE NOTICED
Fisch Sigler LLP - DC
5301 Wisconsin Avenue Nw Fourth Floor
Washington , DC 20015
USA
202/362-3500
Email: Alan.Fisch@fischllp.Com

Desmond Jui
[Term: 03/02/2018]
Fisch Sigler LLP - San Jose
96 North Third Street Suite 260

San Jose , CA 95112
USA
650-362-8200
Email: Desmond.Jui@fischllp.Com

Jeffrey Saltman
ATTORNEY TO BE NOTICED
Fisch Sigler LLP - DC
5301 Wisconsin Avenue Nw Fourth Floor
Washington , DC 20015
USA
202/362-3640
Fax: 202/362-3501
Email: Jeffrey.Saltman@fischllp.Com

Kathleen Ryland
ATTORNEY TO BE NOTICED
Fisch Sigler LLP - DC
5301 Wisconsin Avenue Nw Fourth Floor
Washington , DC 20015
USA
202-362-3534
Email: Kathleen.Ryland@fischllp.Com

Melissa Richards Smith
ATTORNEY TO BE NOTICED
Gillam & Smith, LLP
303 South Washington Avenue
Marshall , TX 75670
USA
903-934-8450
Fax: 903-934-9257
Email: Melissa@gillamsmithlaw.Com

Roy William Sigler
ATTORNEY TO BE NOTICED
Fisch Sigler LLP - DC
5301 Wisconsin Avenue Nw Fourth Floor
Washington , DC 20015
USA
202-362-3500
Fax: 202-362-3501
Email: Bill.Sigler@fischllp.Com

Juniper Networks, Inc.
Counter Claimant

Alan Michael Fisch
LEAD ATTORNEY; ATTORNEY TO BE NOTICED
Fisch Sigler LLP - DC
5301 Wisconsin Avenue Nw Fourth Floor
Washington , DC 20015
USA
202/362-3500
Email: Alan.Fisch@fischllp.Com

Jeffrey Saltman
ATTORNEY TO BE NOTICED
Fisch Sigler LLP - DC
5301 Wisconsin Avenue Nw Fourth Floor
Washington , DC 20015
USA
202/362-3640
Fax: 202/362-3501

Email: Jeffrey.Saltman@fischllp.Com

Melissa Richards Smith
ATTORNEY TO BE NOTICED
Gillam & Smith, LLP
303 South Washington Avenue
Marshall , TX 75670
USA
903-934-8450
Fax: 903-934-9257
Email: Melissa@gillamsmithlaw.Com

Blue Spike, Llc
Counter Defendant

Randall T Garteiser
ATTORNEY TO BE NOTICED
Garteiser Honea PLLC
119 W. Ferguson St.
Tyler , TX 75702
USA
903-705-7420
Fax: 888-908-4400
Email: Rgarteiser@ghiplaw.Com

Date	#	Proceeding Text	Source
01/06/2017	1	COMPLAINT against Juniper Networks, Inc. (Filing fee \$ 400 receipt number 0540-6100195.), filed by Blue Spike, LLC. (Attachments: # 1 Exhibit A, # 2 Exhibit B, # 3 Exhibit C, # 4 Exhibit D, # 5 Exhibit E, # 6 Exhibit F, # 7 Exhibit G, # 8 Exhibit H, # 9 Exhibit I, # 10 Exhibit J, # 11 Exhibit K, # 12 Exhibit L, # 13 Exhibit M, # 14 Exhibit N, # 15 Civil Cover Sheet)(Garteiser, Randall) (Entered: 01/06/2017)	
01/09/2017		Judge Robert W. Schroeder, III and Magistrate Judge K. Nicole Mitchell added. (mll,) (Entered: 01/09/2017)	
01/09/2017	2	CASE REFERRED to Magistrate Judge K Nicole Mitchell. (mll,) (Entered: 01/09/2017)	
01/09/2017		In accordance with the provisions of 28 USC Section 636(c), you are hereby notified that a U.S. Magistrate Judge of this district court is available to conduct any or all proceedings in this case including a jury or non-jury trial and to order the entry of a final judgment. The form Consent to Proceed Before Magistrate Judge is available on our website. All signed consent forms, excluding pro se parties, should be filed electronically using the event Notice Regarding Consent to Proceed Before Magistrate Judge. (mll,) (Entered: 01/09/2017)	
01/12/2017	3	Notice of Filing of Patent/Trademark Form (AO 120). AO 120 mailed to the Director of the U.S. Patent and Trademark Office. (Garteiser, Randall) (Entered: 01/12/2017)	
02/17/2017	4	SUMMONS Issued as to Juniper Networks, Inc. Summons emailed to Plaintiff for service. (rlf) Modified on 2/17/2017 (rlf). (Entered: 02/17/2017)	
03/09/2017	5	SUMMONS Returned Executed by Blue Spike, LLC. Juniper Networks, Inc. served on 2/17/2017, answer due 3/10/2017. (mjc,) (Entered: 03/09/2017)	
03/10/2017	6	NOTICE of Attorney Appearance by Alan Michael Fisch on behalf of Juniper Networks, Inc. (Fisch, Alan) (Entered: 03/10/2017)	
03/10/2017	7	Defendant's Unopposed First Application for Extension of Time to Answer Complaint re Juniper Networks, Inc..(Fisch, Alan) (Entered: 03/10/2017)	
03/13/2017			

- Defendant's Unopposed First Application for Extension of Time to Answer Complaint is granted pursuant to Local Rule CV-12 for Juniper Networks, Inc. to 4/13/2017. 30 Days Granted for Deadline Extension.(mjc,) (Entered: 03/13/2017)
- 04/13/2017 8 Opposed MOTION for Extension of Time to File Answer by Juniper Networks, Inc.. (Attachments: # 1 Affidavit Declaration of S. Coonan, # 2 Text of Proposed Order)(Fisch, Alan) (Entered: 04/13/2017)
- 04/27/2017 9 RESPONSE in Opposition re 8 Opposed MOTION for Extension of Time to File Answer filed by Blue Spike, LLC . (Attachments: # 1 Text of Proposed Order, # 2 Declaration of Kirk Anderson, # 3 Exhibit 1 - Nokia asserting 32 patents, # 4 Exhibit 2 - Ericsson asserting 41 patents, # 5 Exhibit 3 - Docket in 3:16cv558, # 6 Exhibit 4 - Docket in 6:15cv618, # 7 Exhibit 5 - Juniper's Answer in 6:15cv618, # 8 Exhibit 6 - Juniper's Answer in 3:16cv558)(Garteiser, Randall) (Entered: 04/27/2017)
- 05/04/2017 10 REPLY to Response to Motion re 8 Opposed MOTION for Extension of Time to File Answer filed by Juniper Networks, Inc.. (Attachments: # 1 Exhibit 1)(Fisch, Alan) (Entered: 05/04/2017)
- 05/11/2017 11 SUR-REPLY to Reply to Response to Motion re 8 Opposed MOTION for Extension of Time to File Answer filed by Blue Spike, LLC . (Attachments: # 1 Text of Proposed Order, # 2 Exhibit 1 - Uniloc v. Google)(Garteiser, Randall) (Entered: 05/11/2017)
- 05/12/2017 12 AMENDED COMPLAINT for patent infringement against Juniper Networks, Inc., filed by Blue Spike, LLC. (Attachments: # 1 Ex. 1 - evidence related to infringement., # 2 Ex. 2 - evidence related to claim construction, # 3 Ex. 3 - evidence related to infringement., # 4 Ex. 4 - evidence related to infringement., # 5 Ex. 5 - evidence related to infringement., # 6 Ex. 6 - evidence related to infringement., # 7 Ex. 7 - evidence related to infringement., # 8 Ex. 8 - evidence related to infringement., # 9 Ex. 9 - evidence related to infringement., # 10 Ex. 10 - evidence related to infringement., # 11 Ex. 11 - evidence related to infringement., # 12 Ex. 12 - evidence related to infringement.)(Garteiser, Randall) (Entered: 05/12/2017)
- 05/22/2017 13 NOTICE of Attorney Appearance by Melissa Richards Smith on behalf of Juniper Networks, Inc. (Smith, Melissa) (Entered: 05/22/2017)
- 05/26/2017 14 CORPORATE DISCLOSURE STATEMENT filed by Juniper Networks, Inc. (Fisch, Alan) (Entered: 05/26/2017)
- 05/26/2017 15 *** WITHDRAWN PER 20 NOTICE*** MOTION to Dismiss for Improper Venue by Juniper Networks, Inc.. (Attachments: # 1 Text of Proposed Order)(Fisch, Alan) Modified on 7/11/2017 (mll,). (Entered: 05/26/2017)
- 06/05/2017 16 ORDER Setting Hearing on Motion 15 MOTION to Dismiss for Improper Venue : Motion Hearing set for 7/11/2017 09:00 AM in Ctrm 353 (Tyler) before Magistrate Judge K. Nicole Mitchell. Signed by Magistrate Judge K. Nicole Mitchell on 06/05/17. (mll,) (Entered: 06/05/2017)
- 06/09/2017 17 RESPONSE in Opposition re 15 MOTION to Dismiss for Improper Venue filed by Blue Spike, LLC . (Attachments: # 1 Text of Proposed Order) (Garteiser, Randall) (Entered: 06/09/2017)
- 06/10/2017 18 Additional Attachments to Main Document: 17 Response in Opposition to Motion.. (Attachments: # 1 Exhibit 1 - Screen Shot of Juniper Networks' Plano Texas Office, # 2 Exhibit 2 - Juniper Networks' Employees in this District, # 3 Exhibit 3 - Juniper Networks' website listing of training in Frisco, Texas, # 4 Exhibit 4 - Juniper Networks classes offered in this district, # 5 Exhibit 5 - Juniper Networks classes offered in this district, # 6 Exhibit 6 - Juniper Networks classes offered in this district, # 7 Exhibit 7 - Juniper Networks' Memorandum of Law in Support of Transfer to E.D. Texas, # 8 Exhibit 8 - Order transferring Juniper Networks to E.D. Texas at Juniper Networks' request, # 9 Exhibit 9 - Article in American Lawyer, # 10 Exhibit 10 - Juniper Networks' exhibit in support of its motion to

- transfer to E.D. Texas, # 11 Exhibit 11 - Order requiring supplemental briefing post-TC Heartland, # 12 Exhibit 12 - Order requiring supplemental briefing post-TC Heartland)(Garteiser, Randall) (Entered: 06/10/2017)
- 06/19/2017 19 *** WITHDRAWN PER 20 *** Opposed MOTION for Extension of Time to File Response/Reply as to 15 MOTION to Dismiss for Improper Venue by Juniper Networks, Inc.. (Attachments: # 1 Text of Proposed Order)(Fisch, Alan) Modified on 8/18/2017 (gsg). (Entered: 06/19/2017)
- 06/23/2017 20 NOTICE by Juniper Networks, Inc. re 15 MOTION to Dismiss for Improper Venue (Fisch, Alan) (Entered: 06/23/2017)
- 07/06/2017 21 NOTICE of Attorney Appearance - Pro Hac Vice by Jeffrey Saltman on behalf of Juniper Networks, Inc.. Filing fee \$ 100, receipt number 0540-6366481. (Saltman, Jeffrey) (Entered: 07/06/2017)
- 07/07/2017 NOTICE OF CANCELLATION OF HEARING. The Motion Hearing set for 7/11/17 at 9:00 a.m. is CANCELLED. Defendant Juniper filed notice of withdrawal of 15 Motion to Dismiss for Improper Venue. (leh,) (Entered: 07/07/2017)
- 07/10/2017 22 ORDER denying 8 Motion for Extension of Time to Answer. Defendant Juniper shall file its response by 7-20-2017 at 5:00 p.m or re-urge the Motion in light of the Amended Complaint. Signed by Magistrate Judge K. Nicole Mitchell on 07/10/17. (mll,) (Entered: 07/10/2017)
- 07/11/2017 23 NOTICE by Blue Spike, LLC of Motion requesting MDL transfer pursuant to § 1407 (Attachments: # 1 Motion requesting MDL transfer, # 2 Brief in support of Motion requesting MDL transfer)(Garteiser, Randall) (Entered: 07/11/2017)
- 07/20/2017 24 ANSWER to 12 Amended Complaint,, , COUNTERCLAIM against Blue Spike, LLC by Juniper Networks, Inc..(Fisch, Alan) (Entered: 07/20/2017)
- 07/26/2017 25 CORPORATE DISCLOSURE STATEMENT filed by Juniper Networks, Inc. (Fisch, Alan) (Entered: 07/26/2017)
- 07/27/2017 26 SCHEDULING ORDER. Scheduling Conference set for 9/6/2017 AT 10:30 AM in Tyler Courthouse before Magistrate Judge K. Nicole Mitchell. Signed by Magistrate Judge K. Nicole Mitchell on 7/27/2017. (gsg) (Entered: 07/27/2017)
- 08/10/2017 27 RESPONSE to 24 Answer to Amended Complaint, Counterclaim by Blue Spike, LLC. (Garteiser, Randall) (Entered: 08/10/2017)
- 08/11/2017 28 CORPORATE DISCLOSURE STATEMENT filed by Blue Spike, LLC (Garteiser, Randall) (Entered: 08/11/2017)
- 08/17/2017 29 NOTICE of Attorney Appearance - Pro Hac Vice by Kathleen Ryland on behalf of Juniper Networks, Inc.. Filing fee \$ 100, receipt number 0540-6428741. (Ryland, Kathleen) (Entered: 08/17/2017)
- 08/17/2017 30 CORPORATE DISCLOSURE STATEMENT filed by Juniper Networks, Inc. (Fisch, Alan) (Entered: 08/17/2017)
- 08/17/2017 31 NOTICE of Attorney Appearance - Pro Hac Vice by Roy William Sigler on behalf of Juniper Networks, Inc.. Filing fee \$ 100, receipt number 0540-6429140. (Sigler, Roy) (Entered: 08/17/2017)
- 08/24/2017 32 MOTION to Change Venue to the Northern District of California by Juniper Networks, Inc.. (Attachments: # 1 Affidavit Decl. of S. Coonan, # 2 Exhibit 1 - Patent Cover Sheet, # 3 Exhibit 2 - Wistaria Registration, # 4 Exhibit 3 - Cooperman Profile, # 5 Exhibit 4 - Patent Application, # 6 Exhibit 5 - Marvell Website, # 7 Exhibit 6 - Broadcom Website, # 8 Exhibit 7 - Neifield Contact Page, # 9 Exhibit 8 - Revocation of Power of Attorney, # 10 Exhibit 9 - Travel time to ND Cal, # 11 Exhibit 10 - Travel time to Tyler, # 12 Exhibit 11 - Travel time to Dallas, # 13 Exhibit 12 - Median Time to Trial, # 14 Exhibit 13 - Caseload Profile, # 15 Text of Proposed Order)(Fisch, Alan) (Entered: 08/24/2017)

- 08/25/2017 33 Joint MOTION Entry of Docket Control Order, E-Discovery Order, and Protective Order by Juniper Networks, Inc.. (Attachments: # 1 Exhibit 1 - Proposed Docket Control Order, # 2 Exhibit 2 - Proposed E-Discovery Order, # 3 Exhibit 3 - Proposed Protective Order, # 4 Text of Proposed Order)(Fisch, Alan) (Additional attachment(s) added on 8/28/2017: # 5 Exhibit 1 - DCO Corrected, # 6 Exhibit 2 - E-Discovery Corrected, # 7 Exhibit 3 - Protective Order Corrected) (mjc,). (Entered: 08/25/2017)
- 09/06/2017 41 Minute Entry for proceedings held before Magistrate Judge K. Nicole Mitchell: Scheduling Conference held on 9/6/2017. (Court Reporter L Hardwick.) (Attachments: # 1 Attorney Sign In Sheet) (leh,) (Entered: 09/07/2017)
- 09/07/2017 36 DOCKET CONTROL ORDER: Pretrial Conference set for 5/2/2019 09:00 AM in Ctrm 353 (Tyler) before Magistrate Judge K. Nicole Mitchell. Jury Selection and Trial set for 5/13/2019 09:00 AM in Ctrm 353 (Tyler) before Magistrate Judge K. Nicole Mitchell. Markman Hearing set for 5/31/2018 09:00 AM in Ctrm 353 (Tyler) before Magistrate Judge K. Nicole Mitchell. Dispositive Motionn Hearing set for 1/23/2019 09:00 AM in Ctrm 353 (Tyler) before Magistrate Judge K. Nicole Mitchell. Signed by Magistrate Judge K. Nicole Mitchell on 9/7/17. (mjc,) (Entered: 09/07/2017)
- 09/07/2017 37 E-DISCOVERY ORDER entered. Signed by Magistrate Judge K. Nicole Mitchell on 9/7/17. (mjc,) (Entered: 09/07/2017)
- 09/07/2017 38 PROTECTIVE ORDER entered, granting 33 Joint MOTION Entry of Docket Control Order, E-Discovery Order, and Protective Order filed by Juniper Networks, Inc. Signed by Magistrate Judge K. Nicole Mitchell on 9/7/17. (mjc,) (Entered: 09/07/2017)
- 09/07/2017 39 CONSENT to Proceed Before US Magistrate Judge by Blue Spike, LLC, Juniper Networks, Inc. Case reassigned to Magistrate Judge K. Nicole Mitchell. (Attachments: # 1 Blue Spike Consent)(mjc,) (Entered: 09/07/2017)
- 09/07/2017 40 ORDER assigning case to Judge Mitchell by consent of the parties for all further proceedings and entry of judgment. Signed by Judge Robert W. Schroeder, III on 9/7/17. (mjc,) (Entered: 09/07/2017)
- 09/08/2017 42 Joint MOTION for Entry of Discovery Order by Juniper Networks, Inc.. (Attachments: # 1 Text of Proposed Order)(Smith, Melissa) (Entered: 09/08/2017)
- 09/11/2017 43 DISCOVERY ORDER. Signed by Magistrate Judge K. Nicole Mitchell on 09/11/17. (mll,) (Entered: 09/11/2017)
- 09/11/2017 44 Unopposed MOTION for Extension of Time to File Response/Reply as to 32 MOTION to Change Venue to the Northern District of California by Blue Spike, LLC. (Attachments: # 1 Text of Proposed Order)(Garteiser, Randall) (Entered: 09/11/2017)
- 09/11/2017 45 RESPONSE in Opposition re 32 MOTION to Change Venue to the Northern District of California filed by Blue Spike, LLC . (Attachments: # 1 Declaration of Randall Garteiser, # 2 Declaration of Scott Moskowitz, # 3 Text of Proposed Order, # 4 Exhibit 1 - Email between parties, # 5 Exhibit 2 - Email between parties, # 6 Exhibit 3 - Email between parties, # 7 Exhibit 4 - Joint Stipulation of Dismissal, # 8 Exhibit 5 - Juniper Networks' Motion to Transfer to E.D. Texas, # 9 Exhibit 6 - List of potential Juniper Networks' witnesses in Texas, # 10 Exhibit 7 - Article re Juniper Networks asking to be transferred to Texas)(Garteiser, Randall) (Entered: 09/11/2017)
- 09/11/2017 46 SEALED ADDITIONAL ATTACHMENTS to Main Document: 45 Response in Opposition to Motion,.. (Attachments: # 1 Exhibit A - Email between Moskowitz and Juniper Networks, # 2 Exhibit B - Email between Moskowitz and Juniper Networks, # 3 Exhibit C - Email between Moskowitz and Juniper Networks, # 4 Exhibit D - Email between Moskowitz and Juniper

- Networks, # 5 Exhibit E - Declaration of Dr. Unger re Moskowitz's health) (Garteiser, Randall) (Entered: 09/11/2017)
- 09/12/2017 47 ORDER granting 44 Motion for Extension of Time. Signed by Magistrate Judge K. Nicole Mitchell on 9/12/2017. (rlf) (Entered: 09/12/2017)
- 09/18/2017 48 Joint MOTION to Amend/Correct 36 Scheduling Order,, by Juniper Networks, Inc.. (Attachments: # 1 Text of Proposed Order)(Saltman, Jeffrey) (Entered: 09/18/2017)
- 09/19/2017 49 AMENDED DOCKET CONTROL ORDER granting 48 Motion to Amend/Correct Docket Control Order. Signed by Magistrate Judge K. Nicole Mitchell on 9/19/17. (mjc,) (Entered: 09/19/2017)
- 09/20/2017 50 REPLY to Response to Motion re 32 MOTION to Change Venue to the Northern District of California filed by Juniper Networks, Inc.. (Attachments: # 1 Exhibit Exhibit 1, # 2 Exhibit Exhibit 2, # 3 Exhibit Exhibit 3)(Fisch, Alan) (Entered: 09/20/2017)
- 09/22/2017 51 Joint MOTION to Amend/Correct 49 Order on Motion to Amend/Correct by Juniper Networks, Inc.. (Attachments: # 1 Text of Proposed Order) (Saltman, Jeffrey) (Entered: 09/22/2017)
- 09/25/2017 52 AMENDED DOCKET CONTROL ORDER granting 51 Motion to Amend/Correct Docket Control Order. Signed by Magistrate Judge K. Nicole Mitchell on 9/25/17. (mjc,) (Entered: 09/25/2017)
- 09/26/2017 53 NOTICE of Designation of Mediator, Honorable David Folsom, filed by Juniper Networks, Inc.. (Saltman, Jeffrey) (Entered: 09/26/2017)
- 09/27/2017 54 ORDER REFERRING CASE to Mediator. Hon. David Folsom (Ret.) is appointed as mediator. The Court designates counsel for Plaintiff to be responsible for timely contacting Judge Folsom and coordinating a time for mediation. Signed by Magistrate Judge K. Nicole Mitchell on 9/27/17. (mjc,) (Entered: 09/27/2017)
- 09/27/2017 55 SUR-REPLY to Reply to Response to Motion re 32 MOTION to Change Venue to the Northern District of California filed by Blue Spike, LLC . (Garteiser, Randall) (Entered: 09/27/2017)
- 10/05/2017 56 MOTION for Hearing re 32 MOTION to Change Venue to the Northern District of California by Blue Spike, LLC. (Attachments: # 1 Text of Proposed Order)(Garteiser, Randall) (Entered: 10/05/2017)
- 10/05/2017 57 NOTICE by Juniper Networks, Inc. REGARDING MDL CONSOLIDATION (Attachments: # 1 Exhibit Exhibit 1)(Smith, Melissa) (Entered: 10/05/2017)
- 10/05/2017 58 MDL 2794 ORDER that the motion for centralization of the actions listed on Schedule A is denied. (MDL Litigation Panel). (mll,) (Entered: 10/05/2017)
- 10/11/2017 59 ORDER Setting Hearing on Motion 32 MOTION to Change Venue to the Northern District of California : Motion Hearing set for 11/8/2017 09:00 AM in Ctrm 353 (Tyler) before Magistrate Judge K. Nicole Mitchell. Signed by Magistrate Judge K. Nicole Mitchell on 10/11/17. (mll,) (Entered: 10/11/2017)
- 11/07/2017 60 NOTICE of Attorney Appearance - Pro Hac Vice by Desmond Jui on behalf of Juniper Networks, Inc.. Filing fee \$ 100, receipt number 0540-6542796. (Jui, Desmond) (Entered: 11/07/2017)
- 11/08/2017 61 Minute Entry for proceedings held before Magistrate Judge K. Nicole Mitchell: Motion Hearing held on 11/8/2017 re 32 MOTION to Change Venue to the Northern District of California filed by Juniper Networks, Inc. (Court Reporter Amanda Leigh.) (Attachments: # 1 Attorney Sign In Sheet) (leh,) (Entered: 11/08/2017)
- 11/09/2017 62

- Unopposed MOTION to Amend/Correct the Docket Control Order by Juniper Networks, Inc.. (Attachments: # 1 Text of Proposed Order) (Saltman, Jeffrey) (Entered: 11/09/2017)
- 11/13/2017 63 AMENDED DOCKET CONTROL ORDER granting 62 Motion to Amend/Correct the Docket Control Order. Signed by Magistrate Judge K. Nicole Mitchell on 11/13/17. (mjc,) (Entered: 11/13/2017)
- 11/21/2017 64 NOTICE by Blue Spike, LLC Updating the Court on the Dismissal of Toshiba (Attachments: # 1 Exhibit A - Dismissal of Toshiba)(Garteiser, Randall) (Entered: 11/21/2017)
- 11/21/2017 65 Additional Attachments to Main Document: 64 Notice (Other).. (Garteiser, Randall) (Entered: 11/21/2017)
- 12/01/2017 66 NOTICE by Juniper Networks, Inc. Notice of Compliance with Patent Rules 3-3 and 3-4 (Saltman, Jeffrey) (Entered: 12/01/2017)
- 12/04/2017 67 *** DOCUMENT FILED IN ERROR. PLEASE DISREGARD.*** MOTION for Discovery BLUE SPIKES MOTION FOR JUDICIAL NOTICE IN OPPOSITION TO JUNIPER NETWORKS MOTION TO CHANGE VENUE TO THE NORTHERN DISTRICT OF CALIFORNIA (DKT 32) by Blue Spike, LLC. (Attachments: # 1 Exhibit A - Order from Court in N.D. Cal. Granting stipulated dismissal of Toshiba. No other pending Blue Spike cases in N.D. Cal., # 2 Text of ORDER GRANTING BLUE SPIKES MOTION FOR JUDICIAL NOTICE IN OPPOSITION TO JUNIPER NETWORKS MOTION TO CHANGE VENUE TO THE NORTHERN DISTRICT OF CALIFORNIA (DKT 32))(Garteiser, Randall) Modified on 12/5/2017 (mjc,). (Entered: 12/04/2017)
- 12/05/2017 *** FILED IN ERROR. Document # 67 Motion for Discovery. PLEASE IGNORE. DOCUMENT TO BE REFILED AS NOTICE (OTHER).*** (mjc,) (Entered: 12/05/2017)
- 12/05/2017 68 NOTICE by Blue Spike, LLC re 45 Response in Opposition to Motion,, 65 Additional Attachments to Main Document BLUE SPIKES MOTION FOR JUDICIAL NOTICE IN OPPOSITION TO JUNIPER NETWORKS MOTION TO CHANGE VENUE TO THE NORTHERN DISTRICT OF CALIFORNIA (DKT 32) (Attachments: # 1 Exhibit A - Order from Court in N.D. Cal. Granting stipulated dismissal of Toshiba. No other pending Blue Spike cases in N.D. Cal.)(Garteiser, Randall) (Entered: 12/05/2017)
- 01/26/2018 69 NOTICE by Blue Spike, LLC PLAINTIFFS NOTICE OF COMPLIANCE WITH PATENT RULE (P. R.) 4-1 AND UPDATE ON GROWING DISCOVERY DISPUTES. (Garteiser, Randall) (Entered: 01/26/2018)
- 01/26/2018 70 *** FILED IN ERROR BY ATTORNEY. PLEASE DISREGARD.*** NOTICE by Blue Spike, LLC . NOTICE OF JUNIPER NETWORKS NON-COMPLIANCE WITH PATENT RULE (P. R.) 4-1 And COURTS ORDER (DKT 63). (Garteiser, Randall) Modified on 1/26/2018 (rlf). (Entered: 01/26/2018)
- 01/26/2018 71 NOTICE by Juniper Networks, Inc. of Compliance with Patent Rule 4-1 (Saltman, Jeffrey) (Entered: 01/26/2018)
- 01/29/2018 72 RESPONSE to 70 Notice (Other) , 69 Notice (Other) filed by Juniper Networks, Inc.. (Saltman, Jeffrey) (Entered: 01/29/2018)
- 02/09/2018 73 Unopposed MOTION to Amend/Correct 63 Order on Motion to Amend/Correct the Docket Control Order by Juniper Networks, Inc.. (Attachments: # 1 Text of Proposed Order Proposed Order)(Saltman, Jeffrey) (Entered: 02/09/2018)
- 02/12/2018 74 AMENDED DOCKET CONTROL ORDER entered, granting 73 Motion to Amend/Correct the Docket Control Order. Signed by Magistrate Judge K. Nicole Mitchell on 2/12/18. (mjc,) (Entered: 02/12/2018)
- 02/15/2018 75 NOTICE by Juniper Networks, Inc. of Compliance with Service of Privilege Log (Saltman, Jeffrey) (Entered: 02/15/2018)
- 02/15/2018 76 NOTICE by Blue Spike, LLC of Compliance with Service of Privilege Log (Garteiser, Randall) (Entered: 02/15/2018)

- 02/26/2018 77 NOTICE by Juniper Networks, Inc. of Compliance with Patent Rule 4-2 (Saltman, Jeffrey) (Entered: 02/26/2018)
- 03/01/2018 78 Unopposed MOTION to Withdraw as Attorney Desmond Jui by Juniper Networks, Inc.. (Attachments: # 1 Text of Proposed Order)(Sigler, Roy) (Entered: 03/01/2018)
- 03/02/2018 79 ORDER granting 78 Motion to Withdraw as Attorney. Attorney Desmond Jui terminated as counsel for Defendant. Signed by Magistrate Judge K. Nicole Mitchell on 3/2/2018. (mjc,) (Entered: 03/02/2018)
- 03/08/2018 80 ORDER granting 32 Motion to Change Venue to the Northern District of California. Signed by Magistrate Judge K. Nicole Mitchell on 3/8/2018. (mjc,) (Entered: 03/08/2018)
- 03/19/2018 81 MOTION for Reconsideration re 80 Order on Motion to Change Venue by Blue Spike, LLC. (Attachments: # 1 Text of Proposed Order Granting Blue Spike's Motion for Reconsideration and Denying Juniper Networks Motion to Transfer pursuant to § 1404(a))(Garteiser, Randall) (Entered: 03/19/2018)
- 03/19/2018 82 Joint MOTION to Stay by Juniper Networks, Inc.. (Attachments: # 1 Text of Proposed Order)(Smith, Melissa) (Entered: 03/19/2018)
- 03/21/2018 83 ORDER granting 82 Motion to Stay until the case is transferred or until a Court order granting Blue Spike's motion for reconsideration. Signed by Magistrate Judge K. Nicole Mitchell on 3/21/2018. (mjc,) (Entered: 03/21/2018)
- 04/03/2018 84 RESPONSE in Opposition re 81 MOTION for Reconsideration re 80 Order on Motion to Change Venue filed by Juniper Networks, Inc.. (Attachments: # 1 Exhibit 1, # 2 Exhibit 2, # 3 Exhibit 3, # 4 Exhibit 4, # 5 Exhibit 5, # 6 Exhibit 6, # 7 Exhibit 7, # 8 Text of Proposed Order)(Fisch, Alan) (Entered: 04/03/2018)
- 04/10/2018 85 REPLY to Response to Motion re 81 MOTION for Reconsideration re 80 Order on Motion to Change Venue filed by Blue Spike, LLC . (Garteiser, Randall) (Entered: 04/10/2018)
- 04/11/2018 86 REPLY to Response to Motion re 81 MOTION for Reconsideration re 80 Order on Motion to Change Venue [Correct Document to Replace Dkt 85 that was Filed In Error] filed by Blue Spike, LLC . (Garteiser, Randall) (Entered: 04/11/2018)
- 04/19/2018 87 SUR-REPLY to Reply to Response to Motion re 81 MOTION for Reconsideration re 80 Order on Motion to Change Venue filed by Juniper Networks, Inc.. (Attachments: # 1 Exhibit 1)(Fisch, Alan) (Entered: 04/19/2018)
- 05/08/2018 88 ORDER. On May 8, 2018, the parties contacted the Court regarding a discovery dispute. It is hereby ORDERED that a telephonic conference regarding this discovery dispute is set for Wednesday, May 9, 2018 at 2:00 p.m. before Judge K. Nicole Mitchell in Tyler, Texas. The parties shall file a brief, no longer than five pages, detailing the discovery dispute by 12:00 p.m., Wednesday, May 9, 2018. Juniper Networks, Inc. is further ORDERED to arrange the teleconference and inform the Court of the arrangements. Signed by Magistrate Judge K. Nicole Mitchell on 5/8/2018. (kls,) (Entered: 05/08/2018)
- 05/09/2018 89 BRIEF filed Requesting Enforcement of the Parties' Agreement to Continue Discovery by Juniper Networks, Inc.. (Attachments: # 1 Exhibit 1, # 2 Exhibit 2, # 3 Exhibit 3, # 4 Exhibit 4, # 5 Exhibit 5, # 6 Exhibit 6, # 7 Exhibit 7, # 8 Exhibit 8, # 9 Exhibit 9, # 10 Exhibit 10, # 11 Exhibit 11, # 12 Text of Proposed Order)(Sigler, Roy) (Entered: 05/09/2018)
- 05/09/2018 90 BRIEF filed Opposing Juniper Networks' Notice of Non-party Depositions against 83 This Court's Stay of Case Proceedings by Blue Spike, LLC. (Garteiser, Randall) (Entered: 05/09/2018)

- 05/09/2018 91 Minute Entry for proceedings held before Magistrate Judge K. Nicole Mitchell: Telephone Conference held on 5/9/2018. (Court Reporter L Hardwick.) (leh,) (Entered: 05/09/2018)
- 05/11/2018 92 Joint MOTION to Amend/Correct 43 Order on Motion for Miscellaneous Relief Discovery Order by Juniper Networks, Inc.. (Attachments: # 1 Text of Proposed Order Amended Discovery Order)(Saltman, Jeffrey) (Entered: 05/11/2018)
- 05/14/2018 93 AMENDED DISCOVERY ORDER granting 92 Motion to Amend/Correct the Discovery Order. Signed by Magistrate Judge K. Nicole Mitchell on 5/14/2018. (mjc,) (Entered: 05/14/2018)
- 05/16/2018 94 ORDER denying 81 Motion for Reconsideration re 81 MOTION for Reconsideration re 80 Order on Motion to Change Venue filed by Blue Spike, LLC. The Court GRANTS the Motion to Transfer Venue to the Northern District of California 32 . Signed by Magistrate Judge K. Nicole Mitchell on 5/16/2018. (mjc,) (Entered: 05/16/2018)

Copyright © 2018 LexisNexis CourtLink, Inc. All rights reserved.
*** THIS DATA IS FOR INFORMATIONAL PURPOSES ONLY ***

US District Court Civil Docket

U.S. District - Texas Eastern
(Tyler)

6:18cv174

Blue Spike, Llc v. Suddenlink Communications et al

This case was retrieved from the court on Thursday, May 24, 2018

Date Filed: **04/19/2018**
Assigned To: **District Judge Robert W. Schroeder, III**
Referred To: **Magistrate Judge K. Nicole Mitchell** Class Code: **OPEN**
Nature of suit: **Patent (830)** Closed:
Cause: **Patent Infringement** Statute: **35:271**
Lead Docket: **None** Jury Demand: **Plaintiff**
Other: **6:18cv00181** Demand Amount: **\$0**
Docket: **6:18cv00195** NOS Description: **Patent**
6:18cv00223
Jurisdiction: **Federal Question**

Litigants

Attorneys

Blue Spike, Llc
Plaintiff

Randall T Garteiser
ATTORNEY TO BE NOTICED
Garteiser Honea PLLC
119 W. Ferguson St.
Tyler , TX 75702
USA
903-705-7420
Fax: 888-908-4400
Email: Rgarteiser@ghiplaw.Com

Suddenlink Communications
Defendant

Mark Nolan Reiter
LEAD ATTORNEY; ATTORNEY TO BE NOTICED
Gibson Dunn & Crutcher
2100 Mckinney Ave Suite 1100
Dallas , TX 75201
USA
214-698-3100
Fax: 214 571 2907
Email: Mreiter@gibsondunn.Com

Cequel Communications, Llc
Defendant

Mark Nolan Reiter
LEAD ATTORNEY; ATTORNEY TO BE NOTICED
Gibson Dunn & Crutcher
2100 Mckinney Ave Suite 1100

Dallas , TX 75201
 USA
 214-698-3100
 Fax: 214 571 2907
 Email: Mreiter@gibsondunn.Com

Altice USA, Inc.
 Defendant

Mark Nolan Reiter
 LEAD ATTORNEY; ATTORNEY TO BE NOTICED
 Gibson Dunn & Crutcher
 2100 Mckinney Ave Suite 1100
 Dallas , TX 75201
 USA
 214-698-3100
 Fax: 214 571 2907
 Email: Mreiter@gibsondunn.Com

Date	#	Proceeding Text	Source
04/20/2018	1	COMPLAINT against All Defendants (Filing fee \$ 400 receipt number 0540-6747391.), filed by Blue Spike, LLC. (Attachments: # 1 Civil Cover Sheet)(Garteiser, Randall) (Entered: 04/20/2018)	
04/20/2018	2	Notice of Filing of Patent/Trademark Form (AO 120). AO 120 mailed to the Director of the U.S. Patent and Trademark Office. (Garteiser, Randall) (Entered: 04/20/2018)	
04/20/2018		DEMAND for Trial by Jury by Blue Spike, LLC. (mjc,) (Entered: 04/20/2018)	
04/23/2018		District Judge Robert W. Schroeder, III and Magistrate Judge K. Nicole Mitchell added. (mll,) (Entered: 04/23/2018)	
04/23/2018	3	CASE REFERRED to Magistrate Judge K Nicole Mitchell. (mll,) (Entered: 04/23/2018)	
04/23/2018		In accordance with the provisions of 28 USC Section 636(c), you are hereby notified that a U.S. Magistrate Judge of this district court is available to conduct any or all proceedings in this case including a jury or non-jury trial and to order the entry of a final judgment. The form Consent to Proceed Before Magistrate Judge is available on our website. All signed consent forms, excluding pro se parties, should be filed electronically using the event Notice Regarding Consent to Proceed Before Magistrate Judge. (mll,) (Entered: 04/23/2018)	
04/23/2018	4	Additional Attachments to Main Document: 1 Complaint.. (Attachments: # 1 Exhibit 02 - US Patent 7287275, # 2 Exhibit 03 - US Patent 7475246, # 3 Exhibit 04 - US Patent 8224705, # 4 Exhibit 05 - US Patent 8473746, # 5 Exhibit 06 - US Patent 8538011, # 6 Exhibit 07 - US Patent 8739295, # 7 Exhibit 08 - US Patent 9021602, # 8 Exhibit 09 - US Patent 9104842, # 9 Exhibit 10 - US Patent 9934408, # 10 Exhibit 11 - US Patent RE44222, # 11 Exhibit 12 - US Patent RE44307, # 12 Exhibit 13 - US Patent 7159116, # 13 Exhibit A - Suddenlink deals)(Garteiser, Randall) (Entered: 04/23/2018)	
04/24/2018	5	SUMMONS Issued as to Altice USA, Inc., Cequel Communications, LLC, Suddenlink Communications. (Attachments: # 1 Summons(es) - Cequel Communications)(mjc,) (Entered: 04/24/2018)	
04/25/2018	6	SUMMONS Returned Executed by Blue Spike, LLC. Cequel Communications, LLC served on 4/25/2018, answer due 5/16/2018; Suddenlink Communications served on 4/25/2018, answer due 5/16/2018. (Garteiser, Randall) (Entered: 04/25/2018)	
04/25/2018	7		

- SUMMONS Returned Executed by Blue Spike, LLC. Altice USA, Inc. served on 4/25/2018, answer due 5/16/2018. (Garteiser, Randall) (Entered: 04/25/2018)
- 05/03/2018 8 Defendant's Unopposed First Application for Extension of Time to Answer Complaint re Altice USA, Inc., Cequel Communications, LLC, Suddenlink Communications.(Reiter, Mark) (Entered: 05/03/2018)
- 05/04/2018 Defendant's Unopposed First Application for Extension of Time to Answer Complaint is granted pursuant to Local Rule CV-12 for Cequel Communications, LLC to 6/15/2018; Suddenlink Communications to 6/15/2018. 30 Days Granted for Deadline Extension.(mjc,) (Entered: 05/04/2018)
- 05/04/2018 Defendant's Unopposed First Application for Extension of Time to Answer Complaint is granted pursuant to Local Rule CV-12 for Altice USA, Inc. to 6/15/2018. 30 Days Granted for Deadline Extension.(mjc,) (Entered: 05/04/2018)

Copyright © 2018 LexisNexis CourtLink, Inc. All rights reserved.
*** THIS DATA IS FOR INFORMATIONAL PURPOSES ONLY ***

US District Court Civil Docket

U.S. District - Texas Eastern
(Tyler)

6:18cv181

Blue Spike, Llc v. Comcast Corporation D/ B/ A Xfinity et al

This case was retrieved from the court on Thursday, May 24, 2018

Date Filed: 04/27/2018	
Assigned To: District Judge Robert W. Schroeder, III	
Referred To: Magistrate Judge K. Nicole Mitchell	Class Code: OPEN
Nature of suit: Patent (830)	Closed:
Cause: Patent Infringement	Statute: 35:271
Lead Docket: None	Jury Demand: Plaintiff
Other: 6:16cv01384	Demand Amount: \$0
Docket: 6:17cv00016	NOS Description: Patent
6:18cv00174	
6:18cv00195	
6:18cv00223	
Jurisdiction: Federal Question	

Litigants

Attorneys

Blue Spike, Llc
Plaintiff

Randall T Garteiser
ATTORNEY TO BE NOTICED
Garteiser Honea PLLC
119 W. Ferguson St.
Tyler, TX 75702
USA
903-705-7420
Fax: 888-908-4400
Email: Rgarteiser@ghiplaw.Com

Comcast Corporation D/B/A Xfinity
Defendant

Comcast Cable Communications, Llc D/B/A Xfinity
Defendant

Comcast Business Communications, Llc
Defendant

Comcast Enterprise Services, Llc
Defendant

Comcast Cable Communications Management, Llc
Defendant

Comcast of Houston, Llc D/B/A Xfinity
Defendant

Comcast Holdings Corporation
Defendant

Comcast Shared Services, Llc
Defendant

Date	#	Proceeding Text	Source
04/27/2018	1	COMPLAINT for Patent Infringement against All Defendants (Filing fee \$ 400 receipt number 0540-6757568.), filed by Blue Spike, LLC. (Attachments: # 1 Civil Cover Sheet, # 2 Exhibit 01 - Comcast Business Managed Solutions, # 3 Exhibit 02 - US Patent 7287275, # 4 Exhibit 03 - US Patent 7475246, # 5 Exhibit 04 - US Patent 8224705, # 6 Exhibit 05 - US Patent 8473746, # 7 Exhibit 06 - US Patent 8538011, # 8 Exhibit 07 - US Patent 8739295, # 9 Exhibit 08 - US Patent 9021602, # 10 Exhibit 09 - US Patent 9104842, # 11 Exhibit 10 - US Patent 9934408, # 12 Exhibit 11 - US Patent RE44222, # 13 Exhibit 12 - US Patent RE44307, # 14 Exhibit 13 - US Patent 7159116, # 15 Exhibit A - XFINITY services and offers, # 16 Exhibit B - Comcast stores in Liberty, Mont Belvieu and Dayton, TX in Liberty County and Waskom in Harrison County which are all in the EDTX venue, # 17 Exhibit C - Comcast Services in Liberty, TX, # 18 Exhibit D - Comcast in Waskom, TX, # 19 Exhibit E - Comcast Services in Dayton, TX, # 20 Exhibit F - Texas Cable Association on franchise fees) (Garteiser, Randall) (Entered: 04/27/2018)	
04/27/2018		DEMAND for Trial by Jury by Blue Spike, LLC. (mll,) (Entered: 04/30/2018)	
04/30/2018		District Judge Robert W. Schroeder, III and Magistrate Judge K. Nicole Mitchell added. (mll,) (Entered: 04/30/2018)	
04/30/2018	2	CASE REFERRED to Magistrate Judge K Nicole Mitchell. (mll,) (Entered: 04/30/2018)	
04/30/2018		In accordance with the provisions of 28 USC Section 636(c), you are hereby notified that a U.S. Magistrate Judge of this district court is available to conduct any or all proceedings in this case including a jury or non-jury trial and to order the entry of a final judgment. The form Consent to Proceed Before Magistrate Judge is available on our website. All signed consent forms, excluding pro se parties, should be filed electronically using the event Notice Regarding Consent to Proceed Before Magistrate Judge. (mll,) (Entered: 04/30/2018)	

Copyright © 2018 LexisNexis CourtLink, Inc. All rights reserved.
*** THIS DATA IS FOR INFORMATIONAL PURPOSES ONLY ***

US District Court Civil Docket

U.S. District - Texas Eastern
(Tyler)

6:18cv195

Blue Spike, Llc v. Charter Communications, Inc.

This case was retrieved from the court on Thursday, May 24, 2018

Date Filed: **05/04/2018**
Assigned To: **District Judge Robert W. Schroeder, III**
Referred To: **Magistrate Judge K. Nicole Mitchell**
Nature of suit: **Patent (830)**
Cause: **Patent Infringement**
Lead Docket: **None**
Other: **6:17cv00016**
Docket: **6:18cv00174**
6:18cv00181
6:18cv00223
Jurisdiction: **Federal Question**

Class Code: **OPEN**
Closed:
Statute: **35:271**
Jury Demand: **Plaintiff**
Demand Amount: **\$0**
NOS Description: **Patent**

Litigants

Attorneys

Blue Spike, Llc
Plaintiff

Randall T Garteiser
ATTORNEY TO BE NOTICED
Garteiser Honea PLLC
119 W. Ferguson St.
Tyler , TX 75702
USA
903-705-7420
Fax: 888-908-4400
Email: Rgarteiser@ghiplaw.Com

Charter Communications, Inc.
Defendant

Date	#	Proceeding Text	Source
05/04/2018	1	COMPLAINT against Charter Communications, Inc. (Filing fee \$ 400 receipt number 0540-6765886.), filed by Blue Spike, LLC. (Attachments: # 1 Civil Cover Sheet, # 2 Exhibit 01 - Managed Network Services, # 3 Exhibit 02 - US Patent 7287275, # 4 Exhibit 03 - US Patent 7475246, # 5 Exhibit 04 - US Patent 8224705, # 6 Exhibit 05 - US Patent 8473746, # 7 Exhibit 06 - US Patent 8538011, # 8 Exhibit 07 - US Patent 8739295, # 9 Exhibit 08 - US Patent 9021602, # 10 Exhibit 09 - US Patent 9104842, # 11 Exhibit 10 - US Patent 9934408, # 12 Exhibit 11 - US Patent RE44222,	

- # 13 Exhibit 12 - US Patent RE44307, # 14 Exhibit 13 - US Patent 7159116, # 15 Exhibit A - Charter Spectrum Packages Bundle, # 16 Exhibit B - Charter store locations in Plano, # 17 Exhibit C - Spectrum serving Marshall TX, # 18 Exhibit D - Spectrum serving Plano TX in Collin County, # 19 Exhibit E - Texas Cable Assoc on Franchise Fees 101) (Garteiser, Randall) (Entered: 05/04/2018)
- 05/04/2018 2 Notice of Filing of Patent/Trademark Form (AO 120). AO 120 mailed to the Director of the U.S. Patent and Trademark Office. (Garteiser, Randall) (Entered: 05/04/2018)
- 05/09/2018 DEMAND for Trial by Jury by Blue Spike, LLC. (mll,) (Entered: 05/09/2018)
- 05/09/2018 District Judge Robert W. Schroeder, III and Magistrate Judge K. Nicole Mitchell added. (mll,) (Entered: 05/09/2018)
- 05/09/2018 3 CASE REFERRED to Magistrate Judge K Nicole Mitchell. (mll,) (Entered: 05/09/2018)
- 05/09/2018 In accordance with the provisions of 28 USC Section 636(c), you are hereby notified that a U.S. Magistrate Judge of this district court is available to conduct any or all proceedings in this case including a jury or non-jury trial and to order the entry of a final judgment. The form Consent to Proceed Before Magistrate Judge is available on our website. All signed consent forms, excluding pro se parties, should be filed electronically using the event Notice Regarding Consent to Proceed Before Magistrate Judge. (mll,) (Entered: 05/09/2018)

Copyright © 2018 LexisNexis CourtLink, Inc. All rights reserved.
*** THIS DATA IS FOR INFORMATIONAL PURPOSES ONLY ***

US District Court Civil Docket

U.S. District - Texas Eastern
(Tyler)

6:18cv223

Blue Spike, Llc v. Altice USA, Inc.

This case was retrieved from the court on Thursday, May 24, 2018

Date Filed: **05/18/2018**
Assigned To: **District Judge Robert W. Schroeder, III**
Referred To: **Magistrate Judge K. Nicole Mitchell**
Nature of suit: **Patent (830)**
Cause: **Patent Infringement**
Lead Docket: **None**
Other: **6:16cv01384**
Docket: **6:17cv00016**
6:18cv00174
6:18cv00181
6:18cv00195
Jurisdiction: **Federal Question**

Class Code: **OPEN**
Closed:
Statute: **35:271**
Jury Demand: **Plaintiff**
Demand Amount: **\$0**
NOS Description: **Patent**

Litigants

Attorneys

Blue Spike, Llc
Plaintiff

Randall T Garteiser
ATTORNEY TO BE NOTICED
Garteiser Honea PLLC
119 W. Ferguson St.
Tyler, TX 75702
USA
903-705-7420
Fax: 888-908-4400
Email: Rgarteiser@ghiplaw.Com

Altice USA, Inc.
Defendant

Date	#	Proceeding Text	Source
05/19/2018	1	COMPLAINT against Altice USA, Inc. (Filing fee \$ 400 receipt number 0540-6784542.), filed by Blue Spike, LLC. (Attachments: # 1 Civil Cover Sheet, # 2 Exhibit 01 - Altice Lightpatch Managed WiFi I Enterprise WiFi, # 3 Exhibit 02 - US Patent 7287275, # 4 Exhibit 03 - US Patent 7475246, # 5 Exhibit 04 - US Patent 8224705, # 6 Exhibit 05 - US Patent 8473746, # 7 Exhibit 06 - US Patent 8538011, # 8 Exhibit 07 - US Patent 8739295, # 9 Exhibit 08 - US Patent 9021602, # 10 Exhibit 09 - US Patent	

- 9104842, # 11 Exhibit 10 - US Patent 9934408, # 12 Exhibit 11 - US Patent RE44222, # 13 Exhibit 12 - US Patent RE44307, # 14 Exhibit 13 - US Patent 7159116, # 15 Exhibit A - Optimum Customer Service) (Garteiser, Randall) (Entered: 05/19/2018)
- 05/19/2018 2 Notice of Filing of Patent/Trademark Form (AO 120). AO 120 mailed to the Director of the U.S. Patent and Trademark Office. (Garteiser, Randall) (Entered: 05/19/2018)
- 05/19/2018 District Judge Robert W. Schroeder, III and Magistrate Judge K. Nicole Mitchell added. (rlf) (Entered: 05/21/2018)
- 05/19/2018 3 CASE REFERRED to Magistrate Judge K. Nicole Mitchell. (rlf) (Entered: 05/21/2018)
- 05/21/2018 In accordance with the provisions of 28 USC Section 636(c), you are hereby notified that a U.S. Magistrate Judge of this district court is available to conduct any or all proceedings in this case including a jury or non-jury trial and to order the entry of a final judgment. The form Consent to Proceed Before Magistrate Judge is available on our website. All signed consent forms, excluding pro se parties, should be filed electronically using the event Notice Regarding Consent to Proceed Before Magistrate Judge. (rlf) (Entered: 05/21/2018)
- 05/21/2018 DEMAND for Trial by Jury by Blue Spike, LLC. (rlf) (Entered: 05/21/2018)

Copyright © 2018 LexisNexis CourtLink, Inc. All rights reserved.
*** THIS DATA IS FOR INFORMATIONAL PURPOSES ONLY ***

Objects

9104842 or 9,104,842

All authorities

5 families found out of 120,336,226 records searched

Publications Families

Publication number	Publication date	Title
US20160004875A1	01/07/2016	Data protection method and device
US8526611B2	09/03/2013	Utilizing data reduction in steganographic and cryptographic systems
US5089200A	02/18/1992	Process for melt extrusion of polymers
DE4110960A1	10/08/1992	Cleaning socket for tubes or containers - by device ensuring correct alignment of cover surface
GB9104842D0	04/17/1991	A HAZARD CLEARANCE INDICATOR

1

Privacy Policy (<https://www.lexisnexis.com/en-us/terms/privacy-policy.page>) |

Cookie Policy (https://www.lexisnexis.com/global/privacy/privacy-cookies/?gid=1658&locale=en_gb)

Lexis Advance®
Research

More

Select Category

Secondary...

0

Results for: 9104842 or 9,104,842



Actions

FILTERS

Secondary Materials (0)

No documents found in **Secondary Materials** .

Try the following:

Review your results in the other categories.

Modify your search (edit or remove some search terms or add synonyms).

Check for spelling errors.

Use the options available from the **Actions** menu above to modify your search.

Note: Some content may not be visible based on the restrictions of your subscription.



About
LexisNexis®

Privacy
Policy

Terms &
Conditions

Sign
Out

Copyright
© 2018
LexisNexis.
All rights
reserved.



Lexis Advance®
Research

Results for: 9104842 or 9,104,842

News (6)

Sort by: Relevance

1. US Patent Issued on Aug. 11 for "Data protection method and device" (Florida Inventor)

News | US Fed News | Aug 12, 2015 | 153

... further information, including images, charts and tables, please visit:

<http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO2&Sect2=HITOFF&p=1&u=%2Fmetahtml%2FPTO%2Fsearch-bool.html&r=1&f=G&i=50&co1=AND&d=PTXT&s1=9104842&OS=9104842&RS=9104842>

For any query with respect to this article or ...

... Aug. 12 -- United States Patent no. **9,104,842** , issued on Aug. 11. "Data protection method and device" was ...

2. Moskowitz Scott a Awarded Patent for Data Protection Method and Device

News | Global IP News. Information Technology Patent News | Aug 11, 2015 | 284

... Technology Patent News Patent Application Number: 11/895,388 Patent Publication

Number: **9,104,842** International Patent Classification Codes: G06F 21/10 (20130101), G06F 21/125 (20130101), ...

3. Moskowitz Scott a Awarded Patent for Data Protection Method and Device

News | Global IP News. Information Technology Patent News | Aug 11, 2015 | 286

... Application Number: 11/895,388 Patent Publication Number: **9,104,842** International Patent Classification Codes: G06F 21/10 (20130101), G06F ...

4. US Patent granted to Moskowitz (Florida) on August 11 titled as "Data protection method and device"

News | Plus Patent News | Aug 11, 2015 | 124

... States Patent and Trademark Office has granted patent no. **9,104,842** , on August 11, 2014, to Moskowitz (Florida), titled ...

5. Familie&co familie.de

News | GlobalAdSource (German) | Mar 10, 2010 | 29

... Media Type Print Country Germany Source familie&Co Product Familie&co familie.de ...

- 6.

Results for trading or shares in sector Standard MICEX-RTS at 12:00 MSK

News | Russia & CIS Business and Financial Newswire | Jan 26, 2012 | 181

... 28.25 28.257 28.26 44292097 1564700 Surgutneftegaz, pr. 18.644 18.65 18.642 **9104842**
486800 Tatneft 175.24 175.44 177.41 948177 5300 TGK-1 0.009 0.01 ...

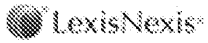
Content type: News

Terms: 9104842 or 9,104,842

Search Type: Boolean - Fewer Results

Narrow By: Sources: News

Date and Time: May 24, 2018 09:42:56 a.m. EDT



[About LexisNexis](#)

[Privacy Policy](#)

[Terms & Conditions](#)

[Sign Out](#)

Copyright
© 2018
LexisNexis.
All rights reserved.





UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
90/014,138	05/16/2018	9104842		7638

31518 7590 06/19/2018
NEIFELD IP LAW, PC
5400 Shawnee Road
Suite 310
ALEXANDRIA, VA 22312-2300

EXAMINER

BONSHOCK, DENNIS G

ART UNIT	PAPER NUMBER
----------	--------------

3992

MAIL DATE	DELIVERY MODE
-----------	---------------

06/19/2018

PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.



UNITED STATES PATENT AND TRADEMARK OFFICE

Commissioner for Patents
United States Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450
www.uspto.gov

DO NOT USE IN PALM PRINTER

(THIRD PARTY REQUESTER'S CORRESPONDENCE ADDRESS)

Joseph F. Edell
Fisch Sigler LLP
5301 Wisconsin Ave, NW
Fourth Floor
Washington, DC 20015

***EX PARTE* REEXAMINATION COMMUNICATION TRANSMITTAL FORM**

REEXAMINATION CONTROL NO. 90/014,138.

PATENT NO. 9,104,842.

ART UNIT 3992.

Enclosed is a copy of the latest communication from the United States Patent and Trademark Office in the above identified *ex parte* reexamination proceeding (37 CFR 1.550(f)).

Where this copy is supplied after the reply by requester, 37 CFR 1.535, or the time for filing a reply has passed, no submission on behalf of the *ex parte* reexamination requester will be acknowledged or considered (37 CFR 1.550(g)).

DECISION ON REQUEST FOR REEXAMINATION

A substantial new question of patentability affecting claims 11-14, of United States Patent Number: 9,104,842 issued to Moskowitz, hereinafter the '842 patent, is raised by the request for *ex parte* reexamination.

The present application is being examined under the pre-AIA first to invent provisions.

References

A total of 4 reference have been asserted in the Request as providing teachings relevant to the claims of the '842 Patent. The proposed references are as follows:

- (1) U.S. Patent No. 5,933,497 issued to Beetcher (hereinafter Beetcher / Ex.3)
- (2) Japanese Patent Application Publication No. H05334072 issued to Beetcher (hereinafter Beetcher '072 / Ex.4)
- (3) PCT Application Publication No. WO 97/26732 issued to Cooperman (hereinafter Cooperman / Ex. 6)
- (4) U.S. Patent No. 5,935,243 issued to Hasebe (hereinafter Hasebe / Ex. 7)

Prosecution History

U.S. Patent Application Serial No. 11/895,388, which resulted in issued Patent 9,104,842 (hereinafter the '842 patent), was filed on August 24, 2007.

During prosecution of the '842 Patent, claims 1-64 were rejected under 35 U.S.C. § 103(a) as being unpatentable over Holmes et al. (US Patent Number: 5,287,407) in further view of Houser et al. (US Patent Number: 5,606,609).

Claims 32-45 and 52-64 were remaining in the case and rejected over Holmes and Houser (supra), when the claims went to appeal.

The Examiners Answer, dated 8/8/2012, subsequently withdrew the rejections of claims 34, 45, 54, and 58, with claims 34, 45, and 54 left as objected to as being dependent upon a rejected base claim, and claim 58 (now patent claim 11) noted as 'recites allowable subject matter', with no further elaboration.

In the Decision on Appeal, dated 3/12/2015, the board considered the rejections under 35 U.S.C. § 103(a) (supra) and rendered the judgment that:

DECISION

The rejection of claims 32, 33, 35, 37, 39, 52, 53, 55-57, 59, and 63-64 under 35 U.S.C. § 103(a) is affirmed.

The rejection of claims 36, 38, 40-44, and 60-62 under 35 U.S.C. § 103(a) is reversed.

The Board in the Decision specifically noted, with respect to the reversed claims, that:

Claims 36 and 60 (patent claim 12)

Claims 36 and 60 include limitations that **require the underlying software functionality be enabled upon the presence or detection of a key, or other software code**. See, e.g., Claim 60 (“software code will provide said specified underlying functionality only after receipt of said first license key”). Appellant argues neither Holmes nor Houser teaches or suggests enabling software functionality based on a license key. App. Br. 80, 98. We agree. Holmes states the data block containing the identification information “does not play any part in the function of the software of the master file itself.” Holmes, col. 3, ll. 41-42. Accordingly, we cannot sustain the rejection of claims 36 and 60.

Claim 61 (patent claim 13)

Appellant contends the Examiner’s rejection of claim 61 does not address various limitations of the claim, such as “**encoding said first code resource to form an encoded first code resource,**” or an “**encoded first code resource, and a decode resource for decoding said encoded first code resource**” in the software. App. Br. 99-100. The Examiner relies on reasoning found in the rejections of claims 32 and 43. Final Act. 7. The Examiner’s findings do not support the combination of Houser and Holmes teaches or suggests a modified software code comprising an encoded first code resource and a decode resource for decoding the encoded first code resource, wherein the decode resource is configured to decode

the encoded first code resource upon receipt of a first license key. Accordingly, we do not sustain the rejection of claim 61.

Claim 62 (patent claim 14)

Appellant argues “neither Holmes nor Houser disclose or suggest encoding code interrelationships between code resources of the software.” App. Br. 103-04. The Examiner bases the rejection of claim 62 on the reasons set forth in rejecting claims 32 and 61. Final Act. 8. We disagree the same reasons apply. For example, claims 32 and 61 do not recite limitations regarding **“software code interrelationships between code resources that result in a specified underlying functionality.”** Because the Examiner has not shown how the references teach or suggest all the limitations of claim 62, we do not sustain its rejection.

On 6/4/2015, the Examiner issued a Notice of Allowance based on the Board’s March 12, 2015 decision. After the notice of allowance. Patent Owner requested claim amendments, adding, in pertinent part, the term “product” to claim 58. The patent issued on August 11, 2015.

Substantial New Question of Patentability

The Requester suggests that the following references and/or combinations of references provide elements which are allegedly equivalent to claims 11-14 of the ‘842 patent.

Claim 11 is presented below with italicized sections showing the limitations that are believed to be the allowable limitations, and which are used by the Examiner to show how specific teachings of the proposed references raise a substantial new question of patentability.

Claim 11:

11. A method for licensed software use, the method comprising:

loading a software product on a computer, said computer comprising a processor, memory, an input, and an output, so that said computer is programmed to execute said software product;

said software product outputting a prompt for input of license information; and

said software product using license information entered via said input in response to said prompt in a routine designed to decode a first license code encoded in said software product.

Claim 12 is presented below with italicized sections showing the limitations that are believed to be the allowable limitations, and which are used by the Examiner to show how specific teachings of the proposed references raise a substantial new question of patentability.

Claim 12:

12. A method for encoding software code using a computer having a processor and memory, comprising: storing a software code in said memory; wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system; and encoding, by said computer using at least a first license key and an

encoding algorithm, said software code, to form a first license key encoded software code; and wherein, when installed on a computer system, said first license key encoded software code will ***provide said specified underlying functionality only after receipt of said first license key.***

Claim 13 is presented below with italicized sections showing the limitations that are believed to be the allowable limitations, and which are used by the Examiner to show how specific teachings of the proposed references raise a substantial new question of patentability.

Claim 13:

13. A method for encoding software code using a computer having a processor and memory, comprising:
storing a software code in said memory;
wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system; and
modifying, by said computer, using a first license key and an encoding algorithm, said software code, to form a modified software code; and
wherein said modifying comprises ***encoding said first code resource to form an encoded first code resource;***
wherein said modified software code comprises said encoded first code resource, and a decode resource for decoding said encoded first code resource;
wherein said ***decode resource is configured to decode said encoded first code resource upon receipt of said first license key.***

Claim 14 is presented below with italicized sections showing the limitations that are believed to be the allowable limitations, and which are used by the Examiner to show how specific teachings of the proposed references raise a substantial new question of patentability.

Claim 14:

14. A method for encoding software code using a computer having a processor and memory, comprising:
storing a software code in said memory;
wherein said software code defines **software code interrelationships between code resources that result in a specified underlying functionality** when installed on a computer system; and
encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key *encoded software code in which at least one of said software code interrelationships are encoded.*

Beetcher

The Requestor alleges that Beetcher raises a substantial new question of patentability with respect to claims 11-14 of the '842 patent. For purposes of Reexamination the reading of Beetcher on the claims is provided on pages 19-50 of the Request.

Beetcher is new art that provides new, non-cumulative technological teachings that were not previously considered and discussed on the record during prosecution or reexamination of the '842 patent.

Beetcher teaches, with respect to claim 11, **using license information entered via said input in response to said prompt in a routine designed to decode a first license code encoded in said software product**, entering license information in response to a user interface supporting input of the entitlement key (see column 7, line 66 through column 8, line 8) resulting in an unlocking routine decoding the license code

in the software product (see column 7, line 39-42 and column 9, lines 49-60 and figure 9).

Beetcher teaches, with respect to claim 12, to ***provide said specified underlying functionality only after receipt of said first license key*** in teaching software distributed without entitlement to run, where an encrypted entitlement key is separately transmitted enabling execution of the software. The entitlement key further includes the serial number of the computer the software is licensed to as well as a plurality of entitlement bits indicating which software modules are enabled to run on the particular enabled computer (see column 6, lines 10-65, column 11, lines 4-39).

It is further noted that "during original prosecution, Patent Owner specified that "[e]ncoding using a key and an algorithm is known" " (see Ex. 2, Prosecution History at 519).

Beetcher further teaches, in relation to claim 13, ***encoding said first code resource to form an encoded first code resource*** in the teaching of placing the entitlement verification triggers in the in the object code (see column 4, lines 25-33); and ***decode resource is configured to decode said encoded first code resource upon receipt of said first license key*** in the teaching of utilizing the encrypted entitlement key to unlock program resources via their respective entitlement triggers (see column 10, lines 20-39).

It is further noted that "during original prosecution, Patent Owner specified that "[e]ncoding using a key and an algorithm is known" " (see Ex. 2, Prosecution History at 519).

Beetcher further teaches, in relation to claim 14's software code interrelationships between code resources that result in a specified underlying functionality, inputting a template formed from source code into the translator 127 generating entitlement verification triggers and inserting them into code and 'resolving references' after triggers are inserted, thereby showing code interrelationships that result in implementation of underlying functionality (see column 9, lines 1-20).

It is further noted that "during original prosecution, Patent Owner specified that "interrelationships between code resource are not that which is novel" " (see Ex. 2, Prosecution History at 519).

It is agreed that Beetcher as proposed in the request, raises a SNQ with respect to claims 11-14 of the '842 patent. There is a substantial likelihood that a reasonable Examiner would consider this teaching important in deciding whether or not these claims are patentable.

Accordingly, the Beetcher reference raises a substantial new question to claims 11-14, which question has not been decided in a previous examination of the '842 patent nor was there a final holding of invalidity by the Federal Courts regarding the '842 patent.

Beetcher '072

The Requestor alleges that Beetcher '072 raises a substantial new question of patentability with respect to claims 11-14 of the '842 patent. For purposes of Reexamination the reading of Beetcher '072 on the claims is provided on pages 50-81 of the Request.

Beetcher '072 is new art that provides new, non-cumulative technological teachings that were not previously considered and discussed on the record during prosecution or reexamination of the '842 patent.

Beetcher '072 teaches, with respect to claim 11, **using license information entered via said input in response to said prompt in a routine designed to decode a first license code encoded in said software product**, the user being prompted to enter a key to enable software functionality, where the computer uses a decode key to decode the license code (see paragraphs 24, 32, and 33 and figures 4 and 9). Here a 'qualification key' 111 is input via the console, lock release routine 430 makes the machine key, and subsequently the lock release routine 430 decodes the qualification grant key 11 using a machine key (see paragraph 40).

Beetcher '072 teaches, with respect to claim 12, to **provide said specified underlying functionality only after receipt of said first license key** in teaching using

the decoded grant key to enable entries in locking table for execution (see paragraph 29, 30, 43, and 44 and figure 10).

It is further noted that "during original prosecution, Patent Owner specified that "[e]ncoding using a key and an algorithm is known" " (see Ex. 2, Prosecution History at 519).

Beetcher '072 further teaches, in relation to claim 13, **encoding said first code resource to form an encoded first code resource** in the teaching software modules producing compiled target code, where the executable code contains qualification triggers performed by the lock checking feature. Here a program template is created and input into a translator 127 that generates the qualification triggers and inserts them into the target code (see paragraphs 28, 34, and 38 and figure 7) and **decode resource is configured to decode said encoded first code resource upon receipt of said first license key** in the teaching of the user being prompted to enter a key to enable software functionality, where the computer uses a decode key to decode the license code (see paragraphs 24, 32, and 33 and figures 4 and 9). Here a 'qualification key' 111 is input via the console, lock release routine 430 makes the machine key, and subsequently the lock release routine 430 decodes the qualification grant key 11 using a machine key (see paragraph 40).

Patent Owner further noted that "during original prosecution, Patent Owner specified that "[e]ncoding using a key and an algorithm is known" " (see Ex. 2, Prosecution History at 519).

Beetcher '072 further teaches, in relation to claim 14's **software code interrelationships between code resources that result in a specified underlying functionality**, software modules producing compiled target code, where the executable code contains qualification triggers performed by the lock checking feature. Here a program template is created and input into a translator 127 that generates the qualification triggers and inserts them into the target code then 'solves references' after insertion (see paragraphs 28, 34, and 38 and figure 7)

It is further noted that "during original prosecution, Patent Owner specified that "interrelationships between code resource are not that which is novel" " (see Ex. 2, Prosecution History at 519).

It is agreed that Beetcher '072 as proposed in the request, raises a SNQ with respect to claims 11-14 of the '842 patent. There is a substantial likelihood that a reasonable Examiner would consider this teaching important in deciding whether or not these claims are patentable.

Accordingly, the Beetcher '072 reference raises a substantial new question to claims 11-14, which question has not been decided in a previous examination of the '842 patent nor was there a final holding of invalidity by the Federal Courts regarding the '842 patent.

Cooperman

The Requestor alleges that Cooperman raises a substantial new question of patentability with respect to claims 11-14 of the '842 patent. For purposes of Reexamination the reading of Cooperman on the claims is provided on pages 81-102 of the Request.

In the '842 patent, the Cooperman reference was listed in the References Cited section, but Cooperman was not subject to any rejection or prior art discussion during the original prosecution. Therefore, the request presents Cooperman in a new light.

Cooperman teaches a method for protecting software through the use of license code. Cooperman accomplishes copyright protection by using a utility application to select 'essential code resources' and encodes the 'data resources' using a key in a stegacipher process. These 'essential code resources' are then not accessible just by installing the software, rather a key will be needed to access the software (see page 10, lines 7-29). Cooperman teaches that a mapping is added at assemble time which specifies which 'data resource' a particular 'code resources' is encoded as. When programs are first run, the system prompts the user to enter a license code, then generates a decoding key to access the 'essential code resource' (see page 11, lines 9-33).

Cooperman teaches, with respect to claim 11, **using license information entered via said input in response to said prompt in a routine designed to decode**

a first license code encoded in said software product, that the application asks the user for a license code and then using the license code it can then generate the proper decode key to access essential code resources (see column 11, lines 24-34).

Cooperman teaches, with respect to claim 12, to **provide said specified underlying functionality only after receipt of said first license key** in teaching accomplishing copyright protection by using a utility application to select 'essential code resources' and encodes the 'data resources' using a key in a stegacipher process. These 'essential code resources' are then not accessible just by installing the software, rather a key will be needed to access the software (see page 10, lines 7-29).

It is further noted that "during original prosecution, Patent Owner specified that "[e]ncoding using a key and an algorithm is known" " (see Ex. 2, Prosecution History at 519).

Cooperman further teaches, in relation to claim 13, **encoding said first code resource to form an encoded first code resource** in the teaching that a mapping is added at assemble time which specifies which 'data resource' a particular 'code resources' is encoded as (see page 3, lines 10-31, page 8, lines 25-30, page 10, lines 8-31) and **decode resource is configured to decode said encoded first code resource upon receipt of said first license key** in the teaching the system prompts the user to enter a license code, then generates a decoding key to access the 'essential code resource' (see page 11, lines 9-33).

It is further noted that "during original prosecution, Patent Owner specified that "[e]ncoding using a key and an algorithm is known" " (see Ex. 2, Prosecution History at 519).

Cooperman further teaches, in relation to claim 14's software code interrelationships between code resources that result in a specified underlying functionality, that the application contains a special code resource ('memory scheduler') which knows all about other code resources in memory, and enables the application to call the memory scheduler to access the stored list of all memory addresses (see column 14, line 35 through column 15, line 17).

It is further noted that "during original prosecution, Patent Owner specified that "interrelationships between code resource are not that which is novel" " (see Ex. 2, Prosecution History at 519).

It is agreed that Cooperman as proposed in the request, raises a SNQ with respect to claims 11-14 of the '842 patent. There is a substantial likelihood that a reasonable Examiner would consider this teaching important in deciding whether or not these claims are patentable.

Accordingly, the Cooperman reference raises a substantial new question to claims 11-14, which question has not been decided in a previous examination of the '842 patent nor was there a final holding of invalidity by the Federal Courts regarding the '842 patent.

Hasebe

The Requestor alleges that Hasebe raises a substantial new question of patentability with respect to claims 11-14 of the '842 patent. For purposes of Reexamination the reading of Hasebe on the claims is provided on pages 103-124 of the Request.

Hasebe is new art that provides new, non-cumulative technological teachings that were not previously considered and discussed on the record during prosecution or reexamination of the '842 patent.

Hasebe teaches, with respect to claim 11, **using license information entered via said input in response to said prompt in a routine designed to decode a first license code encoded in said software product**, a request of information for removal of restrictions provided to a user terminal, where decoding unit 20 decodes a license code encoded in the software via a decode routine that uses the encoded license information (see column 7, lines 17-31 and column 9, line 19-35).

Hasebe teaches, with respect to claim 12, to **provide said specified underlying functionality only after receipt of said first license key** in teaching routine 25 that determines whether the user's license information is legitimate and if so permits access to the main program routine 26 (see column 7, line 55 through column 8, line 9 and

column 9, lines 25-35). Thereby software is executed only when the legitimacy of the license file is verified (see column 8, lines 21-23).

It is further noted that "during original prosecution, Patent Owner specified that "[e]ncoding using a key and an algorithm is known" " (see Ex. 2, Prosecution History at 519).

Hasebe further teaches, in relation to claim 13, **encoding said first code resource to form an encoded first code resource** in the teaching encoding the software and a **decode resource is configured to decode said encoded first code resource upon receipt of said first license key** in the teaching making the conversion information for decoding the encoded software wherein license information is required for execution (see column 4, lines 48-58, column 8, lines 47-53, and column 2, line 43 through column 3, line 15).

It is further noted that "during original prosecution, Patent Owner specified that "[e]ncoding using a key and an algorithm is known" " (see Ex. 2, Prosecution History at 519).

Hasebe further teaches, in relation to claim 14's **software code interrelationships between code resources that result in a specified underlying functionality**, a license display routine 25 and a main program 26 where in the main program there are defined the operating procedures relating to the proper functions of the software; in license display routine 25, there is defined the content to be executed

prior to execution of main program 26. When the software is actuated the CPU, by checking the contents ID in the license file, decides whether or not data corresponding to the software that is being actuated is presented in the license file. If corresponding date exists a comparison of contents of the license file and the signature information is performed to confirm legitimacy (see column 7, line 54 through column 8, line 9).

It is further noted that "during original prosecution, Patent Owner specified that "interrelationships between code resource are not that which is novel" " (see Ex. 2, Prosecution History at 519).

It is agreed that Hasebe as proposed in the request, raises a SNQ with respect to claims 11-14 of the '842 patent. There is a substantial likelihood that a reasonable Examiner would consider this teaching important in deciding whether or not these claims are patentable.

Accordingly, the Hasebe reference raises a substantial new question to claims 11-14, which question has not been decided in a previous examination of the '842 patent nor was there a final holding of invalidity by the Federal Courts regarding the '842 patent.

Summary

Claims 11-14 will be reexamined as requested in the Request.

Waiver of Right to File Patent Owner Statement

Patent Owner has **not** agreed to waive its right to file a Patent Owner Statement under 37 C.F.R. 304. PTO personnel confirmed that the 'patent owner did not agree to waive its right to file a patent owner's statement' via telephone call with Ricard Neifeld on 5/22/2018.

Conclusion

Extensions of time under 37 CFR 1.136(a) do not apply in reexamination proceedings. The provisions of 37 CFR 1.136 apply only to "an applicant" and not to parties in a reexamination proceeding. Further, in 35 U.S.C. 305 and in 37 CFR 1.550(a), it is required that reexamination proceedings "will be conducted with special dispatch within the Office."

The patent owner is reminded of the continuing responsibility under 37 CFR 1.565(a) to apprise the Office of any litigation activity, or other prior or concurrent proceeding, involving the patent throughout the course of this reexamination proceeding. The requester is also reminded of the ability to similarly appraise the Office of any such activity or proceeding throughout the course of this reexamination proceeding. See MPEP § § 2207, 2282, and 2286.

All correspondence relating to this *ex parte* reexamination proceeding should be directed:

By Mail to: Mail Stop Ex Parte Reexam
Central Reexamination Unit
Commissioner for Patents
United States Patent & Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450

By FAX to: (571) 273-9900
Central Reexamination Unit

By hand: Customer Service Window
Randolph Building
401 Dulany Street
Alexandria, VA 22314

By EFS-Web:

Registered users of EFS-Web may alternatively submit such correspondence via the electronic filing system EFS-Web, at

<https://efs.uspto.gov/efile/myportal/efs-registered>

EFS-Web offers the benefit of quick submission to the particular area of the Office that needs to act on the correspondence. Also, EFS-Web submissions are “soft scanned” (i.e., electronically uploaded) directly into the official file for the reexamination proceeding, which offers parties the opportunity to review the content of their submissions after the “soft scanning” process is complete.

Any inquiry concerning this communication or earlier communications from the Reexamination Legal Advisor or Examiner, or as to the status of this proceeding, should be directed to the Central Reexamination Unit at telephone number (571) 272-7705.

/DENNIS BONSHOCK/
Primary Examiner, Art Unit 3992

Conferees:

/ADAM L BASEHOAR/
Primary Examiner, Art Unit 3992

/ALEXANDER KOSOWSKI/
Supervisory Patent Examiner, Art Unit 3992

Order Granting Request For Ex Parte Reexamination	Control No. 90/014,138	Patent Under Reexamination 9104842
	Examiner DENNIS BONSHOCK	Art Unit 3992

--The MAILING DATE of this communication appears on the cover sheet with the correspondence address--

The request for *ex parte* reexamination filed 16 May 2018 has been considered and a determination has been made. An identification of the claims, the references relied upon, and the rationale supporting the determination are attached.

Attachments: a) PTO-892, b) PTO/SB/08, c) Other: _____

1. The request for *ex parte* reexamination is GRANTED.

RESPONSE TIMES ARE SET AS FOLLOWS:

For Patent Owner's Statement (Optional): TWO MONTHS from the mailing date of this communication (37 CFR 1.530 (b)). **EXTENSIONS OF TIME ARE GOVERNED BY 37 CFR 1.550(c).**

For Requester's Reply (optional): TWO MONTHS from the **date of service** of any timely filed Patent Owner's Statement (37 CFR 1.535). **NO EXTENSION OF THIS TIME PERIOD IS PERMITTED.** If Patent Owner does not file a timely statement under 37 CFR 1.530(b), then no reply by requester is permitted.


/DENNIS BONSHOCK/ Primary Examiner, Art Unit 3992		
--	--	--

cc:Requester (if third party requester)

U.S. Patent and Trademark Office
PTOL-471G(Rev. 01-13)

Office Action in *Ex Parte* Reexamination

Part of Paper No. 20180606


Reexamination 	Application/Control No. 90014138	Applicant(s)/Patent Under Reexamination 9104842
	Certificate Date	Certificate Number

Requester Correspondence Address:	<input type="checkbox"/> Patent Owner	<input checked="" type="checkbox"/> Third Party
Joseph F. Edell Fisch Sigler LLP 5301 Wisconsin Ave, NW Fourth Floor Washington, DC 20015		

LITIGATION REVIEW <input checked="" type="checkbox"/>	/DGB/ (examiner initials)	06/13/2018 (date)
Case Name		Director Initials
Blue Spicke, LLC v. Juniper Networks, Inc.; Docket Number: 6		
Blue Spicke, LLC v. Suddenlink Communications et al.; Docket		
Blue Spicke, LLC v. Comcast Corporation D/B/A Xfinity et al.;		
Blue Spicke, LLC v. Charter Communications, Inc.; Docket Numb		
Blue Spicke, LLC v. Altice Usa, Inc.; Docket Number: 6:18cv2		

COPENDING OFFICE PROCEEDINGS	
TYPE OF PROCEEDING	NUMBER

--	--

Search Notes 	Application/Control No. 90014138	Applicant(s)/Patent Under Reexamination 9104842
	Examiner DENNIS BONSHOCK	Art Unit 3992

CPC- SEARCHED		
Symbol	Date	Examiner

CPC COMBINATION SETS - SEARCHED		
Symbol	Date	Examiner

US CLASSIFICATION SEARCHED			
Class	Subclass	Date	Examiner

* See search history printout included with this form or the SEARCH NOTES box below to determine the scope of the search.

SEARCH NOTES		
Search Notes	Date	Examiner
Previous Prosecution Searched	6/13/2018	DGB
Litigation Searched	6/13/2018	DGB

INTERFERENCE SEARCH			
US Class/ CPC Symbol	US Subclass / CPC Group	Date	Examiner

--	--

Doc code: IDS

Doc description: Information Disclosure Statement (IDS) Filed

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it contains a valid OMB control number.

PTO/SB/08a (01-10)

Approved for use through 07/31/2012. OMB 0851-0031

U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

INFORMATION DISCLOSURE STATEMENT BY APPLICANT (Not for submission under 37 CFR 1.99)	Application Number	RE of Patent No. 9,104,842
	Filing Date	2007-08-24
	First Named Inventor	Scott A. Moskowitz
	Art Unit	TBD
	Examiner Name	TBD
	Attorney Docket Number	

U.S.PATENTS								
Examiner Initial*	Cite No	Patent Number	Kind Code ¹	Issue Date	Name of Patentee or Applicant of cited Document	Pages,Columns,Lines where Relevant Passages or Relevant Figures Appear		
	1	5933497		1999-08-03	Robert Carl Beetcher			
	2	5935243		1999-08-10	Takayuki Hasebe			
If you wish to add additional U.S. Patent citation information please click the Add button.								
U.S.PATENT APPLICATION PUBLICATIONS								
Examiner Initial*	Cite No	Publication Number	Kind Code ¹	Publication Date	Name of Patentee or Applicant of cited Document	Pages,Columns,Lines where Relevant Passages or Relevant Figures Appear		
	1							
If you wish to add additional U.S. Published Application citation information please click the Add button.								
FOREIGN PATENT DOCUMENTS								
Examiner Initial*	Cite No	Foreign Document Number ³	Country Code ² i	Kind Code ⁴	Publication Date	Name of Patentee or Applicant of cited Document	Pages,Columns,Lines where Relevant Passages or Relevant Figures Appear	T ⁵
	1	05-334072	JP		1993-12-17	Robert Carl Beetcher		<input type="checkbox"/>
	2	9726732	WO		1997-07-24	Scott A. Moskowitz		<input type="checkbox"/>

EFS Web 2.1.17

ALL REFERENCES CONSIDERED EXCEPT WHERE LINED THROUGH. /D.G.B/

INFORMATION DISCLOSURE STATEMENT BY APPLICANT (Not for submission under 37 CFR 1.99)	Application Number	RE of Patent No. 9,104,842
	Filing Date	2007-08-24
	First Named Inventor	Scott A. Moskowitz
	Art Unit	TBD
	Examiner Name	TBD
	Attorney Docket Number	

If you wish to add additional Foreign Patent Document citation information please click the Add button

NON-PATENT LITERATURE DOCUMENTS			
Examiner Initials*	Cite No	Include name of the author (in CAPITAL LETTERS), title of the article (when appropriate), title of the item (book, magazine, journal, serial, symposium, catalog, etc), date, pages(s), volume-issue number(s), publisher, city and/or country where published.	T ⁵
	1		<input type="checkbox"/>

If you wish to add additional non-patent literature document citation information please click the Add button

EXAMINER SIGNATURE			
Examiner Signature	/DENNIS G BONSHOCK/	Date Considered	06/14/2018

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609. Draw line through a citation if not in conformance and not considered. Include copy of this form with next communication to applicant.

¹ See Kind Codes of USPTO Patent Documents at www.USPTO.GOV or MPEP 901.04. ² Enter office that issued the document, by the two-letter code (WIPO Standard ST.3). ³ For Japanese patent documents, the indication of the year of the reign of the Emperor must precede the serial number of the patent document. ⁴ Kind of document by the appropriate symbols as indicated on the document under WIPO Standard ST.16 if possible. ⁵ Applicant is to place a check mark here if English language translation is attached.

INFORMATION DISCLOSURE STATEMENT BY APPLICANT (Not for submission under 37 CFR 1.99)	Application Number	RE of Patent No. 9,104,842
	Filing Date	2007-08-24
	First Named Inventor	Scott A. Moskowitz
	Art Unit	TBD
	Examiner Name	TBD
	Attorney Docket Number	

CERTIFICATION STATEMENT

Please see 37 CFR 1.97 and 1.98 to make the appropriate selection(s):

That each item of information contained in the information disclosure statement was first cited in any communication from a foreign patent office in a counterpart foreign application not more than three months prior to the filing of the information disclosure statement. See 37 CFR 1.97(e)(1).

OR

That no item of information contained in the information disclosure statement was cited in a communication from a foreign patent office in a counterpart foreign application, and, to the knowledge of the person signing the certification after making reasonable inquiry, no item of information contained in the information disclosure statement was known to any individual designated in 37 CFR 1.56(c) more than three months prior to the filing of the information disclosure statement. See 37 CFR 1.97(e)(2).

See attached certification statement.

The fee set forth in 37 CFR 1.17 (p) has been submitted herewith.

A certification statement is not submitted herewith.

SIGNATURE

A signature of the applicant or representative is required in accordance with CFR 1.33, 10.18. Please see CFR 1.4(d) for the form of the signature.

Signature	/Joseph F. Edell/	Date (YYYY-MM-DD)	2018-05-16
Name/Print	Joseph F. Edell	Registration Number	67,625

This collection of information is required by 37 CFR 1.97 and 1.98. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 1 hour to complete, including gathering, preparing and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. **SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.**

Privacy Act Statement

The Privacy Act of 1974 (P.L. 93-579) requires that you be given certain information in connection with your submission of the attached form related to a patent application or patent. Accordingly, pursuant to the requirements of the Act, please be advised that: (1) the general authority for the collection of this information is 35 U.S.C. 2(b)(2); (2) furnishing of the information solicited is voluntary; and (3) the principal purpose for which the information is used by the U.S. Patent and Trademark Office is to process and/or examine your submission related to a patent application or patent. If you do not furnish the requested information, the U.S. Patent and Trademark Office may not be able to process and/or examine your submission, which may result in termination of proceedings or abandonment of the application or expiration of the patent.

The information provided by you in this form will be subject to the following routine uses:

1. The information on this form will be treated confidentially to the extent allowed under the Freedom of Information Act (5 U.S.C. 552) and the Privacy Act (5 U.S.C. 552a). Records from this system of records may be disclosed to the Department of Justice to determine whether the Freedom of Information Act requires disclosure of these records.
2. A record from this system of records may be disclosed, as a routine use, in the course of presenting evidence to a court, magistrate, or administrative tribunal, including disclosures to opposing counsel in the course of settlement negotiations.
3. A record in this system of records may be disclosed, as a routine use, to a Member of Congress submitting a request involving an individual, to whom the record pertains, when the individual has requested assistance from the Member with respect to the subject matter of the record.
4. A record in this system of records may be disclosed, as a routine use, to a contractor of the Agency having need for the information in order to perform a contract. Recipients of information shall be required to comply with the requirements of the Privacy Act of 1974, as amended, pursuant to 5 U.S.C. 552a(m).
5. A record related to an International Application filed under the Patent Cooperation Treaty in this system of records may be disclosed, as a routine use, to the International Bureau of the World Intellectual Property Organization, pursuant to the Patent Cooperation Treaty.
6. A record in this system of records may be disclosed, as a routine use, to another federal agency for purposes of National Security review (35 U.S.C. 181) and for review pursuant to the Atomic Energy Act (42 U.S.C. 218(c)).
7. A record from this system of records may be disclosed, as a routine use, to the Administrator, General Services, or his/her designee, during an inspection of records conducted by GSA as part of that agency's responsibility to recommend improvements in records management practices and programs, under authority of 44 U.S.C. 2904 and 2906. Such disclosure shall be made in accordance with the GSA regulations governing inspection of records for this purpose, and any other relevant (i.e., GSA or Commerce) directive. Such disclosure shall not be used to make determinations about individuals.
8. A record from this system of records may be disclosed, as a routine use, to the public after either publication of the application pursuant to 35 U.S.C. 122(b) or issuance of a patent pursuant to 35 U.S.C. 151. Further, a record may be disclosed, subject to the limitations of 37 CFR 1.14, as a routine use, to the public if the record was filed in an application which became abandoned or in which the proceedings were terminated and which application is referenced by either a published application, an application open to public inspections or an issued patent.
9. A record from this system of records may be disclosed, as a routine use, to a Federal, State, or local law enforcement agency, if the USPTO becomes aware of a violation or potential violation of law or regulation.

Reexamination Control Number: 90014138
Confirmation No: 6880
RE: US Patent 9104842

Filing via FAX to: 5712733744
Attn: Alexander Kosowski, Central Reexamination Unit, USPTO

Statement Supporting Re-Filing

During the period surrounding and including when the patent owner statement was due in this case, the USPTO was experiencing an electronic emergency. All electronic filing systems were offline, and many of the USPTO's internal systems were offline. Basically, the USPTO was a disaster area during this time.

The undersigned filed and served a patent owner statement in this case. Filing was by Priority Mail Express, and service was by Priority Mail.

On 9/5/2018, Manuel Saldana, paralegal in central reexam contacted the undersigned by telephone to request re-filing of patent owner statements in Reexamination Control Numbers 90014137 and 90014138, in response to a note in the file that patent owner statements had been filed in each reexam. The undersigned asked specifically what Mr. Saldana was instructing, given the rule 1.248 service requirement for filings in ex parte reexams. Mr. Saldana instructed the undersigned to telephone Alexander Kosowski, at 5712723744.

On 9/5/2018, the undersigned telephoned Alexander Kosowski, at 5712723744. Mr. Kosowski's vm replied, and the undersigned stated relevant facts and requested Mr. Kosowski call the undersigned.

On 9/5/2018, Mr. Kosowski called back, noted that the USPTO had notes in the file that the patent owner had filed a patent owner statement in this case, but had no record of the patent owner statement. Mr. Kosowski admitted his understanding from internal PTO communications that there remained some 10,000 or so faxes and mailed paper documents in patent cases that had still not been matched with their file, as a consequence of the electronic emergency.

After some discussion, and based upon suggestions from Mr. Kosowski, the undersigned and Mr. Kosowski agreed on the following course of action to provide the missing documents for this proceeding.

The undersigned would prepare and fax to Mr. Kasowski at his USPTO facsimile transmission number, 5712733744, proof of filing of the patent owner statement, the facts and circumstances leading to that fax, and a certificate of service of the foregoing facsimile transmission.

The undersigned would re-file, via the USPTO's EFS (Electronic Filing System), into the corresponding USPTO electronic file for this proceeding, electronic versions of the paper documents constituting the patent owner's previously filed and served patent owner statement. The undersigned intends to provide the document descriptions "Miscellaneous Incoming" to each filing, and Mr. Kasowski indicated the USPTO paralegals would change the document description accordingly, thereafter.

Accordingly, included with this document via facsimile to 5712733744, find five more pages number "Reexamination Control Number: 90014138 Page 1 of 5" to "Reexamination

Control Number: 90014138 Page 5 of 5." These pages are:

Page 1, Priority Mail Express mailing slip, tracking number ending in 6720, dated 08/20/2018, addressed to the address specified by regulations for Ex Parte reexamination filings, and Click-N-Ship Label Record.

Page 2 Label Record, Priority Mail mailing slip, tracking number ending in 5749, dated 08/20/2018, addressed to the address specified for the reexamination requestor, and Click-N-Ship Label Record.

Page 3, copies of USPS receipt of packages for filing and service, showing receipt of packages having tracking numbers ending in 6720 and 7329, dated 08/20/2018

Page 4, USPS web site Tracking information for package having tracking number ending in 6720, showing USPS receipt 8/20/2015, 5:40 PM, and USPS deliver to the USPTO on 8/21/2018, 9:12 AM.

Page 5, USPS web site Tracking information for package having tracking number ending in 7329, showing USPS receipt 8/20/2015, 5:40 PM, and USPS deliver to the reexamination requestor on 8/21/2018, 9:12 AM.

Page 6, showing snippet of a Windows Explorer view containing file "CertificateOfService_90014138.pdf", showing this file has a date modified and a date created of 8/20/2018, 6:34 PM.

Page 7, Certificate of Service contained in file "CertificateOfService_90014138.pdf" shown in the prior page, showing service by "first class mail (priority mail)" on the reexamination requestor, dated 8/17/2018, of documents constituting patent owner statement, showing document names and page counts. (Note that this date (8/17/2018) is in error and should state 8/20/2018; this is an artifact of the fact that the undersigned was delayed from 8/17 to 8/20 in filing, related to the necessity to paper file during the USPTO's electronic emergency, and the undersigned failed to update the date in the Certificate, prior to service.)

These pages show that the patent owner Priority-Mail-Express filed the patent owner statement on 8/20/2018 and served the reexamination requestor on 8/20/2018 by Priority Mail.

Accordingly, based upon the discussions with Mr. Kasowski, the patent owner will, on the same day, (likely 9-6-2018), serve the reexam requester a copy of this Statement Supporting Re-Filing and evidentiary pages 1-7 noted above, and attempt to re-file the previous paper filing, via EFS.

Truly, /Richard Neifeld/
Richard Neifeld, Reg. No. 35,299
Attorney for patent owner.

Reexamination Control Number: 90014138
Confirmation No: 6880
RE: US Patent 9104842

37 CFR 1.248 Certificate of Service

I served by first class mail (priority mail) to the third party reexamination requestor's correspondence address:



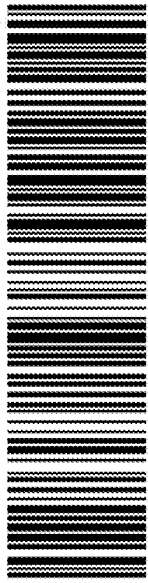
Attn: Joseph P. Edell
FISCH SIGLER LLP
5301 WISCONSIN AVENUE, NW
FOURTH FLOOR
WASHINGTON, DC 20015

the following documents:

A copy of this Certificate
Statement Supporting Re-Filing
Evidentiary Pages 1-7 noted in the Statement Supporting Re-Filing

Date of Service: 9/6/2018

/RichardNeifeld/
Richard Neifeld, Reg. No. 35,299
Attorney for patent owner



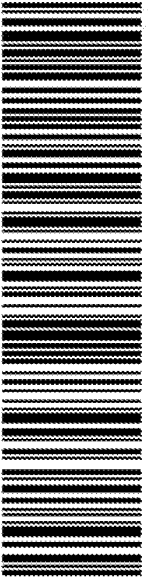
 UNITED STATES POSTAL SERVICE® Click-N-Ship®	
	
<small>usps.com</small> US POSTAGE <small>9470 1036 9930 0039 1367 20 0247 0002 0012 2918</small> \$24.70	<small>08/20/2018 2:18:03 PM</small> <small>Mailed from 22312 06250000001306</small>
PRIORITY MAIL EXPRESS 1-DAY™	
SCHEDULED DELIVERY DATE: 08/21/18 0007	
WAIVER OF SIGNATURE SCHEDULED DELIVERY 12:00 PM	
SHIP TO: COMMISSIONER FOR PATENTS PO BOX 1450 MAIL STOP EX PARTE REEXAM ALEXANDRIA VA 22313-1450	
DANIEL H SACHS NEIFELD IP LAW, PC 5400 SHAWNEE RD STE 310 ALEXANDRIA VA 22312-2300	
USPS TRACKING #  9470 1036 9930 0039 1367 20	

Use Priority Mail Express packaging or stickers. Securely affix label to mail piece. Do not tape over barcode. Service guarantees begin with the acceptance processing of this item when brought to a USPS retail location, or when the item returns to the Post Office after being collected during delivery/collection or from a Priority Mail Express Collection box. This Online Record must be presented to Postal personnel if applying for a service related refund. Refunds for unused postage paid labels can be requested online 30 days from the print date.

 PRIORITY MAIL EXPRESS™		Click-N-Ship® Label Record		DO NOT MAIL	
PO ZIP Code 22312		<input checked="" type="checkbox"/> 1-Day <input type="checkbox"/> 2-Day 12:00 PM		Flat Rate <input type="checkbox"/>	
Weight lbs 2 ozs 0		Address to PO Box <input type="checkbox"/>		Postage \$24.70	
<input type="checkbox"/> Saturday <input checked="" type="checkbox"/> Sunday/Holiday		Contents Value \$1.00		COD Fee <input type="checkbox"/>	
				Ins Fee \$0.00	
		Total Postage & Fees \$24.70		Ship Date: 08/20/2018 Scheduled Delivery Date: 08/21/2018 Insured Value: \$1.00	
				Signature Service:	
				 9470 1036 9930 0039 1367 20	

LABEL INFORMATION	
FROM: DANIEL H SACHS NEIFELD IP LAW, PC 5400 SHAWNEE RD STE 310 ALEXANDRIA VA 22312-2300	TO: COMMISSIONER FOR PATENTS PO BOX 1450 MAIL STOP EX PARTE REEXAM ALEXANDRIA VA 22313-1450
FOR PICKUP OR TRACKING GO TO USPS.COM usps.com	

USPS Employee: For service failure refunds, follow standard refund procedures.
 Reexamination Control Number: 90014138 Page 1 of 7

 UNITED STATES POSTAL SERVICE® Click-N-Ship®	 <small>usps.com</small> 9405 8036 9930 0681 5573 29 0072 5002 0012 0016 US POSTAGE \$7.25
	<small>08/20/2018 2 lb 0 oz Mailed from 22116 06250000001311</small>
PRIORITY MAIL 1-DAY™ <small>Expected Delivery Date: 08/21/18</small>	
DANIEL H SACHS NEIFELD IP LAW, PC 5400 SHAWNEE RD STE 310 ALEXANDRIA VA 22312-2300	
SHIP TO: JOSEPH EDELL FISCH SIGLER LLP 5301 WISCONSIN AVE NW # 4 WASHINGTON DC 20015-2015	
USPS TRACKING #  9405 8036 9930 0681 5573 29	
Electronic Rate Approved #038555749	

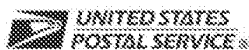
Cut on dotted line.

Instructions

- Each Click-N-Ship® label is unique. Labels are to be used as printed and used only once. **DO NOT PHOTO COPY OR ALTER LABEL.**
- Place your label so it does not wrap around the edge of the package.
- Adhere your label to the package. A self-adhesive label is recommended. If tape or glue is used, **DO NOT TAPE OVER BARCODE.** Be sure all edges are secure.
- To mail your package with PC Postage®, you may schedule a Package Pickup online, hand to your letter carrier, take to a Post Office™, or drop in a USPS collection box.
- Mail your package on the "Ship Date" you selected when creating this label.

Click-N-Ship® Label Record

USPS TRACKING # / Insurance Number: 9405 8036 9930 0681 5573 29			
Trans. #:	442189532	Priority Mail® Postage:	\$7.25
Print Date:	08/20/2018	Insurance Fee:	\$0.00
Ship Date:	08/20/2018	Total:	\$7.25
Expected Delivery Date:	08/21/2018		
Insured Value:	\$1.00		
From: DANIEL H SACHS NEIFELD IP LAW, PC 5400 SHAWNEE RD STE 310 ALEXANDRIA VA 22312-2300			
To: JOSEPH EDELL FISCH SIGLER LLP 5301 WISCONSIN AVE NW # 4 WASHINGTON DC 20015-2015			
<small>* Retail Pricing Priority Mail rates apply. There is no fee for USPS Tracking® service on Priority Mail service with use of this electronic rate shipping label. Refunds for unused postage paid labels can be requested online 30 days from the print date.</small>			



Thank you for shipping with the United States Postal Service!
 Check the status of your shipment on the USPS Tracking® page at usps.com

Reexamination Control Number: 90014138 Page 2 of 7

Filing Service

MERRIFIELD
8409 LEE HWY
MERRIFIELD
VA
22116-9998
5165400241

08/20/2018 (800)275-8777 5:40 PM

Product Description	Sale Qty	Final Price
---------------------	----------	-------------

Prepaid Mail	1	
(Weight:1 lbs. 7.20 oz.)		
(Destination:WASHINGTON, DC 20015)		
(Acceptance Date:08/20/2018 17:40:38)		
(Label #:420200159405803699300681557329)		

Total \$0.00

In a hurry? Self-service kiosks offer quick and easy check-out. Any Retail Associate can show you how.

Preview your Mail
Track your Packages
Sign up for FREE @
www.informedelivery.com

All sales final on stamps and postage
Refunds for guaranteed services only
Thank you for your business

HELP US SERVE YOU BETTER

TELL US ABOUT YOUR RECENT
POSTAL EXPERIENCE

Go to:
<https://postalexperience.com/Pos>

840-5220-0061-001-00045-51679-02

or scan this code with
your mobile device:



or call 1-800-410-7420.

YOUR OPINION COUNTS

Bill #: 840-52200061-1-4551679-2
Clerk: 16

Filing Express

MERRIFIELD
8409 LEE HWY
MERRIFIELD
VA
22116-9998
5165400241

08/20/2018 (800)275-8777 5:40 PM

Product Description	Sale Qty	Final Price
---------------------	----------	-------------

Prepaid Mail	1	
(Weight:1 lbs. 7.20 oz.)		
(Destination:ALEXANDRIA, VA 22313)		
(Acceptance Date:08/20/2018 17:40:02)		
(Label #:420223139470103699300039136720)		

Total \$0.00

In a hurry? Self-service kiosks offer quick and easy check-out. Any Retail Associate can show you how.

Preview your Mail
Track your Packages
Sign up for FREE @
www.informedelivery.com

All sales final on stamps and postage
Refunds for guaranteed services only
Thank you for your business

HELP US SERVE YOU BETTER

TELL US ABOUT YOUR RECENT
POSTAL EXPERIENCE

Go to:
<https://postalexperience.com/Pos>

840-5220-0061-001-00045-51666-02

or scan this code with
your mobile device:



or call 1-800-410-7420.

YOUR OPINION COUNTS

Bill #: 840-52200061-1-4551666-2
Clerk: 16

File Edit View History Bookmarks Tools Help

USPS.com® - USPS Tracking® X

https://tools.usps.com/track.htm 67%

usps

Allergies/Ingredients - ... Most Visited What You Need For U... Neifeld IP Law, PC, Pat... maxval patent alerts - ...

Tracking Number: 42022313847010069900000136720 Remove X

Scheduled Delivery by **TUESDAY** **21** AUGUST 2018 **12:00pm** by **12:00pm**

Status **Delivered**
 August 21, 2018 at 9:12 am
 Delivered
 ALEXANDRIA, VA 22313
 Get Updates

Delivered

Text & Email Updates

Proof of Delivery

Tracking History

August 21, 2018, 9:12 am
 Delivered
 ALEXANDRIA, VA 22313
 *Your item was delivered at 9:12 am on August 21, 2018 in ALEXANDRIA, VA 22313.

August 21, 2018, 7:27 am
 Arrived at Post Office
 STERLING, VA 20155

August 21, 2018, 7:22 am
 Arrived at USPS Regional Facility
 DULLES VA DISTRIBUTION CENTER

August 20, 2018, 9:32 pm
 Arrived at USPS Regional Facility
 DULLES VA DISTRIBUTION CENTER

August 20, 2018, 7:37 pm
 Accepted at USPS Origin Facility
 ALEXANDRIA, VA 22312

August 20, 2018, 5:49 pm
 USPS in possession of item
 MERRIFIELD, VA 22119

Product Information

Track Another Package +

Tracking Number: 420200159405803699300081587329

Remove X

On Time

Status

Expected Delivery on

Delivered

TUESDAY

August 21, 2018 at 11:01 am
Delivered, In/At Mailbox
WASHINGTON, DC 20018

21 AUGUST 2018 by 8:00pm

Get Updates

Delivered

Text & Email Updates

Tracking History

August 21, 2018, 11:01 am

Delivered, In/At Mailbox

WASHINGTON, DC 20018

Your item was delivered in or at the mailbox at 11:01 am on August 21, 2018 at WASHINGTON, DC 20018.

August 21, 2018, 8:00 am

Arrived at Unit

WASHINGTON, DC 20018

August 21, 2018, 8:31 am

Departed USPS Regional Facility

WASHINGTON DC DISTRIBUTION CENTER

August 21, 2018, 8:58 am

Arrived at USPS Regional Facility

WASHINGTON DC DISTRIBUTION CENTER

August 21, 2018, 8:22 am

Departed USPS Regional Facility

MERRIFIELD VA DISTRIBUTION CENTER

August 20, 2018, 8:16 pm

Arrived at USPS Regional Origin Facility

MERRIFIELD VA DISTRIBUTION CENTER

August 20, 2018, 7:00 pm

Accepted at USPS Origin Facility

MERRIFIELD, VA 22118

August 20, 2018, 6:40 pm

USPS in possession of item

MERRIFIELD, VA 22118

Reexamination Control Number: 90014138 Page 5 of 7

90014138_USP9104842_SCOT0014-4 Prosecution

Name	Date modified	Type	Size	Date created
AttachmentsFiled.zip	8/20/2018 6:41 PM	Compressed (zipp...	13,477 KB	8/20/2018 6:39 PM
CertificateOfService_90014138.pdf	8/20/2018 6:34 PM	Adobe Acrobat D...	8 KB	8/20/2018 6:34 PM
2018-8-20_ProofOfServiceAndFilingPOStatement_90014138.pdf	8/20/2018 6:31 PM	Adobe Acrobat D...	52 KB	8/20/2018 6:31 PM
Label-442189532-1.pdf	8/20/2018 5:16 PM	Adobe Acrobat D...	1,946 KB	8/20/2018 5:16 PM
Label-442189532.pdf	8/20/2018 5:08 PM	Adobe Acrobat D...	1,974 KB	8/20/2018 5:08 PM
OrderGrantingRequest_90014138_USP9104842_SCOT0014-4_8-19-2...	8/20/2018 2:57 PM	Adobe Acrobat D...	877 KB	8/17/2018 9:02 PM
90014138_ReexamRequest.pdf	8/17/2018 5:16 PM	Adobe Acrobat D...	4,717 KB	8/17/2018 5:16 PM

Reexamination Control Number: 90014138
Confirmation No: 6880
RE: US Patent 9104842

37 CFR 1.234 Certificate of Service

I served by first class mail (priority mail) to the third party reexamination requestor's correspondence address:

Attn: Joseph P. Edell
FISCH SIGLER LLP
5301 WISCONSIN AVENUE, NW
FOURTH FLOOR
WASHINGTON, DC 20015

the following documents:

Certificate of Service (1 page), showing service by first claims mail (express mail) of:
This Patent Owner Statement (22 pages)
Attachment 1, Pages 1-28 from the file history of application 08/587,793 (issued as USP5745569) (27 Pages)
Attachment 2, USP5745569 (6 pages)
Attachment 3, Pages 1-12 and 172-192 of the transcript of the deposition of Marc S. Cooperman, May 17, 2018, in Blue Spike LLC v. Juniper Networks, Inc., civil case 6:17-cv-00016-KNM. (33 Pages)
Attachment 4, Comparison of Disclosures of US application 08/587,943; WO 97/26732; USP5,745,569; USP9021602; and USP9104842. (5 pages)
Attachment 7, 2002 Settlement Agreement (37 pages)
Attachment 9, Title Abstract for application, 08587943, now USP5745569. (2 Pages)
Attachment 10, a claim support chart showing support in 08587943 filed 1/17/1996; USP5745569; and USP9104842 for the citations used by the reexam request to show disclosure of the claimed subject matter by the Cooperman reference. (6 pages)

Date of Service: 8/17/2018

/RichardNeifeld/
Richard Neifeld, Reg. No. 35,299
Attorney for patent owner

Electronic Acknowledgement Receipt

EFS ID:	33646670
Application Number:	90014138
International Application Number:	
Confirmation Number:	7638
Title of Invention:	DATA PROTECTION METHOD AND DEVICE
First Named Inventor/Applicant Name:	9104842
Customer Number:	31518
Filer:	Richard A. Neifeld
Filer Authorized By:	
Attorney Docket Number:	
Receipt Date:	06-SEP-2018
Filing Date:	16-MAY-2018
Time Stamp:	15:27:58
Application Type:	Reexam (Third Party)

Payment information:

Submitted with Payment	no
------------------------	----

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)/ Message Digest	Multi Part /.zip	Pages (if appl.)
1	Miscellaneous Incoming Letter	RefilingNoticeAndCOC_90014138_IMAGE.pdf	2956795 fb1646759018c3da78c92fb01917da82d6da639a	no	10

Warnings:

Information:	
Total Files Size (in bytes):	2956795
<p>This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.</p> <p><u>New Applications Under 35 U.S.C. 111</u> If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.</p> <p><u>National Stage of an International Application under 35 U.S.C. 371</u> If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.</p> <p><u>New International Application Filed with the USPTO as a Receiving Office</u> If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.</p>	

Y:\Clients\SCOT Scott A Moskowitz and Wistaria Trading, Inc\90014138, USP9104842,
SCOT0014-4\Drafts\PatentOwnerStatement.wpd

Reexamination control number: 90014138
Confirmation number: 7638
RE: United States patent 9104842

37 CFR 1.530 Patent Owner Statement

I. Summary

I.1 Cooperman and Hasebe references are not prior art

Either Moskowitz or Moskowitz and Cooperman conceived of the inventions defined by the claims 11-14 before any prior art date of Cooperman or Hasebe. Therefore, the Cooperman or Hasebe grounds fail.

After confirming that Mr. Cooperman was not currently represented by counsel, the undersigned recently contacted Mr. Cooperman and recently asked him to determine whether he contributed to the conception of any claim in USP9104842. The undersigned, as of today, awaits Mr. Cooperman's determination.

If Mr. Cooperman concludes that he did so contribute, then the undersigned will petition for a certificate of correction to correct the inventorship of the patent by naming Mr. Cooperman, in which case the inventive entities of USP9104842 and USP5745569 will be the same, which would prove invention of claims 11-14 not later than 1/17/1996.

If, on the other hand, Mr. Cooperman determines he is not an inventor of the subject matter of any claim, including claims 11-14 of USP9104842, then that is proof that the subject matter defined by claims 11-14, by its disclosure in USP5745569 and US application 08/587,793 is proof that Mr. Moskowitz invented that subject matter prior to the dates of availability of the Cooperman and Hasebe references. Mr. Cooperman recently testified in the related district court litigation and the transcript of his testimony suggests that he was not an inventor of at least some of the subject matter defined by claims in the subject patent, USP9104842.

The subject matter of claims 11-14 is disclosed in prior patent USP5745569 which issued from US application 08/587,793 filed 1/17/1997, and this same subject matter is disclosed in US application 08/587,793 and also in USP USP9104842. See Attachment 1, Pages 1-28 from the file history of application 08/587,793 (issued as USP5745569) (27 Pages); Attachment 2, USP5745569 (6 pages); and attachment 10, a claim support chart showing support in 08587943 filed 1/17/1996; USP5745569; and USP9104842 for the citations used by the reexam request to show disclosure of the claimed subject matter by the Cooperman reference.

So in either case, neither of the Cooperman and Hasebe references are legal prior art.

If Mr. Cooperman refuses to cooperate, then that fact would also indicate he is not an inventor of the subject matter defined by claims 11-14 in USP9104842.

I.2 Beetcher497 and Beetcher072 references do not disclose any claim

Beetcher072 is a Japanese language reference and translation and is a less complete version of the Beetcher497 US patent. Beetcher072 adds nothing relevant to the disclosure of Beetcher497. Beetcher (both 497 and 072) fail to disclose or suggest the subject matter defined by claims 11-14. Therefore, the grounds based upon these references fail.

In very brief summary, the Beetcher references disclose a customer's computer system decrypting an encrypted key (entitlement key 111) to enable the computer's use of installed software modules that are installed without entitlement to run. Beetcher USP5993447 col. 4:13. Specified values in the key (Fig. 2, key 111, flags 205 and version field 203) control which installed software modules the computer is entitled to use. The software modules are not encoded using this key. In fact, Beetcher discloses that every distributed software module is identical. USP5993447 col. 6:2-4. Only the keys that Beetcher distributes to the customers vary, to selectively enable execution of the software modules that customer has licensed. USP5993447 col. 6:4-6. The distributed key is not licensed software, its just a code that the customer's computer uses to enable the computer to use licensed software modules. USP5993447 col. 2-7 and col. 6:20-40.

Beetcher discloses encrypting the key for a customer by using that particular customer's computer's hard wired serial number, and distributing that encrypted key to the customer's computer, and Beetcher disclose the customer's computer decrypting the encrypted key using its hard wired serial number. USP5993447 col. 4:5-12. But encrypting and decrypting the *key* is not encoding or decoding of any *software module*. The key is not software; just data.

Beetcher discloses the customer's computer compares the software version number contained in the key (or zero, if the key has a flag indicating the customer is not licensed to use the software) with the software version embedded in the installed software, and enabling the software to run only when the software version number contained in the key is not less than the software version embedded in the installed software. See USP5993447 col. 9:49 to col. 10:19; and col. 10:52-58, Fig. 10 step 1006 (storing a value that is either the key's version number or zero, depending upon the value of a flag bit in the key) and comparing (Fig. 10 step 1006) that value to the software version in verification instruction 301. But that is a *comparison* of two values, and not *decoding* of anything.

Thus, Beetcher does not disclose encoding or decoding of *software* using a key, or decoding *license code encoded in software*. Beetcher's license key is its entitlement key 111. Beetcher's entitlement verification triggering instruction 301 is an instruction to compare the software's version number with the key's version number. See USP5993447 trigger instruction 1004, Fig. 10, col. 10:52-54. The trigger instruction 1004 instructs the computer to perform the comparison step 1006. See USP5993447 trigger instruction 1006, Fig. 10, col. 10:54-60. Beetcher's entitlement verification triggering instruction 301 is merely an instruction to perform that comparison, it does not authorize Beetcher's installed software to run. In fact, Beetcher's verification triggering instruction 301 is what locks the software so that it cannot run, unless and until the computer receives a license key, which is entitlement key 111, and which unlocks the software module. The unlock of the installed software occurs at Fig. 10, step 1006, when the comparison result is "No", thereby resulting in step 1003, checking for "More Instructions?" in

which the computer checks for more object code instructions in the executable software module 300, of Fig. 3. See USP5993447 col. 10:51-53; col. 6:43-46; col. 9:18-20.

I.3 What Beetcher does not disclose.

Consequently, Beetcher: does not disclose using information entered into a computer to decode a license code encoded in a software product (see claim 11); does not disclose encoding software code using a license key and an encoding algorithm such that the software will only run after the software receives the license key (see claim 12); does not disclose modifying software code stored in memory using a license key and an encoding algorithm that provides an encoded code resource and a decode resource configured to decode the encoded first code resource upon receipt of the license key (see claim 13); and does not disclose encoding software codes stored in computer memory using a license key and an encoding algorithm, so that at least one interrelationship between code resources of the software code is encoded (see claim 14).

As shown below, these and other limitations not disclosed by Beetcher (shown bolded) are defined by claims 11-4.

I.4 Beetcher does not disclose the highlighted limitations of claims 11-14.

Claim 11 reads: "11. A method for licensed software use, the method comprising: loading a software product on a computer, said computer comprising a processor, memory, an input, and an output, so that said computer is programmed to execute said software product; said software product outputting a prompt for input of license information; and **said software product using license information entered via said input in response to said prompt in a routine designed to decode a first license code encoded in said software product.**"

Regarding claim 11, Beetcher does not disclose decoding a license code encoded in the software; does not disclose decoding a license code encoded in the software; and does not disclose a software product using the license information in such a routine.

Claim 12 reads "12. A method for encoding software code using a computer having a processor and memory, comprising: storing a software code in said memory; wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system; and **encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key encoded software code**; and wherein, when installed on a computer system, **said first license key encoded software code will provide said specified underlying functionality only after receipt of said first license key.**"

Regarding claim 12, Beetcher does not disclose encoding the software code using a license key and that the software must subsequently receive that license key to function.

Claim 13 reads "13. A method for encoding software code using a computer having a processor and memory, comprising: storing a software code in said memory; wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system; and **modifying, by said computer, using a first license key and an encoding algorithm, said software code, to form a modified software code**; and

wherein said modifying comprises encoding said first code resource to form an encoded first code resource; wherein said modified software code comprises said **encoded first code resource, and a decode resource for decoding said encoded first code resource**; wherein **said decode resource is configured to decode said encoded first code resource upon receipt of said first license key.**"

Regarding claim 13, Beetcher does not disclose using a first license key and an encoding algorithm, [and] said software code, to form a modified software code; and does not disclose that modifying comprises encoding said first code resource to form an encoded first code resource and also a decode resource; and does not disclose that this decode resource is configured to decode said encoded upon the software code's receipt of the license key.

Claim 14 reads "14. A method for encoding software code using a computer having a processor and memory, comprising: storing a software code in said memory; wherein said software code defines software code interrelationships between code resources that result in a specified underlying functionality when installed on a computer system; and **encoding, by said computer using at least a first license key and an encoding algorithm, said software code**, to form a first license key encoded software code **in which at least one of said software code interrelationships are encoded.**"

Regarding claim 14, Beetcher does not disclose using a licensing key and an encoding algorithm to encode software code; and also does not disclose using a licensing key and encoding algorithm to encode interrelationships of code resources in the software code.

II. List of attachments

Certificate of Service (1 page), showing service by first claims mail (express mail) of:

This Patent Owner Statement (22 pages)

Attachment 1, Pages 1-28 from the file history of application 08/587,793 (issued as USP5745569) (27 Pages)

Attachment 2, USP5745569 (6 pages)

Attachment 3, Pages 1-12 and 172-192 of the transcript of the deposition of Marc S. Cooperman, May 17, 2018, in Blue Spike LLC v. Juniper Networks, Inc., civil case 6:17-cv-00016-KNM. (33 Pages)

Attachment 4, Comparison of Disclosures of US application 08/587,943; WO 97/26732; USP5,745,569; USP9021602; and USP9104842. (5 pages)

Attachment 7, 2002 Settlement Agreement (37 pages)

Attachment 9, Title Abstract for application, 08587943, now USP5745569. (2 Pages)

Attachment 10, a claim support chart showing support in 08587943 filed 1/17/1996; USP5745569; and USP9104842 for the citations used by the reexam request to show disclosure of the claimed subject matter by the Cooperman reference. (6 pages)

III. SNQs

Decision pages 1-20 found a substantial new question (SNQ) of patentability for claims 11-14.

The decision, at page 10, found a SNQ of claims 11-14 based upon Beetcher497 "as proposed in the request."

The decision, at page 13, found a SNQ of claims 11-14 based upon Beetcher072 "as proposed in the request."

The decision, at page 16, found a SNQ of claims 11-14 based upon Cooperman "as proposed in the request." The decision at page 14 states that "the reading of Cooperman on the claims is provided on pages 81-102 of the Request."

The decision, at page 16, found a SNQ of claims 11-14 based upon Hasebe "as proposed in the request."

IV. Why Cooperman and Hasebe Are Not Prior Art

Pages 81-102 of the request show that the Cooperman reference discloses the subject matter defined by claims 11-14. The examiner relied upon pages 81-102 in the reexam request, in the order granting reexamination and concluding that Cooperman presented a SNQ of patentability. The patent owner agrees that Cooperman discloses the subject matter defined by claims 11-14. Cooperman is WO 97/26732 which claims priority to US application 08/587,943. Cooperman and US application 08/587,943 have identical disclosure. Therefore, 08/587,943 also discloses the subject matter defined by claims 11-14. And this is confirmed by the claim chart of Attachment 10. US application 08/587,943 was filed 1/17/1996. This is proof that the subject matter defined by claims 11-14 was invented by someone not later than the 1/17/1996. US application 08/587,943 as filed and as issued as US5745569, names Moskowitz and Cooperman as inventors. US application 08/587,943 is proof that either Cooperman; Moskowitz; or Cooperman and Moskowitz, invented what is disclosed in WO 97/26732 not later than 1/17/1996. Application 08/587,943 issued as US5745569 showing that it's disclosure was not abandoned suppressed, dedicated, or concealed.

V. Why Beetcher Does Not Disclose or Suggest Claims 11-14

V.1 Beetcher

V.1.1 Distributing software modules and keys

Beetcher discloses that a distributor sends every customer the same generic set of software modules to install on their computers. USP5933497, col. 6:2-4 ("Each customer is shipped the same generic set of software modules, irrespective of which ones the customer is licensed to use."); see also col. 4:34-39.

Beetcher discloses that a distributor sends an encrypted version of an entitlement key to a user. USP5933497, col. 5:43-44 ("Encrypted entitlement key 111 is distributed from a sales office 121 of the distributor.") Beetcher's key consists of "serial number of the computer for which the software is licensed, together with a plurality of entitlement bits indicating which software modules are entitled to run on the machine." USP5933497, Abstract.

Beetcher discloses that each customer gets a unique entitlement key, which enables only

those software modules the customer has licensed. USP5933497, col. 4:39-44 ("Each customer will receive a unique entitlement key, enabling the customer to run only those software modules to which he is licensed. It makes no difference that the customer is given some modules to which he is not licensed, as he will be unable to execute them without the appropriate key.")

Beetcher discloses that the user enters their encrypted entitlement key into their computer system. USP5933497, col. 5:59-64 ("operator can enter it [sic; Encrypted entitlement key 111 is] into system 101 by typing the key on console 109.").

V.1.2 Decrypting (decoding) the encrypted entitlement key 111

Beetcher discloses that the highest version number of a software module the customer's computer is entitled to run is stored in product lock table 460. This information is obtained by the customer's computer, solely from the entitlement key 111 and the computer hard coded serial number.

Beetcher's computer system includes a decoder for decoding the encrypted entitlement key 111. USP5933497, col. 6:66-67 ("Computer system 101 contains means for receiving and decoding encrypted entitlement key 111...").

Beetcher discloses storing the encrypted entitlement key in memory. USP5933497, col. 8:23-27. ("Encoded product key table 450 is shown in FIG. 5. The table is contained in random access memory 104, and duplicated on a non-volatile storage device so it can be recovered if the system must be powered down or otherwise re-initialized.")

Beetcher discloses that user's computer stores a "permanent, unalterable" serial number 410 in hardware 401. USP5933497, Fig. 4, and col. 7:15-16.

Beetcher refers to "microcode." Microcode means instruction set which is stored permanently in a computer. See for example <https://en.wikipedia.org/wiki/Microcode> ("Microcode is a computer hardware technique that imposes an interpreter between the CPU hardware and the programmer-visible instruction set architecture of the computer.[1] As such, the microcode is a layer of hardware-level instructions"); and <https://en.oxforddictionaries.com/definition/us/microcode> ("A very low-level instruction set which is stored permanently in a computer or peripheral controller and controls the operation of the device.")

Beetcher discloses that the microcode's get machine key function 420, retrieves the computer's permanent serial number 410, and generates the machine key. USP5933497, Fig. 4, and col. 7:23-27; and col. 9:57-59.

Beetcher discloses that unlock routine 430 uses the machine key to decode entitlement key 111. USP5933497, Fig. 4 and col. 9:59-60 ("Unlock routine 430 then uses the machine key to decode the entitlement key 111 at step 903.").

V.1.3 Storing data based upon the decrypted (decoded) entitlement key 111

Beetcher discloses that each entry in table 450 contains a copy of the encrypted entitlement key. USP5933497 col. 8:28-29 ("Each entry 501 comprises a complete copy of the encrypted entitlement key....")

Beetcher discloses that set lock function 421 sets the version number in each entry in product lock table 460 based upon the entitlement flag and version number in entitlement key 111. USP5933497, col. 7:25-26; and col. 19:14-15 ("Unlock routine 430 will therefore invoke set lock function 421 to set the version number in product lock table 460"); and col. 10: 31-33 ("set lock function 421 is invoked to set the version number in the entry 601 in product lock table 460"). The entitlement flag and version number are data in the entitlement key 111. See Fig. 2; version bits 202, entitlement flags 205. Thus, Beetcher discloses that the version data in product lock table 460 is derived only from the data in entitlement key 111.

Beetcher discloses that each entry in product lock table 460 contains a number indicating the maximum version level of the corresponding software module entitled to run on that computer. USP5933497, col. 8:46-50 ("Table 460 contains 80 entries 601, one corresponding to each possible product number. Each entry contains a version number indicating the maximum version level of entitlement. A version number of 0 indicates no entitlement to any version of the product.")

V.1.4 Verifying entitlement is a comparison of data derived solely from the entitlement key 101 to the version number in the entitlement verification instruction 301

Beetcher discloses that the user's computer's invokes the microcode, check lock function 422, every time execution encounters one of the entitlement verification triggering instructions 301 in a software module. USP5933497 Fig. 4, "Check Lock 422"; col. 10:48-55 ("The process for executing a software module according to the preferred embodiment is shown in FIG. 10. System 101 executes the module by fetching (step 1001) and executing (step 1002) object code instructions until done (step 1003). If any instruction is an entitlement verification triggering instruction 301 (step 1004) check lock function 422 is invoked.")

Beetcher discloses that check lock function 422 accesses the product lock table entry 601 and compares the version number in table entry 601 with the version number in the triggering instruction. USP5933497 col. 10:54-65 ("Check lock function 422 accesses the product lock table entry 601 corresponding to the product number contained in the triggering instruction at step 1005. If the version number in product lock table 460 is equal to or greater than the version number 303 contained in triggering instruction 301, the software is entitled to execute (step 1006). In this case, check lock function 422 takes no further action, and the system proceeds to execute the next object code instruction in the software module. If the software is not entitled, check lock function generates an exception condition, causing control to pass to exception handler 432, which will terminate program execution (step 1007).")

To reiterate, look at Fig. 10.

Fig. 10 shows step 1004 named "Trigger". Beetcher discloses this step 1004 invokes check lock function 422. USP5933497 col. 10:42-54 ("If any instruction is an entitlement verification triggering instruction 301 (step 1004) check lock function 422 is invoked.")

Fig. 10 shows that if the instruction in the software module is an entitlement verification triggering instruction 301, then step 1005 executes. Beetcher discloses this step 1005 has is check lock function 422 accessing the product lock table 460's entry 601 for the corresponding

software module. USP5933497 col. 10:52-57 ("If any instruction is an entitlement verification triggering instruction 301 (step 1004) check lock function 422 is invoked. Check lock function 422 accesses the product lock table entry 601 corresponding to the product number contained in the triggering instruction at step 1005.")

Fig. 10 shows that step 1006 executes after step 1005. Beetcher identifies step 1006 in Fig. 10 as "Entitled Version < Product Version?", and Beetcher indicates this is a comparison step. USP5933497 col. 10:56-65 ("If the version number in product lock table 460 is equal to or greater than the version number 303 contained in triggering instruction 301, the software is entitled to execute (step 1006). In this case, check lock function 422 takes no further action, and the system proceeds to execute the next object code instruction in the software module. If the software is not entitled, check lock function generates an exception condition, causing control to pass to exception handler 432, which will terminate program execution (step 1007).")

V.1.5 Important points relative to incorrect assertions in the reexam request

The following points are important to note in connection with the incorrect assertions in the reexam request.

It is important to note that this describes a comparison operation, not a software encode or decode operation. No value of the software module is changed as a result of this comparison operation.

An important point is that Beetcher discloses the machine key is used to decrypt (decode) Beetcher's encrypted entitlement key and is unrelated to and never applied to modify, encode, or decode, Beetcher's software module.

An important point is that Beetcher's encrypted entitlement key 111 is what a user of Beetcher's computer (system 101, Figs. 1, 4) enters into Beetcher's computer system in response to a prompt, and what specified entitlement to use software, that is functionally embodies a license right.

An important point is that Beetcher discloses that Beetcher's software modules are distributed without any license code encoded therein. USP5933497 col. 4:4-5 ("Software is distributed according to the present invention without entitlement to run."); col. 4:34-35 ("Because the software itself does not contain any entitlement, no restrictions on the distribution are necessary."); col. 6:2-7 ("Each customer is shipped the same generic set of software modules, irrespective of which ones the customer is licensed to use. The separately distributed entitlement key contains information enabling system 101 to determine which software modules are entitled to execute on it.")

V.1.6 Beetcher Does Not Disclose Claim 11.

Compare those important points to the language of claim 11. Claim 11 requires:

said software product using license information entered via said input in response to said prompt in a routine designed to decode a first license code encoded in said software product.

The only thing Beetcher discloses entering in response to a prompt is the encrypted entitlement key 111. So Beetcher's encrypted entitlement key 111 must correspond to claim 11's "license information entered via said input in response to said prompt." This is entirely consistent with Beetcher's correspondence of the entitlement bits of the entitlement key to the user's license of software modules, in Beetcher's Summary of his invention. USP5933497 col. 4:5-8 ("The key includes the serial number of the machine for which the software is licensed, together with a plurality of entitlement bits indicating which software modules are entitled to run on the machine.")

Claim 11 then requires using that license information entered in response to the prompt be used "in a routine designed to decode a first license code *encoded in said software product.*"

Beetcher discloses no license code encoded in his software modules. As noted above, Beetcher at USP5933497 col. 4:4-5; col. 4:34-35; col. 6:2-7 discloses that Beetcher's software modules are distributed without any license code encoded therein. Beetcher's software modules, as distributed, are non functional modules. Not because they contain un-decoded license codes, but because they contain a number of identical triggering instructions 301, embedded in object code, each requiring entitlement verification, for the software to run. USP5933497, col. 6:45-65. Accordingly, Beetcher' discloses no license code encoded in a software product.

Moreover, Beetcher does not disclose decoding anything encoded in the software product. Specifically, Beetcher does not disclose decoding of triggering instructions 301. Instead, Beetcher discloses that triggering instructions 301 are instructions executed by executed by the microcode check lock function 422. USP5933497 col. 8:19-22 ("The executable code contains entitlement verification triggering instructions 301 (only one shown), which are executed by horizontal microcode check lock function 422.") Check lock function 422 compares a value for version number of verification triggering instructions 301 with a corresponding value in product lock table entry 601, USP5933497 col. 10:54-60 ("Check lock function 422 accesses the product lock table entry 601 corresponding to the product number contained in the triggering instruction at step 1005. If the version number in product lock table 460 is equal to or greater than the version number 303 contained in triggering instruction 301, the software is entitled to execute (step 1006).") (The version number in table 460 is set in response to values of an entitlement key 111; see USP5933497 col. 9:48 to 10:19.). Thus, Beetcher does not disclose that triggering instructions 301 is decoded.

V.2 The Reexam Request Fails to Show That Beetcher Discloses Claim 11

V.2.1 The reexam request corresponds Beetcher's encrypted entitlement key 111 to claim 11's "license information entered via said input in response to said prompt."

First, the reexam request corresponds Beetcher's encrypted entitlement key 111 with claim 11's "license information." Request page 23 to page 24:10 ("a user interface routine for the customer to input a license key into the computer before the product can be used"); and 24:10-12 ("Beetcher details that the customer enters entitlement key 111, i.e., license

Y:\Clients\SCOT Scott A Moskowitz and Wistaria Trading, Inc\90014138, USP9104842, SCOT0014-4\Drafts\PatentOwnerStatement.wpd

information, in response to the prompt").

In response, thus, the reexam request admits that input of Beetcher's encrypted entitlement key 111 corresponds to claim 11's "license information entered via said input in response to said prompt."

V.2.2 The only software product that Beetcher discloses that includes a user interface routine for the customer to input a license key into the computer, is Beetcher's virtual machine operating system

Second, the reexam request, page 23:3-4 asserts that "Beetcher explains that its software product includes a user interface routine for the customer to input a license key into the computer before the product can be used."

In response, that statement is correct only for Beetcher's virtual machine operating system installation, and not for any other software module. As Beetcher explains, "special install input routine 440 is required to input the key during initial installation of the operating system." So it is this special input routine 440, only, and only for installation of the operating system, that uses information entered via a user input interface to enable the operating system to function.

Beetcher explains that general input routine 441 is part of the operating system. USP5933497 col. 7:66 to 8:1 ("This operation system support at virtual machine level 404 contains two user interface routines needed to support input of the entitlement key.") Beetcher explains that general input routine 441 handles input of entitlement keys, subsequent to installation of the operating system. USP5933497 col. 8:1-3. Therefore, other than for initial installation of Beetcher's operating system, Beetcher does not disclose loading a software product on computer 101 *in which the same software product* uses an entitlement key 111 (licensing information) entered into a user interface in response to a prompt. In contrast, claim 11 requires "loading a software product on a computer, ... *said software product* using license information entered via said input in response to said prompt." Instead, for all software loads subsequent-to-operating-system-installation, Beetcher discloses the operating system's "[g]eneral input routine 441" using the license information.

V.2.3 Beetcher does not teach "loading a software product" that then decodes a "license code encoded in said software product," as required by claim 11.

Note a conventional definition of "decrypt" is to "make (a coded or unclear message) intelligible". See for example the dictionary definition at URL: <https://en.oxforddictionaries.com/definition/decrypt>.

Note that a conventional definition of "decode" is to "Convert (a coded message) into intelligible language." See for example the dictionary definition at URL: <https://en.oxforddictionaries.com/definition/decode>.

Beetcher uses "decrypt" and "decode" interchangeable when references the decryption of encrypted entitlement key 111. See for example USP5933497 col. 10:27-31 ("Unlock routine 430 then fetches the *encrypted* entitlement key from the appropriate entry in encoded product key table 450 at step 921, obtains the 30 machine key at step 922, and *decodes* the entitlement

key at step 923.") and the corresponding description in Fig. 4 of unlock routine 430 as "unlock (*decode* key)" and corresponding description in Fig. 8 of step 923 as "*Decrypt* entitlement key."

Third, the reexam request page 24:10-12 states that "After entering that key, Beetcher teaches that the customer's computer uses a decode key to initiate unlock routine 430 *to decode the license code encoded in the software product.*"

In response, that statement is incorrect. Beetcher does not teach decoding a license code *encoded in a software product*. Beetcher does not disclose a license code *encoded in a software product*. Beetcher also does not disclose *decoding* a license code encoded in a software product. Instead, the only thing Beetcher teaches decoding, is the encrypted entitlement key 111. USP5933497 col. 10:27-31 ("Unlock routine 430 then fetches the encrypted entitlement key from the appropriate entry in encoded product key table 450 at step 921, obtains the machine key at step 922, and decodes the entitlement key at step 923.") Encrypted entitlement key 111 is not contained in the software product loaded onto the Beetcher's computer 101; that key is "separately distributed" from the software. Beetcher abstract; see also col. 2:59-61 ("Non-entitlement means that the software as distributed is disabled, and requires a separately distributed authorization to be able to run."); and col. 4:2-3 summarizing Beetcher's invention as non entitlement software ("Software is distributed according to the present invention without entitlement to run.") Because Beetcher's entitlement key is not contained in the software product, Beetcher does not correspond to decoding "a first license code *encoded in said software product,*" as required by claim 11. Because Beetcher's loaded software product does not decode a first license code encoded in that software product, Beetcher does not correspond to claim 11's "said software product using ... a routine designed to decode a first license code encoded in said software product."

V.2.4 Claim 11 Requires the Software Product to have the license code encoded, when the Software Product is loaded onto the computer.

Keep in mind that claim 11 defines the software product as that which was loaded on the computer: "loading a software product on a computer ... a first license code encoded in said software product." Accordingly, the fact that the entitlement code is stored in computer system 101 in one or more tables, after the software is loaded on Beetcher's computer system 101, does not define a software program, as loaded onto computer system 101, encoding a license code.

As explained by the subject patent, the software product, as installed on the computer system, contains the encoding, when installed. And that is the broadest reasonable construction of claim 11. As explained by the subject patent, "When code and data resources are compiled and assembled into a precursor of an executable program the ... several essential code resources, and encode them into one or several data resources... [t]he purpose of this scheme is to make a particular licensed copy of an application distinguishable from any other." USP9104842, col. 13:9-32; and USP5745569 col. 5:40-63. And as explained by the subject patent the software product only must receive the license code, after installation on the user's computer, to function. USP9104842, col. 6:22-32; and USP5745569 col. 6:22-31 ("The application can then operate as follows: 1) when it is run for the first time, after installation. it asks the user for personalization

Y:\Clients\SCOT Scott A Moskowitz and Wistaria Trading, Inc\90014138, USP9104842, SCOT0014-4\Drafts\PatentOwnerStatement.wpd

information, which *includes the license code*. This can include a particular computer configuration; 2) it stores this information in a personalization data resource; 3) *Once it has the license code, it can then generate the proper decoding key* to access the essential code resources.") And as explained by the subject patent the decode key corresponds or is equal to the licence license code. USP9104842, col. 13:36-37; and USP5745569 col. 5:67 to 6:2 ("This method, then, is to choose the key so that it corresponds, is equal to, or is a function of, a license code.")

Therefore, Claim 11 requires the software product to have the license code encoded, when the software product is loaded onto the computer.

Thus, claim 11 does not read on a computer system in which a license code is encoded in the software product after the software product is loaded onto the computer. Accordingly, *even if*, and the patent owner does not believe this to be the case, Beetcher did disclose software being loaded onto computer system 111 that was, subsequent to being loaded on that computer system, encoded, and then decoding that encoded software, such a method would not read on claim 11.

V.2.5 Beetcher's Figures 4 and 9a do not show decoding a first license code encoded in the software product.

The reexam request's next sentence at page 24: 12-14, states "Beetcher's Figures 4 and 9a, which are provided below, show the software using the key (i.e., license information) entered by the customer to decode a first license code encoded in the software product."

In response, that statement is also incorrect. Beetcher does disclose a license code encoded in the software product. Beetcher does not disclose decoding license code encoded in the software product.

Beetcher Fig. 4 shows entitlement key 111 as an input to "unlock (decode key) 430. Beetcher explains that "Unlock routine 430 uses the unique machine key to decodes entitlement key 111." And entitlement key 111 is not part of any software product; it is the key delivered to the user and entered by the user in response to some prompt. Accordingly, Beetcher's disclosure of relating to Fig. 4 of unlock routine 430 contradicts the reexam request's assertion that Beetcher discloses a license code encoded in the software product.

Beetcher's Fig. 9a also contradicts the reexam request's assertion. Beetcher Fig. 9a, step 901 shows that entitlement key is input into the computer system. Beetcher col. :51-52 explains that "A customer enters entitlement key 111 into computer system 101 via console 109 at step 901." and Beetcher's Abstract for example shows that this key is separately distributed from the distributed software ("Software is distributed without entitlement to run, while a separately distributed encrypted entitlement key enables execution of the software.") Beetcher Fig. 9a, step 903 then specifies that this separately entered key is decrypted. Beetcher , which is followed by step 903 "decrypt entitlement key." Beetcher explains this decryption means decoding of the entitlement key 111. USP5933497 col. 9:59-60 ("Unlock routine 430 then uses the machine key to decode the entitlement key 111 at step 903."). Accordingly, Beetcher's disclosure of relating to Fig. 9a of unlock routine 430 contradicts the reexam request's assertion that Beetcher discloses a license code encoded in the software product.

Therefore, Beetcher's Figures 4 and 9a do not show decoding a first license code encoded in the software product.

V.2.7 The reexam request admits that Beetcher's decoding refers to the decoding of encrypted entitlement key 111.

The reexam request, at page 26:3-5 states "Beetcher's unlock routine 430 will complete the decoding process by building an encoded product key table (step 904), populating the key table for the relevant software product specified in the entitlement key (steps 905-9(8), and saving the key table (step 905-908)."

In response, that statement is an admission that Beetcher disclosure of decoding refers to decoding encrypted entitlement key 111. Encrypted entitlement key 111 is not part of Beetcher's distributed software.

V.2.8 Beetcher does not disclose that check lock function 422 decodes a first license code encoded in a software product.

The reexam request page 27:4-6 asserts that "Beetcher uses its license information in a routine, such as check lock function 422, designed to decode a first license code encoded in a software product via the triggering instructions."

In response, this statement is incorrect.

First, Beetcher's check lock function 422 is not part of Beetcher's software. Its microcode, that is, part of horizontal microcode 402. See USP5933497 Fig. 4 and col. 7:18-21 ("It [sic; horizontal microcode 402] is physically stored in control store 103, which in the preferred embodiment is a read-only memory (ROM) which *is not capable of alteration by the customer.*"); and col. 8:21-22("horizontal microcode check lock function 422."). Therefore, check lock function 422 is not part of a software product, and certainly not part of the software products Beetcher explains is distributed by Beetcher's distributor to Beetcher's computer system 101.

Second, Beetcher does not disclose that check lock function 422 decodes a first license code encoded in a software product. Beetcher discloses that check lock function 422 reads an entry in product lock table 460 to verify entitlement. ("Check lock function 422 accesses product lock table 460 and reads one of the entries to verify entitlement.") Reading and verifying are not decoding.

In support of its incorrect assertion that "Beetcher uses its license information in a routine, such as check lock function 422, designed to decode a first license code encoded in a software product via the triggering instructions," the reexam request at page 27:7 to page 28:2 quotes Beetcher col. 10:52-62. We repeat that quote below:

If any instruction is an entitlement verification triggering instruction 301 (step 1004) check lock function 422 is invoked. Check lock function 422 accesses the product lock table entry 601 corresponding to the product number contained in the triggering instruction at step 1005. If the version number in product lock

table 460 is equal to or greater than the version number 303 contained in triggering instruction 301, the software is entitled to execute (step 1006). In this case, check lock function 422 takes no further action, and the system proceeds to execute the next object code instruction in the software module. [USP5933497 Beetcher col. 10:52-62.]

This passage refers to steps 1004 to 1006 which appear in Beetcher's Fig. 10. Beetcher's Brief Description of Fig. 10 states "FIG. 10 is a block diagram of the steps required to verify entitlement during execution of a software module according to the preferred embodiment."

Beetcher Fig. 10 shows a decision point at 1004, determining if the fetched instruction is entitlement verification triggering instruction 301. If so, Fig. 10 shows that steps 105 and 106 are executed in sequence. Step 105 is "Access Designated Product Lock Table." Step 106 is a comparison, "Entitled Version < Product Version?". Neither step 105 nor step 106 performs a decode or decrypt operation on anything. That is, neither step 105 nor step 106 makes or converts the form of a message, and therefore neither step 105 nor step 106 performs a decode operation. Therefore, contrary to the assertion in the reexamination request, Beetcher col. 10:52-62 does not disclose decode a first license code.

Furthermore, Beetcher col. 10:52-62 does not disclose changing the form of entitlement verification triggering instruction 301. Therefore, Beetcher col. 10:52-62 does not disclose decoding of entitlement verification triggering instruction 301.

It should be noted that when referring to decoding and decrypting, Beetcher uses these terms interchangeably, see citations herein above. And Beetcher explains that, as is conventional in the art, Beetcher's decryption mechanism uses a key to perform the encryption and decryption. USP5933497 col. 4:1-12 ("Software is distributed according to the present invention without entitlement to run. A separately distributed encrypted entitlement key enables execution of the software. The key includes the serial number of the machine for which the software is licensed, together with a plurality of entitlement bits indicating which software modules are entitled to run on the machine. A secure decryption mechanism is contained on the machine. The decryption mechanism fetches the machine serial number and uses it as a key to decrypt the entitlement key.") It is and was old and well known in the art that key based encryption and decryption changed the form of the data being encrypted or decrypted. Beetcher does not change the form of the entitlement verification triggering instructions 301; it merely uses them for a comparison. That is neither decryption nor decoding.

For all these reasons, Beetcher does not disclose that check lock function 422 decodes a first license code encoded in a software product.

V.2.9 Beetcher does not indicate that encoding entitlement verification triggering instruction 301 in its col. 11:14-28 alternative embodiment is any different from encoding entitlement verification triggering instruction 301 in its primary embodiment, and in any case, "encoding" is not relevant to claim 11

Y:\Clients\SCOT Scott A Moskowitz and Wistaria Trading, Inc\90014138, USP9104842, SCOT0014-4\Drafts\PatentOwnerStatement.wpd

The reexam request at page 28:3-4, citing Beetcher USP5933497 col. 11:14-28, asserts that "Beetcher teaches that the triggering instructions will be encoded into the code resources controlling software functionality," concluding as a result, at page 28:17, that "Beetcher discloses claim 11."

In response, that assertion is incorrect, and also irrelevant to claim 11.

First, Claim 11 does not require encoding, only decoding of "a first license code encoded in said software product." Accordingly, the accuracy of this assertion is not relevant to claim 11.

Second, the encoding issue is a straw man argument. In the computer arts, a conventional definition of encoding is "the process of converting data from one form to another." See for example URL: <https://techterms.com/definition/encoding> (stating that "Home : Software Terms: Encoding Definition [--] Encoding [--] Encoding is the process of converting data from one form to another. While "encoding" can be used as a verb, it is often used as a noun, and refers to a specific type of encoded data. There are several types of encoding, including image encoding, audio and video encoding, and character encoding." interpolation supplied to indicate original formatting not reproduced herein).

Therefore both Beetcher's Fig. 2 embodiment of entitlement verification triggering instruction 301 and any other embodiment thereof are encoded.

USP5933497 col. 11:24-28 explains the alternative embodiment of the form of entitlement verification triggering instruction 301, stating:

the compiler can automatically generate the object code to perform the alternative function (and simultaneously, the entitlement verification trigger) as part of its normal compilation procedure.

All that means is that entitlement verification triggering instruction 301 is encoded in the object code; just like with the original embodiment.

This alternative embodiment discloses no change in the Fig. 10 algorithm for using the "entitlement verification triggering instruction 301" by comparing them in step 1006, to a value in product lock table 460 to determine whether to either abort (step 1007) or continue program execution (step 1003). Accordingly, the alternative embodiment at USP5933497 col. 11:14-28 fails to disclose, or suggest, decoding a first license code encoded in said software product. Claim 11, in contrast, requires "said software product using... a routine designed to decode a first license code encoded in said software product." Beetcher discloses no routine to decode a first license code encoded in said software product. Accordingly, Beetcher does not disclose claim 11.

V.3 The Reexam Request Fails to Show that Beetcher Discloses Claim 12

Claim 12 recites "12. A method for encoding software code using a computer having a processor and memory, comprising: storing a software code in said memory; wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system; and encoding, by said computer using at least a first

Y:\Clients\SCOT Scott A Moskowitz and Wistaria Trading, Inc\90014138, USP9104842, SCOT0014-4\Drafts\PatentOwnerStatement.wpd

license key and an encoding algorithm, said software code, to form a first license key encoded software code; and wherein, when installed on a computer system, said first license key encoded software code will provide said specified underlying functionality only after receipt of said first license key."

The reexam request pages 35-39 imply that "entitlement verification triggering instructions 301" correspond to claim 12's "license key" and that Beetcher Fig. 3's software module 300 is license key encoded software.

In response, Beetcher does not disclose Claim 12's "encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key encoded software code. This is because Beetcher's "entitlement verification triggering instructions 301" is not a "license key." Instead, it's a lock. It is what prevents the functioning of Beetcher's software modules, unless and until a customer enters Beetcher's "entitlement key 111."

V.3.1 A license key means something that unlocks code, not something that locks code, such as Beetcher's entitlement verification triggering instructions 301.

A "key" is something that unlocks. In contrast, Beetcher's "entitlement verification triggering instructions 301" is a lock. The specification of the subject patent explains that a license key is derived from a license code in describing the claimed invention:

For the encoding of the essential code resources, a "key" is needed. Such a key is similar to those described in U.S. Pat. No. 5,613,004, the "Steganographic Method and Device" patent. The purpose of this scheme is to make a particular licensed copy of an application distinguishable from any other. It is not necessary to distinguish every instance of an application, merely every instance of a license. A licensed user may then wish to install multiple copies of an application, legally or with authorization. **This method, then, is to choose the key so that it corresponds, is equal to, or is a function of, a license code or license descriptive information**, not just a text file, audio clip or identifying piece of information as desired in digital watermarking schemes extant and typically useful to stand-alone, digitally sampled content. **The key is necessary to access the underlying code, i.e., what the user understands to be the application program.** [USP9104842 col. 13:27-43; USP5745569 col. 5:58 to col. 6:8.]

The specification continues, stating "... 3) Once it has the license code, it can then generate the proper **decoding key to access the essential code resources.**" USP9104842 Col. 13:65-67; USP5745569 col. 6:29-32. This means that the application can function as intended only after it has the decoding key which it derives from the license code or key. The key unlocks the software.

Moreover, the conventional definition of a license key in the computer arts consistent with how it is used in the specification. A convention definition in the computer arts is "a data

Y:\Clients\SCOT Scott A Moskowitz and Wistaria Trading, Inc\90014138, USP9104842, SCOT0014-4\Drafts\PatentOwnerStatement.wpd

string that verifies authorized software product access." See URL: <https://www.techopedia.com/definition/26227/license-key> (stating "Definition - What does License Key mean? A license key is a data string that verifies authorized software product access. This type of software security helps prevent software piracy and gives organizations the ability to protect their software from unauthorized copying or sharing by unlicensed users."); see also URL: https://www.webopedia.com/TERM/L/license_key.html (stating "software license key [] By Vangie Beal []A software license key is a pattern of numbers and/or letters provided to licensed users of a software program. License keys are typically created and delivered via a license generator once a software user has paid for the software and has agreed to the conditions of use and distribution as legally specified in the software license (also known as an End-User License Agreement, or EULA)." Bracket added to indicate lien returns in the original.) This is exactly how the specifications of USP9104842 and USP5745569 use the term key.

The reexam's request of equating that which locks the software so it cannot run (verification triggering instructions 301) with a license key, which is something that enables software to run, is contrary to both the specification of the subject patent, and the meaning of a license key in the computer arts, and inconsistent.

V.3.2 The reexamination request's argument is inconsistent because Beetcher does not disclose the installed version of software module 300 receiving an entitlement verification triggering instructions 301, and even if it did, that would not unlock Beetcher's software

Claim 12 also requires that "**when installed on a computer system**, said first license key encoded software code *will provide said specified underlying functionality only after receipt of said first license key*" Even assuming arguendo that "entitlement verification triggering instructions 301" corresponds to a license key, and Beetcher's software module 300 corresponds to claim 12's "license key encoded software code," Beetcher's software module 300 does not provide the specified functionality "*only after receipt of*" "entitlement verification triggering instructions 301." In fact, Beetcher does not disclose the software module 300 as installed on the customer's computer 101 receiving "entitlement verification triggering instructions."

Moreover, even if Beetcher's software module 300 did receive (more) "entitlement verification triggering instructions," they would not enable module 300 to function. The only effect of Beetcher's "entitlement verification triggering instructions" is to invoke a check lock function 421 that disables software module 300 from functioning.

V.4 The Reexam Request Fails to Show That Beetcher Discloses Claim 13.

Claim 13 reads "13. A method for encoding software code using a computer having a processor and memory, comprising: storing a software code in said memory; wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system; and modifying, by said computer, using a first license key and an encoding algorithm, said software code, to form a modified software code; and wherein said modifying comprises encoding said first code resource to form an encoded first code

resource; wherein said modified software code comprises said encoded first code resource, and a decode resource for decoding said encoded first code resource; wherein said decode resource is configured to decode said encoded first code resource upon receipt of said first license key."

V.4.1 Beetcher's "entitlement verification triggering instructions 301" does not contain triggering information that unlock Beetcher's software.

The reexam request at pages 40-42 incorrectly asserts that Beetcher discloses claim 13's "modifying, by said computer, using a first license key and an encoding algorithm, said software code, to form a modified software code," arguing that Beetcher's "entitlement verification triggering instructions 301" correspond to the claimed "first license key." See reexam request, page 40, last two lines, stating "Beetcher's distributor system includes a computer that encodes software code using a first license key (e.g., triggering information)."

In response, the reexam request's implication that Beetcher's "entitlement verification triggering instructions 301" contains triggering information that unlock Beetcher's software is incorrect. In contrast, Beetcher clearly states that what "entitlement verification triggering instructions 301" triggers, is execution of instructions 1005, 1006 that will disable Beetcher's software. See Fig. 10, decision box 1004 "Trigger?", which invokes check lock function 422.

V.4.2 Beetcher does not disclose a resource that decodes Beetcher's software module

The reexam request at page 43-44 incorrectly alleges that Beetcher discloses "wherein said modified software code comprises said encoded first code resource, and a decode resource for decoding said encoded first code resource." The dashed lines contain the following and only the following elements from Fig. 4: "Unlock (Decode key) 430"; "Exception handler 432"; and "Check Lock 422."

In response, this assertion is incorrect.

First, the reexam request provides no explanation how those element meet the claim limitation.

Second, the reexam request fails to account for the requirement in the claim recitation that the "modified software code comprises .. a decode resource **for decoding said encoded first code resource;**" The reexam request fails to identify, and Beetcher does not disclose, the items in this dashed box functioning to decode anything in Beetcher's software module.

Third, Beetcher fails to disclose decoding anything in Beetcher's installed software module.

Fourth, "Check Lock 422" is not part of Beetcher's software module. Beetcher's "Check Lock 422" could not possibly be part of Beetcher's software module, because it is part of the microcode level 402, Fig. 4. And the microcode is part of the hardware level, in this case, embodied in ROM, and therefore not possibly part of Beetcher's installable software module. See USP5933497 col. 7:20-23 ("Horizontal microcode 402 contains microcode entries interpreting the executable instruction set. It is physically stored in control store 103, which in the preferred embodiment is a read-only memory (ROM) which is not capable of alteration by the customer.") The reexam request requires "Check Lock 422" to be part of the decode

Y:\Clients\SCOT Scott A Moskowitz and Wistaria Trading, Inc\90014138, USP9104842, SCOT0014-4\Drafts\PatentOwnerStatement.wpd

resource, which means that "Unlock (Decode key) 430"; "Exception handler 432"; alone, are insufficient to define a decode resource. However, claim 13 requires the "modified software code comprises ... a decode resource for decoding said encoded first code resource." Under the reexam's own correspondence of what defines a decode resource, Beetcher does not disclose that the modified software code comprises a decode resource, as required by claim 13. Therefore, Beetcher does not disclose claim 13.

V.4.2 Beetcher does not disclose a decode resource configured to decode an encoded first code resource upon receipt of a license key

Next, at pages 44-45, the reexam request incorrectly asserts that Beetcher discloses "wherein said decode resource is configured to **decode said encoded first code resource** upon receipt of said first license key," stating at page 44:1 to 45:1 that:

Beetcher discloses element 13.5. Beetcher specifies that its decode resource decodes the encoded first code resource upon receipt of the license key. Beetcher, for example, states that unlock routine 430 "fetches the encrypted entitlement key from ... table 450 ... and **decodes the entitlement key** The triggering instruction is then retried and program execution continues at step 928." [Bold added for emphasis.]

In response, note that the reexam request equated decoding of the entitlement key with the claimed "decode said encoded first code resource," which corresponds Beetcher's entitlement key 111 with the claim's "first encoded code resource." However, Beetcher's entitlement key is not a code resource, as defined in the subject patent. As stated in the subject patent:

The memory address of the first instruction in one of these sub-objects is called the "entry point" of the function or procedure. The rest of the instructions comprising that sub-object immediately follow from the entry point. Some systems may prefix information to the entry point which describes calling and return conventions for the code which follows, an example is the Apple Macintosh Operating System (MacOS). **These sub-objects can be packaged into what are referred to in certain systems as "code resources,"** which may be stored separately from the application, or shared with other applications, although not necessarily. Within an application there are **also data objects, which consist of some data to be operated on by the executable code.** These data objects are not executable. That is, they do not consist of executable instructions. The data objects can be referred to in certain systems as "resources."

This statement explains that a "code resource" as distinct from data objects operated on by the code. Beetcher's entitlement key 111 is a data object. It is operated on by code. It contains no executable instruction. It has no entry point.

The subject patent goes on to state that:

In general, any code resource can be considered "essential" in that if the program proceeds to a point where it must "call" the code resource and the code resource is not present in memory, or cannot be loaded, then the program fails.

That passage explains that a code resource is something that is subject to a "call". In the computer programming arts, "call" means to execute a software routine. In this passage, the word "call" is in quotes in the patent which clearly indicates it is a term of art.

In the computer and software arts, a conventional definition of "call" is at URL: <https://www.webopedia.com/TERM/C/call.html> which states:

To invoke a routine in a programming language. Calling a routine consists of specifying the routine name and, optionally, parameters. For example, the following is a function call in the C programming language:

```
printf("Hello")
```

The name of the function is printf and the parameter is "Hello." This function call causes the computer to display the word Hello on the display screen.

Beetcher's does not disclose that entitlement key 111 is a computer routine, and does not disclose calling entitlement key 111. Accordingly, the reexam request's implication that Beetcher's entitlement key 111 corresponds to claim 13's "encoded first code resource" is inconsistent with the specification of the subject patent and the definition of a call. Thus, Beetcher's entitlement key 111 is neither a code resource nor an encoded code resource. Therefore, Beetcher's structure for decoding entitlement key 111 does not correspond to claim 13's requirement for "a decode resource for decoding encoded first code resource."

V.5 Beetcher Does Not Disclose Claim 14.

Claim 14 reads "14. A method for encoding software code using a computer having a processor and memory, comprising: storing a software code in said memory; wherein said software code defines software code interrelationships between code resources that result in a specified underlying functionality when installed on a computer system; and encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key encoded software code in which at least one of said software code interrelationships are encoded."

The reexam request, page 49:4-14 concludes that Beetcher discloses claim 14's "encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key encoded software code." The reexam request incorporates its argument from reexam request section 12.3 and provides no other arguments. However, reexam request section 12.3, which appears on reexam request page 35, dealt with a

Y:\Clients\SCOT Scott A Moskowitz and Wistaria Trading, Inc\90014138, USP9104842, SCOT0014-4\Drafts\PatentOwnerStatement.wpd

more generic claim limitation. That is, reexam request section 12.3 dealt with the claim recitation "encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key encoded software code," in claim 12. This claim 12 recitation does not also require encoding of "software code interrelationships," which is required by claim 14.

In response, first, the reexam request is incorrect for the reasons explained herein above for claim 12's recitation in which we explained that Beetcher's "entitlement verification triggering instructions 301" is not a "license key." And therefore, Beetcher's software module 300 is not encoded using a license key, and therefore not "license key encoded software code."

Second, the reexam request fails to show that Beetcher "software code interrelationships" as required by claim 14. The reexam request equates the software code interrelationships to the references between code resources disclosed by Beetcher, stating on reexam request page 46:17-19 that:

And Beetcher specifies that translator 127 "resolves references" in the software code, which corresponds to defining code interrelationships between code resources.

But the reexam request fails to assert that Beetcher also discloses *encoding* these references.

The reexam request indicates that Beetcher "resolves" the references. But that merely means correcting the references to lines of executable code after inserting an entitlement verification triggering instruction 301 into the executable code so the code instructions execute sequentially. See USP5933497 col.9:12-20 ("Translator 127 automatically generates a substantial number of entitlement verification triggers, inserts them in random locations in the object code, and resolves references after the triggers have been inserted. The resultant executable object code form of the software module which is output at 705 contains the embedded triggers. This executable form of the module comprises object code instructions at executable code level 403.") This resolution, accounting for changes in the relative location of object code in memory upon addition of verification instructions 301, is not a conversion of the form of the interrelationships.

As shown by the definition herein above, "Encoding is the process of converting data from one *form* to another." Beetcher's resolving "references after the triggers have been inserted" does not disclose encoding, a change of form, and therefore does not disclose encoding of code interrelationships, as required by claim 14.

Therefore, the reexam request fails to show that Beetcher discloses claim 14.

Truly, /RichardNeifeld/
Richard Neifeld, Ref. No. 35,299
Attorney for patentee

Y:\Clients\SCOT Scott A Moskowitz and Wistaria Trading, Inc\90014138, USP9104842,

Y:\Clients\SCOT Scott A Moskowitz and Wistaria Trading, Inc\90014138, USP9104842,
SCOT0014-4\Drafts\PatentOwnerStatement.wpd

SCOT0014-4\Drafts\PatentOwnerStatement.wpd

Attachment 10

USP9104842 claim support chart showing support in 08587943 filed 1/17/1996; USP5745569 and USP9104842 for the citations used by the reexam request to show disclosure of the claimed subject matter by the Cooperman reference.

<p>USP9104842 Claim recitations, claims 11-14</p>	<p>Support in US application 08587943 (Attachment1); USP5745569 (Attachment2); and USP9104842 for the support cited by the reexam request in the Cooperman Reference</p>
<p>Claim 11</p>	
<p>11. A method for licensed software use, the method comprising:</p>	<p>Reexam request page 82 cites Cooperman passages "ask[ing] the user for personalization information, which include the license code" and "the user must have a key. The key, in turn, is a function of the license information for the copy of the software in question." This appears at APP 08587943 page 11:25-26 and 12:15; USP5745569 col 6:23-24 and 6:49-50; USP9104842 col.13:59-60 and 14:18-19.</p>

Attachment 10

<p>loading a software product on a computer, said computer comprising a processor, memory, an input, and an output, so that said computer is programmed to execute said software product;</p>	<p>Reexam request page 82 cites Cooperman passage "[a] computer application seeks to provide a user with certain utilities or tools, that is, users interact with a computer or similar device to accomplish various tasks and applications provide the relevant interface." This appears at APP 08587943 page 3:16-19; USP5745569 col 2:7-10; USP9104842 col.2:19-15.</p> <p>Reexam request page 82 cites Cooperman passage "computer memory for the purpose of execution." which is part of claim 5.</p> <p>Reexam request page 83 cites Cooperman passage "order in the computer memory is of vital importance." This appears at APP 08587943 page 7:4-5; USP5745569 col 4:4-5; USP9104842 col.11:42-42.</p> <p>Reexam request page 83 cites Cooperman passage "process[] a digital sample stream for the purpose of modifying it or playing the digital sample stream." This appears at APP 08587943 page 17:4-5, as part of claim 5. USP5745569 does not include that claim but does refer to the watermark bearing content stream, col. 6:61-62; same recitation at USP9104842 col. 14:29-30.</p>
<p>said software product outputting a prompt for input of license information;</p>	<p>Reexam request page 83 cites Cooperman passage "1) when it is run for the first time, after installation, it asks the user for personalization information, which includes the license code. This can include a particular computer configuration." and "when prompted at start-up"; This appears at APP 08587943 page 11: 25-29 and page 1:28 to page 2:1; USP5745569 col. 6:23-26 and 1:27-28; USP9104842 col. 13:59-63 and 1:43-44.</p>

Attachment 10

<p>and said software product using license information entered via said input in response to said prompt in a routine designed to decode a first license code encoded in said software product.</p>	<p>Reexam request page 85 cites Cooperman passage "Given that there are one or more of these essential resources, ... Method and Device" patent application." This appears at APP 08587943 page 9:22-27; USP5745569 col. 5:19-24; USP9104842 col. 12:55-59.</p>
<p>Claim 12</p>	
<p>12. A method for encoding software code using a computer having a processor and memory, comprising:</p>	<p>Reexam request page 86 cites Cooperman passage "choose one or several essential code resources, and encode them into one or several data resources using the stegacipher process." This appears at APP 08587943 page 10:13-18; USP5745569 col. 5:19-24; USP9104842 col. 13:14-17.</p>
<p>storing a software code in said memory;</p>	<p>Reexam request page 86 argues that Cooperman describes techniques for randomizing the location of software code stored in memory. Randomization appears at APP 08587943 page 13:23-30; USP5745569 col. 7:22-29; USP9104842 col. 14:57-64.</p>
<p>wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system; and</p>	<p>Reexam request cites Cooperman passage "The basic premise for this scheme is that there are a certain sub-set of executable code resources, that comprise an application and that are 'essential' to the proper function of the application." and "The memory address of the first instruction ... as 'resources'". This appears at APP 08587943 page 8:10 et seq and page 9:9 et seq; USP5745569 col. 4:60 et seq and 4:18 et seq.; USP9104842 col. 12:29 11:55 et seq.</p>

Attachment 10

<p>encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key encoded software code; and wherein, when installed on a computer system, said first license key encoded software code will provide said specified underlying functionality only after receipt of said first license key.</p>	<p>Reexam request cites Cooperman passages "[t]he assembly utility can be supplied with a key generated from a license code generated for the license in question" and "The utility will choose one or several essential code resources, and encode them into one or several data resources using the stegacipher process." These appear at APP 08587943 page 11:9 et seq and 10:13 et seq; USP5745569 col. 4:8 et seq and 5:45 et seq; USP9104842 col. 13:44 et seq. and 13:13 et seq.</p>
<p>Claim 13</p>	
<p>13. A method for encoding software code using a computer having a processor and memory, comprising:</p>	<p>See claim 12</p>
<p>storing a software code in said memory;</p>	<p>See claim 12.</p>
<p>wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system;</p>	<p>See claim 12</p>
<p>and modifying, by said computer, using a first license key and an encoding algorithm, said software code, to form a modified software code; and wherein said modifying comprises encoding said first code resource to form an encoded first code resource;</p>	<p>The reexam request pages 93-49 cites Cooperman passages "The first method of the present invention described involves hiding necessary ... patent application." and "watermarking with 'keys' derived from license codes ... and using the watermarks encoded with such keys to hide an essential subset for the application code resources." These appear at APP 08587943 page 8:25 et seq and 5:19 et seq; USP5745569 col. 4:56 et seq and 3:14 et seq; USP9104842 col. 12:25 et seq. and 3:28 et seq.</p>

Attachment 10

<p>wherein said modified software code comprises said encoded first code resource, and a decode resource for decoding said encoded first code resource;</p>	<p>The reexam request page 95 cites Cooperman passages "Note further that the application contains a code resource which performs the function of decoding an encoded code resource from a data resource." and "Once [the application] has the license code, it can then generate the proper decoding key to access the essential code resources." These appear at APP 08587943 page 11:17 et seq and 11:131 et seq; USP5745569 col. 6:15 et seq and 6:29 et seq; USP9104842 col. 13:50 et seq. and 13:65 et seq.</p>
<p>wherein said decode resource is configured to decode said encoded first code resource upon receipt of said first license key.</p>	<p>The reexam request page 95 cites Cooperman passages "The application must also contain a data resource...." This appears at APP 08587943 page 11:20 et seq; USP5745569 col. 6:18 et seq; USP9104842 col. 13:54 et seq.</p>
<p></p>	<p></p>
<p>Claim 14</p>	<p></p>
<p>14. A method for encoding software code using a computer having a processor and memory, comprising:</p>	<p>See claim 13.</p>
<p>storing a software code in said memory;</p>	<p>See claim 13.</p>
<p>wherein said software code defines software code interrelationships between code resources that result in a specified underlying functionality when installed on a computer system; and</p>	<p>The reexam request page 97-98 cites Cooperman passages "Under the present invention, the application contains a special code resource..." and "The memory address of the first instruction ..." and "Once the code resources of a program are loaded into memory...." These appear at APP 08587943 page 14:35 et seq; page 7:14 et seq; page 13:31 et seq; USP5745569 col. 8:1 et seq; 4:18 et seq; 7:30 et seq; USP9104842 col. 15:36 et seq; 11:55 et seq; 14:65 et seq.</p>

Attachment 10

<p>encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key encoded software code</p>	<p>See claim 12.</p>
<p>in which at least one of said software code interrelationships are encoded.</p>	<p>The reexam request page 102 cites Cooperman passages "The application must also contain a data resource..."; "Under the present invention, the application contains..."</p> <p>"The application must also contain a data resource...." appears at APP 08587943 page 11:20 et seq; USP5745569 col. 6:18 et seq; USP9104842 col. 13:54 et seq.</p> <p>Under the present invention appear at APP 08587943 page 14:35 et seq; USP5745569 col. 8:1 et seq; USP9104842 col. 15:36 et seq.</p>

/RichardNeifeld/
 Richard Neifeld, Reg. No. 35,299
 Attorney for patentee

Y:\Clients\SCOT Scott A Moskowitz and Wistaria Trading, Inc\90014138, USP9104842, SCOT0014-4\Drafts\Attachment10, 9104842 claim support chart; support in 08587943.wpd

Reexamination Control Number: 90014138
Confirmation No: 6880
RE: US Patent 9104842

37 CFR 1.234 Certificate of Service

I served by first class mail (priority mail) to the third party reexamination requestor's correspondence address:

Attn: Joseph P. Edell
FISCH SIGLER LLP
5301 WISCONSIN AVENUE, NW
FOURTH FLOOR
WASHINGTON, DC 20015

the following documents:

Certificate of Service (1 page), showing service by first claims mail (express mail) of:
This Patent Owner Statement (22 pages)
Attachment 1, Pages 1-28 from the file history of application 08/587,793 (issued as USP5745569) (27 Pages)
Attachment 2, USP5745569 (6 pages)
Attachment 3, Pages 1-12 and 172-192 of the transcript of the deposition of Marc S. Cooperman, May 17, 2018, in Blue Spike LLC v. Juniper Networks, Inc., civil case 6:17-cv-00016-KNM. (33 Pages)
Attachment 4, Comparison of Disclosures of US application 08/587,943; WO 97/26732; USP5,745,569; USP9021602; and USP9104842. (5 pages)
Attachment 7, 2002 Settlement Agreement (37 pages)
Attachment 9, Title Abstract for application, 08587943, now USP5745569. (2 Pages)
Attachment 10, a claim support chart showing support in 08587943 filed 1/17/1996; USP5745569; and USP9104842 for the citations used by the reexam request to show disclosure of the claimed subject matter by the Cooperman reference. (6 pages)

Date of Service: 8/17/2018

/RichardNeifeld/
Richard Neifeld, Reg. No. 35,299
Attorney for patent owner

Electronic Acknowledgement Receipt

EFS ID:	33648036
Application Number:	90014138
International Application Number:	
Confirmation Number:	7638
Title of Invention:	DATA PROTECTION METHOD AND DEVICE
First Named Inventor/Applicant Name:	9104842
Customer Number:	31518
Filer:	Richard A. Neifeld
Filer Authorized By:	
Attorney Docket Number:	
Receipt Date:	06-SEP-2018
Filing Date:	16-MAY-2018
Time Stamp:	16:17:49
Application Type:	Reexam (Third Party)

Payment information:

Submitted with Payment	no
------------------------	----

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)/ Message Digest	Multi Part /.zip	Pages (if appl.)
1	Miscellaneous Incoming Letter	PatentOwnerStatement.pdf	287651 <small>5695e8363654da30cb62c8f75789e3b031c8a4e5</small>	no	22

Warnings:

Information:					
2	Miscellaneous Incoming Letter	Attachment1_Image_Spec.pdf	5264508	no	27
			353a91c888d4e6cf7a7001b1886a03be4ff9b28		
Warnings:					
Information:					
3	Miscellaneous Incoming Letter	Attachment2_Image_5745569_Moskowitz_Cooperman_Dice.pdf	2457818	no	6
			9ed9ed381b7d124190b5c355b9ca0904fe087d9f		
Warnings:					
Information:					
4	Miscellaneous Incoming Letter	Attachment3_Image_CoopermanDepositionExcerpt_051718.pdf	5853837	no	33
			b5d602542ce0f2ae549925f60660a11e8baa8d4e		
Warnings:					
Information:					
5	Miscellaneous Incoming Letter	Attachment4_ComparisonOfDisclosures_RN_8-14-2018.pdf	614220	no	5
			ac751bf85fc1ccc28d540137865841c30d2fda43		
Warnings:					
Information:					
6	Miscellaneous Incoming Letter	Attachment7_2002CoopermanSettlementAgreement.pdf	1875167	no	37
			5c1dbc71a273ede4098cc41ca80b73761c1e0649		
Warnings:					
Information:					
7	Miscellaneous Incoming Letter	Attachment9_Image_TitleAbstract_08587943_USP5745569.pdf	417106	no	2
			e43029454370f2a173b90a30bd795f1d759597c		
Warnings:					
Information:					
8	Miscellaneous Incoming Letter	Attachment10-9104842claimsupportchart_supportin08587943.pdf	85774	no	6
			91a8ab1e95cf06af526001f5a10afc2648eefab7		
Warnings:					
Information:					

9	Miscellaneous Incoming Letter	CertificateOfService_90014138 _IMAGE.pdf	305365 150372d6e1e56848e59c4b8b899cf1e9acfa 8615	no	1
Warnings:					
Information:					
Total Files Size (in bytes):				17161446	
<p>This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.</p> <p><u>New Applications Under 35 U.S.C. 111</u> If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.</p> <p><u>National Stage of an International Application under 35 U.S.C. 371</u> If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.</p> <p><u>New International Application Filed with the USPTO as a Receiving Office</u> If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.</p>					



8-21-18

Reexam

Reexamination Control Number: 90014138
Confirmation No: 6880
RE: US Patent 9104842

Filing

37 CFR 1.234 Certificate of Service

I served by first class mail (priority mail) to the third party reexamination requestor's correspondence address:

Attn: Joseph P. Edell
FISCH SIGLER LLP
5301 WISCONSIN AVENUE, NW
FOURTH FLOOR
WASHINGTON, DC 20015

the following documents:

- Certificate of Service (1 page), showing service by first class mail (express mail) of:
- This Patent Owner Statement (22 pages)
- Attachment 1, Pages 1-28 from the file history of application 08/587,793 (issued as USP5745569) (27 Pages)
- Attachment 2, USP5745569 (6 pages)
- Attachment 3, Pages 1-12 and 172-192 of the transcript of the deposition of Marc S. Cooperman, May 17, 2018, in Blue Spike LLC v. Juniper Networks, Inc., civil case 6:17-cv-00016-KNM. (33 Pages)
- Attachment 4, Comparison of Disclosures of US application 08/587,943; WO 97/26732; USP5,745,569; USP9021602; and USP9104842. (5 pages)
- Attachment 7, 2002 Settlement Agreement (37 pages)
- Attachment 9, Title Abstract for application, 08587943, now USP5745569. (2 Pages)
- Attachment 10, a claim support chart showing support in 08587943 filed 1/17/1996; USP5745569; and USP9104842 for the citations used by the reexam request to show disclosure of the claimed subject matter by the Cooperman reference. (6 pages)

Date of Service: 8/17/2018

/RichardNeifeld/
Richard Neifeld, Reg. No. 35,299
Attorney for patent owner



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
90/014,138	05/16/2018	9104842		7638

31518 7590 12/12/2018
NEIFELD IP LAW, PC
5400 Shawnee Road
Suite 310
ALEXANDRIA, VA 22312-2300

EXAMINER

BONSHOCK, DENNIS G

ART UNIT	PAPER NUMBER
----------	--------------

3992

MAIL DATE	DELIVERY MODE
-----------	---------------

12/12/2018

PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.



UNITED STATES PATENT AND TRADEMARK OFFICE

Commissioner for Patents
United States Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450
www.uspto.gov

DO NOT USE IN PALM PRINTER

(THIRD PARTY REQUESTER'S CORRESPONDENCE ADDRESS)

Joseph F. Edell
Fisch Sigler LLP
5301 Wisconsin Ave, NW
Fourth Floor
Washington, DC 20015

***EX PARTE* REEXAMINATION COMMUNICATION TRANSMITTAL FORM**

REEXAMINATION CONTROL NO. 90/014,138 .

PATENT UNDER REEXAMINATION 9104842 .

ART UNIT 3992 .

Enclosed is a copy of the latest communication from the United States Patent and Trademark Office in the above identified *ex parte* reexamination proceeding (37 CFR 1.550(f)).

Where this copy is supplied after the reply by requester, 37 CFR 1.535, or the time for filing a reply has passed, no submission on behalf of the *ex parte* reexamination requester will be acknowledged or considered (37 CFR 1.550(g)).

Office Action in Ex Parte Reexamination	Control No. 90/014,138	Patent Under Reexamination 9104842	
	Examiner DENNIS G BONSHOCK	Art Unit 3992	AIA Status No

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

- a. Responsive to the communication(s) filed on 17 August 2018.
 A declaration(s)/affidavit(s) under **37 CFR 1.130(b)** was/were filed on _____.

b. This action is made FINAL.

c. A statement under 37 CFR 1.530 has not been received from the patent owner.

A shortened statutory period for response to this action is set to expire 2 month(s) from the mailing date of this letter. Failure to respond within the period for response will result in termination of the proceeding and issuance of an *ex parte* reexamination certificate in accordance with this action. 37 CFR 1.550(d). **EXTENSIONS OF TIME ARE GOVERNED BY 37 CFR 1.550(c)**. If the period for response specified above is less than thirty (30) days, a response within the statutory minimum of thirty (30) days will be considered timely.

Part I THE FOLLOWING ATTACHMENT(S) ARE PART OF THIS ACTION:

- | | |
|--|---|
| 1. <input type="checkbox"/> Notice of References Cited by Examiner, PTO-892. | 3. <input type="checkbox"/> Interview Summary, PTO-474. |
| 2. <input type="checkbox"/> Information Disclosure Statement, PTO/SB/08. | 4. <input type="checkbox"/> _____. |

Part II SUMMARY OF ACTION

- 1a. Claims 11-14 are subject to reexamination.
- 1b. Claims 1-10 are not subject to reexamination.
2. Claims _____ have been canceled in the present reexamination proceeding.
3. Claims _____ are patentable and/or confirmed.
4. Claims 11-14 are rejected.
5. Claims _____ are objected to.
6. The drawings, filed on _____ are acceptable.
7. The proposed drawing correction, filed on _____ has been (7a) approved (7b) disapproved.
8. Acknowledgment is made of the priority claim under 35 U.S.C. 119(a)-(d) or (f).
 - a) All b) Some* c) None of the certified copies have
 - 1 been received.
 - 2 not been received.
 - 3 been filed in Application No. _____.
 - 4 been filed in reexamination Control No. _____.
 - 5 been received by the International Bureau in PCT application No. _____.
- * See the attached detailed Office action for a list of the certified copies not received.
9. Since the proceeding appears to be in condition for issuance of an *ex parte* reexamination certificate except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte* Quayle, 1935 C.D. 11, 453 O.G. 213.
10. Other: _____

cc: Requester (if third party requester)

NON-FINAL OFFICE ACTION

***ex parte* Reexamination**

This is an *ex parte* reexamination of U.S. Patent Number: 9,104,842 issued to Moskowitz, hereinafter the '842 Patent. This action addresses patent claims 11-14 for which it has been determined in the Order mailed 6/19/2018 that a substantial new question of patentability was raised in the Request for *ex parte* reexamination filed 5/16/2018. The Patent Owner's Statement filed on 8/17/2018 is discussed and refuted below.

The present application is being examined under the pre-AIA first to invent provisions.

Availability of References as Prior Art:

Claims 11-14 are reexamined on the basis of the following references:

U.S. Patent No. 5,933,497 issued to Beetcher (hereinafter Beetcher / Ex.3)

Japanese Patent Application Publication No. H05334072 issued to Beetcher (hereinafter Beetcher '072 / Ex.4)

PCT Application Publication No. WO 97/26732 issued to Cooperman (hereinafter Cooperman / Ex. 6)

U.S. Patent No. 5,935,243 issued to Hasebe (hereinafter Hasebe / Ex. 7)

Prosecution History

U.S. Patent Application Serial No. 11/895,388, which resulted in issued Patent 9,104,842 (hereinafter the '842 patent), was filed on August 24, 2007.

During prosecution of the '842 Patent, claims 1-64 were rejected under 35 U.S.C. § 103(a) as being unpatentable over Holmes et al. (US Patent Number: 5,287,407) in further view of Houser et al. (US Patent Number: 5,606,609).

Claims 32-45 and 52-64 were remaining in the case and rejected over Holmes and Houser (supra), when the claims went to appeal.

The Examiners Answer, dated 8/8/2012, subsequently withdrew the rejections of claims 34, 45, 54, and 58, with claims 34, 45, and 54 left as objected to as being dependent upon a rejected base claim, and claim 58 (now patent claim 11) noted as 'recites allowable subject matter', with no further elaboration.

In the Decision on Appeal, dated 3/12/2015, the board considered the rejections under 35 U.S.C. § 103(a) (supra) and rendered the judgment that:

DECISION

The rejection of claims 32, 33, 35, 37, 39, 52, 53, 55-57, 59, and 63-64 under 35 U.S.C. § 103(a) is affirmed.

The rejection of claims 36, 38, 40-44, and 60-62 under 35 U.S.C. § 103(a) is reversed.

The Board in the Decision specifically noted, with respect to the reversed claims, that:

Claims 36 and 60 (patent claim 12)

Claims 36 and 60 include limitations that **require the underlying software functionality be enabled upon the presence or detection of a key, or other software code**. See, e.g., Claim 60 (“software code will provide said specified underlying functionality only after receipt of said first license key”). Appellant argues neither Holmes nor Houser teaches or suggests enabling software functionality based on a license key. App. Br. 80, 98. We agree. Holmes states the data block containing the identification information “does not play any part in the function of the software of the master file itself.” Holmes, col. 3, ll. 41-42. Accordingly, we cannot sustain the rejection of claims 36 and 60.

Claim 61 (patent claim 13)

Appellant contends the Examiner’s rejection of claim 61 does not address various limitations of the claim, such as “**encoding said first code resource to form an encoded first code resource,**” or an “**encoded first code resource, and a decode resource for decoding said encoded first code resource**” in the software. App. Br. 99-100. The Examiner relies on reasoning found in the rejections of claims 32 and 43. Final Act. 7. The Examiner’s findings do not support the combination of Houser and Holmes teaches or suggests a modified software code comprising an encoded first code resource and a decode resource for decoding the encoded first code resource, wherein the decode resource is configured to decode the encoded first code resource upon receipt of a first license key. Accordingly, we do not sustain the rejection of claim 61.

Claim 62 (patent claim 14)

Appellant argues “neither Holmes nor Houser disclose or suggest encoding code interrelationships between code resources of the software.” App. Br. 103-04. The Examiner bases the rejection of claim 62 on the reasons set forth in rejecting claims 32 and 61. Final Act. 8. We disagree the same reasons apply. For example, claims 32 and 61 do not recite limitations regarding **“software code interrelationships between code resources that result in a specified underlying functionality.”** Because the Examiner has not shown how the references teach or suggest all the limitations of claim 62, we do not sustain its rejection.

On 6/4/2015, the Examiner issued a Notice of Allowance based on the Board’s March 12, 2015 decision. After the notice of allowance. Patent Owner requested claim amendments, adding, in pertinent part, the term “product” to claim 58. The patent issued on August 11, 2015.

REJECTIONS

The rejections below are confined to what has been deemed to be the best available art from the Request. However, prior to conclusion of this reexamination proceeding, claims must be patentable over all prior art cited in the order granting reexamination in order to be considered patentable or confirmed on the reexamination certificate. The references cited in the request but not utilized in the current office action appear to be largely cumulative to the teachings in the reference applied below.

Claim Rejections – 35 USC § 102 and § 103

The following is a quotation of the appropriate paragraphs of pre-AIA 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

A person shall be entitled to a patent unless –

(a) the invention was known or used by others in this country, or patented or described in a printed publication in this or a foreign country, before the invention thereof by the applicant for a patent.

(e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

The following is a quotation of pre-AIA 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negated by the manner in which the invention was made.

Beetcher Reference

Claims 11, 12, 13, and 14 are anticipated by Beetcher under 35 U.S.C. § 102(a).

(The Beetcher '072 reference is equally applicable, however is not relied upon here for the sake of brevity, please see third party requestors mapping for correspondence on pages 50-81 of the Request)

In summary:

Beetcher teaches utilizing components of the product key (version # / product #) when encoding software code via a templating algorithm (see column 9, lines 1-20 and column 6, lines 22-40). The resultant executable code including referencable triggers inserted therein executable via checks to interrelated entries in the product lock table (see column 4, lines 14-23). Where the software module is said to be distributed as compiled object code with embedded entitlement verification triggering instructions that contain version # and product # fields (see column 6, lines 41-58). Thereinafter trigger enabled software codes segments are only executable if an entitlement key is provided that enables the corresponding trigger / flag (see column 4, lines 14-23 and column 9, line 49 through column 10, line 47).

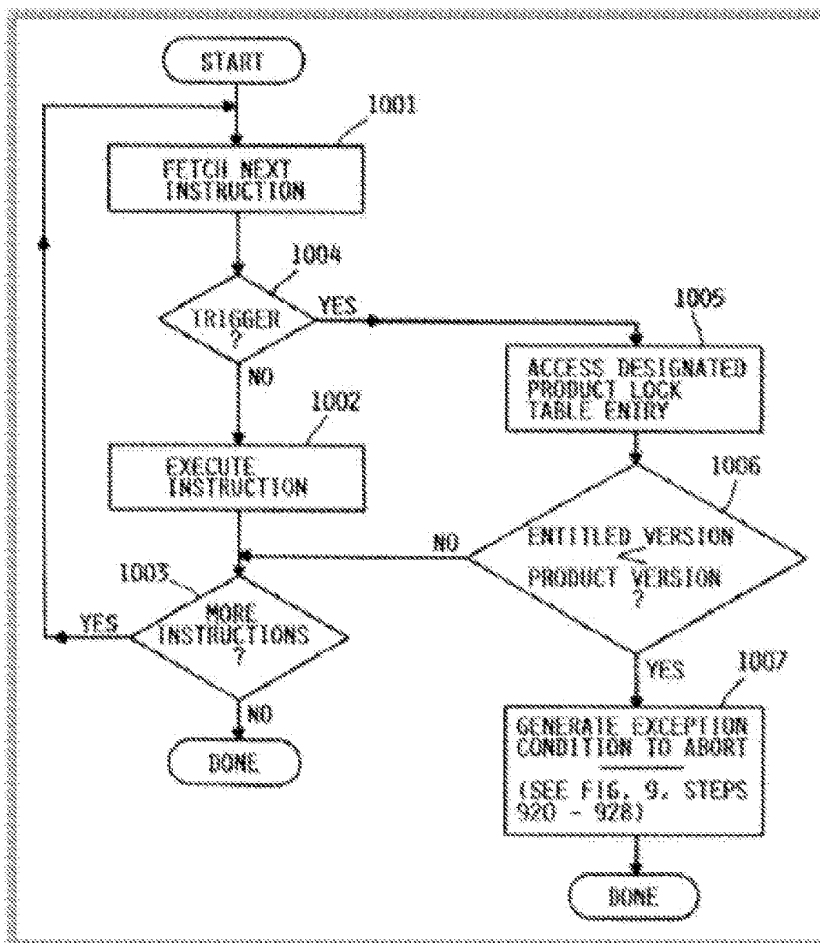
Claim 11

a) Preamble: “A method for licensed software use, the method comprising”

Under the broadest reasonable construction, the preamble is non-limiting. Nevertheless, Beetcher discloses claim 11'S preamble. Specifically, Beetcher describes a method of controlling access to licensed software using an encrypted entitlement key. (Beetcher at Abstract, 4:3-13, 4:39-44, 10:48-11:3; see also id. at 1:7-11, 1:54-57, 3:54-62) Beetcher, for instance, summarizes its invention as:

Software is distributed according to the present invention without entitlement to run. A separately distributed encrypted entitlement key enables execution of the Software, The key includes the serial number of the machine for which the Software is licensed, together with a plurality of entitlement bits indicating which Software modules are entitled to run on the machine, (Beetehar at 4:3-9)

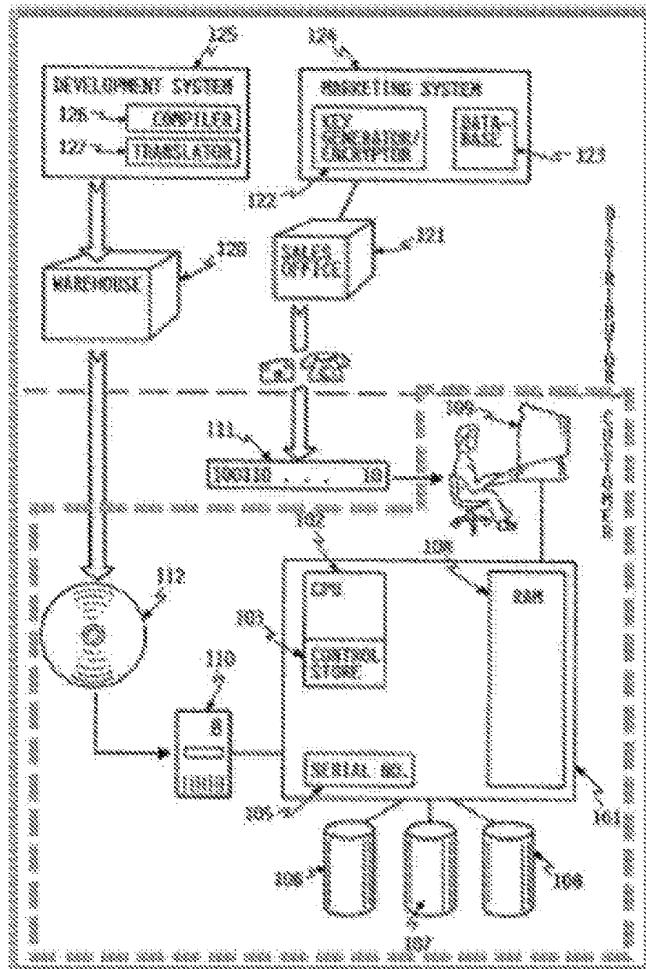
Beetcher's Figure 10, as provided below, illustrates the use of an entitled version of software based on the customer's license:



As such, Beetcher teaches this preamble (Silva Declaration at 35-36).

b) Element 11.1: “loading a software product on a computer, said computer comprising a processor, memory, an input, and an output, so that said computer is programmed to execute said software product”

Beetcher discloses element 11.1. Specifically, Beetcher’s system includes a customer computer 101 including a CPU 102, memory 104, and storage devices 106-108. (Beetcher at 5:14-21, Fig. 1) This customer computer 101 also includes a media reader 110 (i.e., an input) and an operator console 109 (i.e., an output). (Id. at 5:25-32, 6:7-15, Fig. 1) As shown below in annotated Figure 1, Beetcher discloses a computer having software product 112 loaded for execution (dashed perimeter) (Silva Declaration at ¶ 38-40):



Beetcher details that the customer loads the media, such as an optical disk, containing a software product onto the computer to execute the software product:

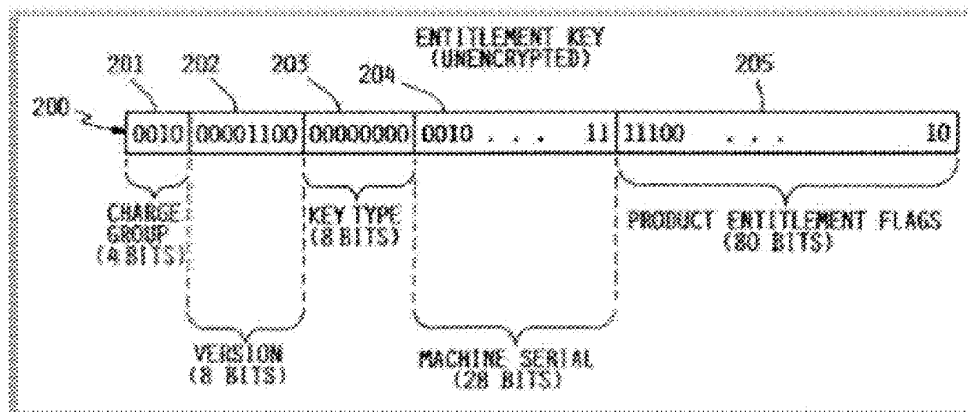
[S]oftware media 112 comprise one or more optical read/only disks, and unit 110 is an optical disk reader, it being understood that electronic distribution or other distribution media could be used. Upon receipt of software media 11, the customer will typically load the desired software modules from unit 110 into system 101, and store the software modules on storage devices 105-108. (Beetcher at 6:7-15\ see also id. at Abstract, 3:48-50, 9:51-55,: Fig. 1, claim 6)

c) Element 11.2: “said software product outputting a prompt for input of license information”

Beetcher discloses element 11.2. Specifically, Beetcher explains that its software product includes a user interface routine for the customer to input a license key into the computer before the product can be used. (Id. at 7:66-8:8; see also id at 3:25-28) For instance, Beetcher explains that the software product prompts the user to input license information:

This operation system support at virtual machine level 404 contains two user interface routines needed to support input of the entitlement key. General input routine 441 is used to handle input during normal operations. In addition, special install input routine 440 is required to input the key during initial installation of the operating system. This is required because that part of the operating system above machine interface level 405 is treated for purposes of this invention as any other program product; it will have a product number and its object code will be infected with entitlement verification triggers. (Id. at 7:66-8:8)

Beetoher's Figure 2 illustrates this license information in unencrypted form:

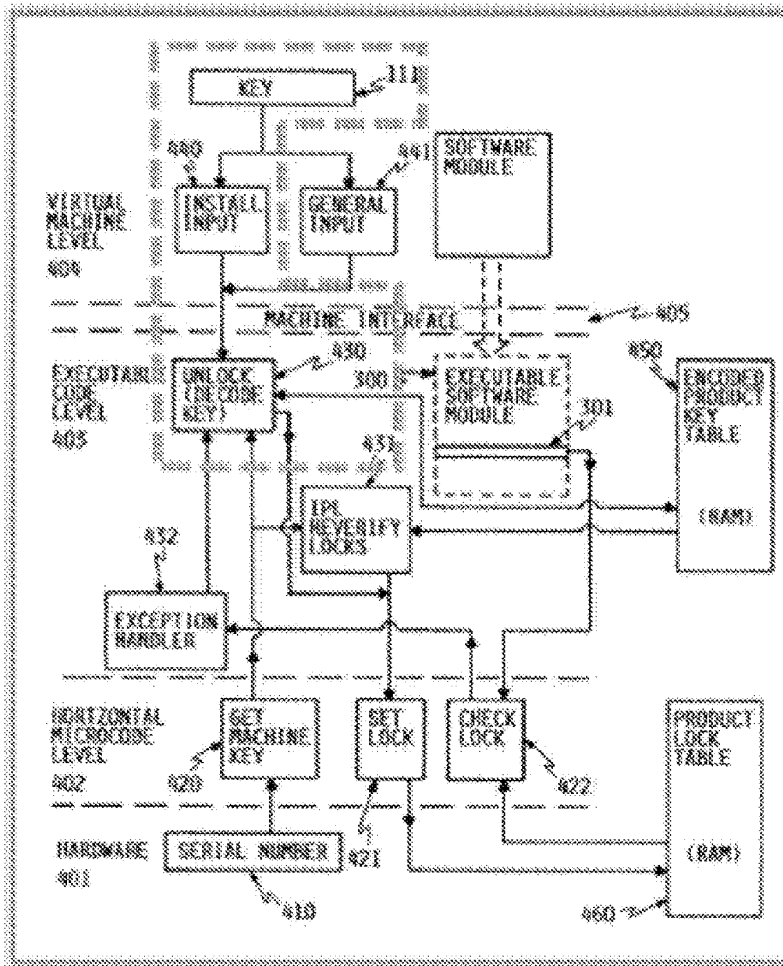


Beetcher further explains that the software's "install input routine 440 interacts with the operator to receive the input" of the customer's license information during the software's initial installation. (Id. at 9:51-55; me also id. at Fig. 4 (reference number 440), claim 6) And as discussed with respect to element 11.1, the customer's computer includes an operator console 109 shown with a monitor and keyboard that "can receive input from an operator." (Id. at 3:25-28, Fig. 1; Silva Declaration at ¶¶42-44)

d) Element 11.3: "said software product using license information entered via said input in response to said prompt in a routine designed to decode a first license code encoded in said software product"

Beetcher discloses element 11.3. Upon inserting the software's disk 112, Beetcher explains that the operator console prompts the customer to enter a license key. (E.g., Beetcher at 6:11-19, 7:66-8:8, Figs. 1, 9a) Beetcher details that the customer enters entitlement key 111, i.e., license information, in response to the prompt initiated by install input routine 440. (Id. at 7:66-8:8; see also id. at 9:51-55, Figs. 1, 4, claim 6) After entering that key, Beetcher teaches that the customer's computer uses a decode key to initiate unlock routine 430 to decode the license code encoded in the software product. (Id. at 7:39-42, 9:49-60; see also id. at 6:66-7:5, 8:60-62 Figs. 4, 9a) Beetcher's Figures 4 and 9a, which are provided below, show the software using the key (i.e., license information) entered by the customer to decode a first license code encoded in the software product. For instance, annotated Figure 4 illustrates that the install input routine 440 starts unlock routine 430 once the customer inputs key 111

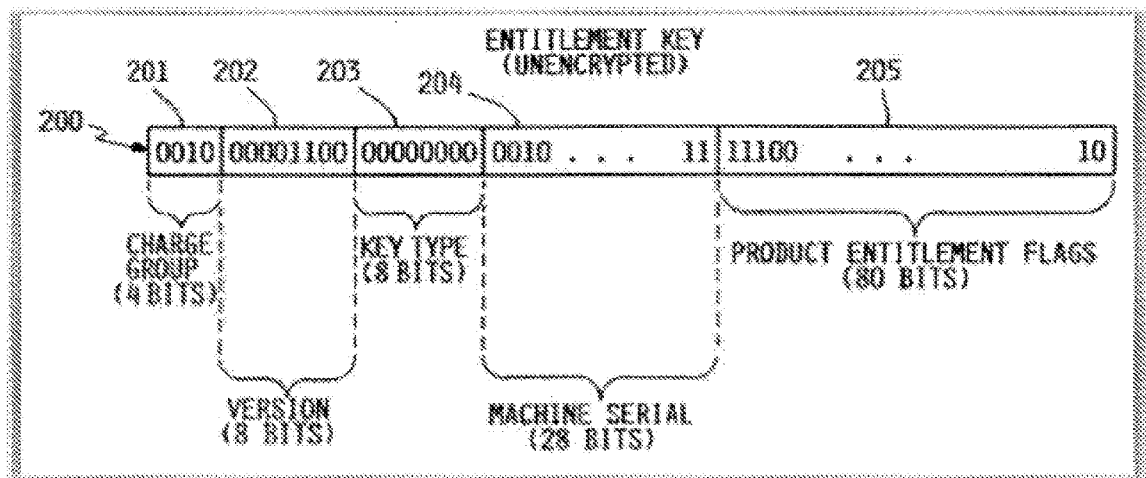
into the computer. (Id. at 8:3-13, 9:52-60) And “[u]nlock routine 430 uses the unique machine key to decode[] entitlement key 111” (dashed perimeter): (Id. at 7:39-42; see also id. at 8:62-62; 10:27-36)



Beetcher details that unlock routine 430 “handles the decoding process,” which is illustrated in Figure 9a’s steps 902-909: “Unlock routine 430 causes get machine key function 420 to retrieve the machine serial number and generate the machine key at

902. Unlock routine 430 then uses the machine key to decode the entitlement key 111 at step 903.” (Id. at 9:57-60)

Beetcher specifies that its unencrypted entitlement key includes multiple fields, which includes version field 202 specifying entitled version levels and product entitlement flags 205 specifying customer’s accessible product numbers. (Id at 6:22-40) Beetcher’s Figure 2 shows this license information with fields 201 to 205:

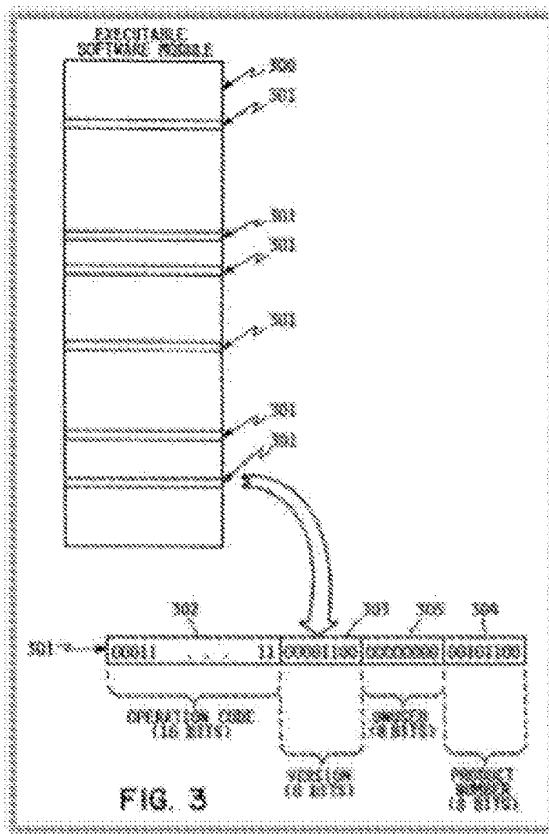


Beetcher’s unlock routine 430 will complete the decoding process by building an encoded product key table (step 904), populating the key table for the relevant software product Specified in the entitlement key (steps 905-908), and saving the key table (step 909). (Id. at 9:60-10:19, Figs. 5, 9a) Beetcher also specifies that the customer’s RAM includes table 460 populated with products having entitlement keys. (Id. at 7:42-44, 8:43-52, 10:20-47, Fig. 6, Fig. 9a) Beetcher’s software product uses the key’s version and product number fields to decode a license code.

When compiling and translating the software code, Beetcher explains that the code includes entitlement verification triggering instructions encoded into the software.

(Id. at 6:41 -58. 11:4-59; see also id. at 4:14-23, 8:5-22, 8:56-9:20)

Beetcher's triggering instructions are encoded into the software when the software code is compiled and translated, as shown in Figure 3 provided below:



Beetcher explains that its software code verifies the customer is entitled to use the software when the code encounters a triggering instruction. When it encounters one of these instructions, Beetcher's code accesses the license key information stored in the

key table 460. (Id. at 10:48—11:39; See also id., at Abstract, 8:14-22, 8:53-9:20, Fig. 10)

As such, a POSITA would have understood that Beetcher uses its license information in a routine, such as check lock function 422, designed to decode a first license code encoded in a software product via the triggering instructions:

If any instruction is an entitlement verification triggering instruction 301 (step 1004) check lock function 422 is invoked. Check lock function 422 accesses the product lock table entry 601 corresponding to the product number contained in the triggering instruction at step 1005. If the version number in product lock table 460 is equal to or greater than the version number 303 contained in triggering instruction 301, the software is entitled to execute (step 1006). (Id. at 10:52-62, Fig. 10; Silva Declaration at ¶ 46-51)

Moreover, Beetcher teaches that the triggering instructions will be encoded into the code resources controlling software functionality:

[An] additional barrier would be to define the entitlement triggering instruction to simultaneously perform some other function.... The alternative function must be so selected that any compiled software module will be reasonably certain of containing a number of instructions performing the function. If these criteria are met, the compiler can automatically generate the object code to perform the alternative function (and simultaneously, the entitlement verification trigger) as part of its normal compilation procedure. This definition would provide a significant barrier to patching of the object code to nullify the entitlement triggering instructions. (Beetcher at 11:14-28; see also id. at 4:25-33, 6:58-65)

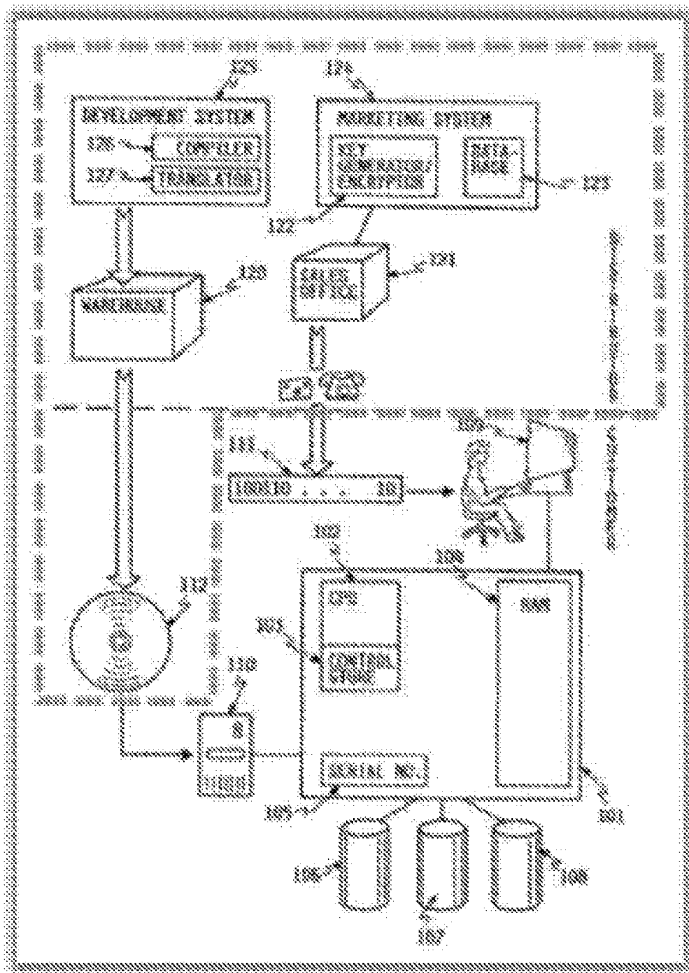
And Beetcher details that “the triggering instruction is also a direct instruction to perform some other useful work | Execution of the triggering instruction causes system 101 to perform some other operation simultaneous with the entitlement verification.” (Id. at 6:58-65 (Beetcher specifies that these functions are those “which do not require that an operand for the action be specified in the instruction.”); Silva Declaration at ¶ 52-53)

Accordingly, Beetcher discloses claim 11.

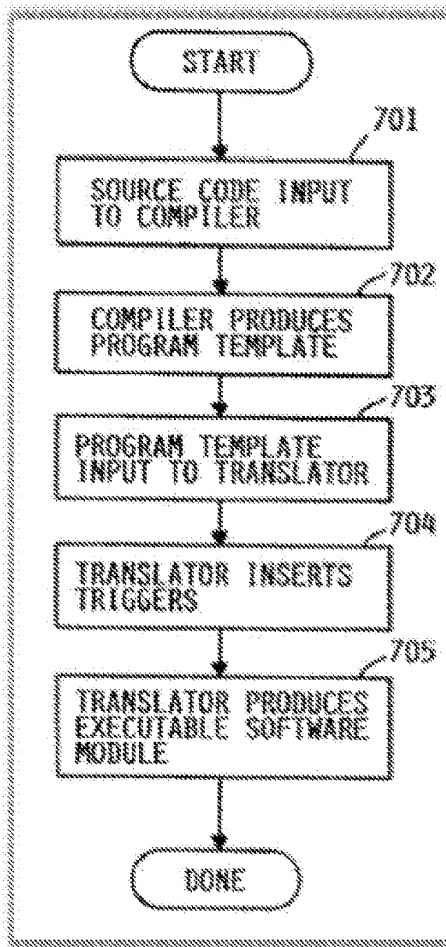
Claim 12

a) Preamble: “A method for encoding software code using a computer having a processor and memory, the method comprising”

Beetcher discloses claim 12’s preamble. Specifically, Beetcher describes a method for encoding software code using a computer with a processor and memory. Beetcher details that the software distributor has “development computer system 125, which contains compiler 126 and translator 127” where “[t]he software modules are recorded on software recording media 112” and “entitlement key generator/encrypter 122 and a database 123 containing customer information.” (see Beetcher at 5:38-48; see also *id.* at 9:1-20). Beetcher specifies these compiling and key generating functions may be performed by a single computer (*Id.* at 5:51-58). Below annotated Figure 1 illustrates the distributor’s computer system distributing memory media 112 and compiling encoded software code:



Beetcher's Figure 7 illustrates the software code being encoded to include watermarking triggers decoded by the customer's licensing information (Id. at 9:1-20, Fig. 7).



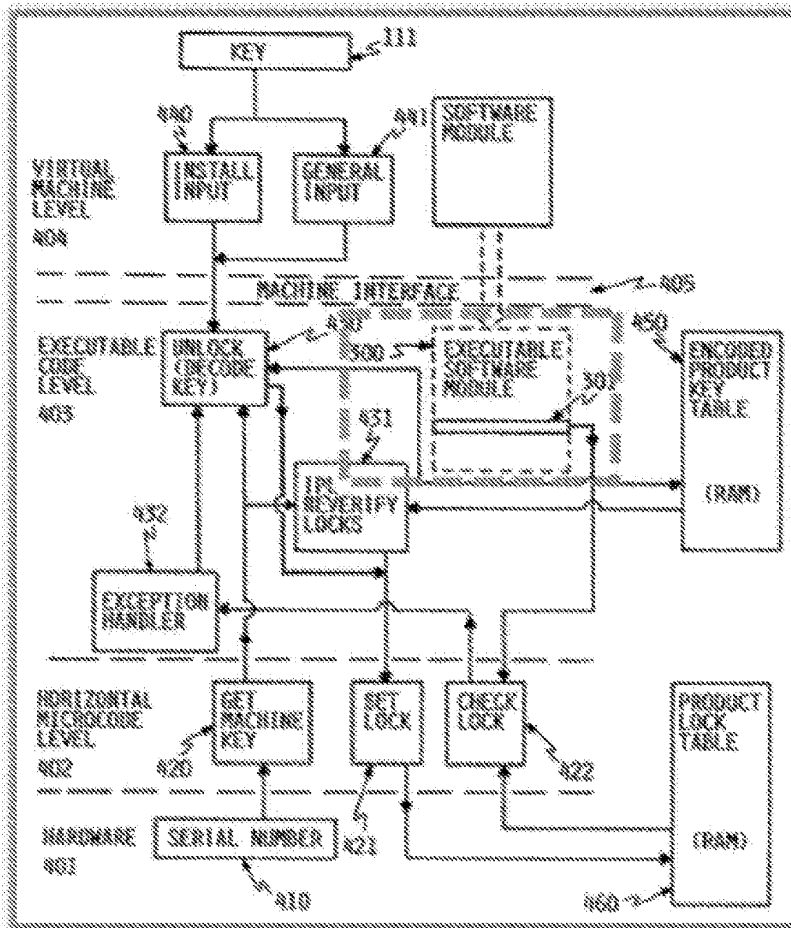
As such, a POSITA would have understood that Beetcher's distributor compiles and stores the encoded software code using a process or and memory akin to the console's CPU 102 and memory devices 106-108. As expert Dr, Silva explains in his declaration (Ex. 9), Beetcher's computer would necessarily include a processor and memory in order to function. (Silva Declaration at ¶¶156-59)

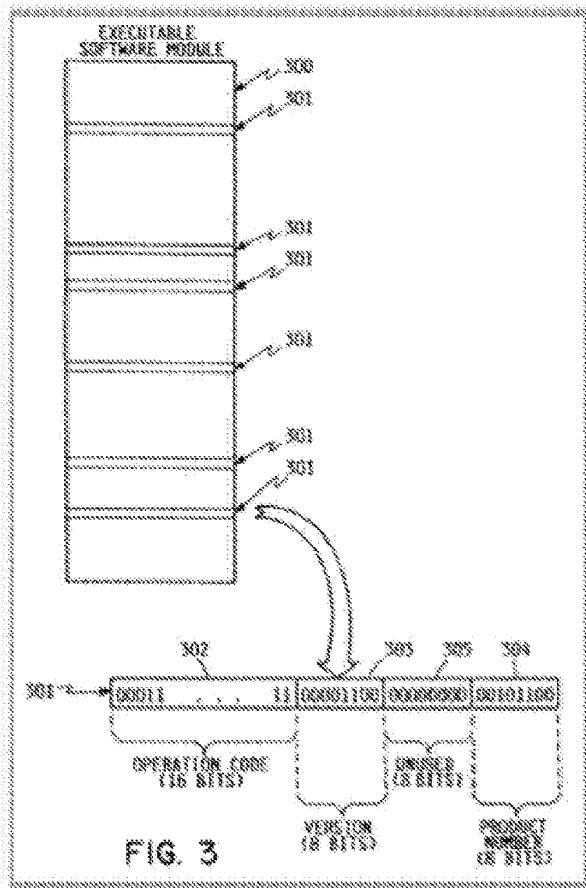
b) Element 12.1: "storing a software code in said memory"

Beetcher discloses element 12.1. Specifically, Beetcher discloses a development system 125 for compiling and translating for the software code (Beetcher at 5:38-48, 9:1-20). Beetcher details that the software code is stored as disks 112 in warehouse 120. A POSITA would have understood that developer system 125 stores the compiled and translated code in memory and records that code onto disks 112 for distribution to customers. As expert Dr. Silva explains in his declaration (Ex. 9), Beetcher's computer would necessarily include store software code in memory in order to function. (Silva Declaration at ¶ 62)

c) Element 12.2: “wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system”

Beetcher discloses element 12.2. Specifically, Beetcher explains that its software code includes multiple code resources that include a first code resource. (Beetcher at 5:40-43, 6:1-15) Beetcher's code resources include software modules 300 (dashed box) including sub-objects within the code, as shown below in annotated Figure 4 and Figure 3 (Id. at 6:41-45, 8:14-17, Fig. 4; see also id. at 7:45-48, Fig. 3). These sub-objects control multiple functions of the software installed on the customer's computer system 101 (Id. at 6:58-65, 11:4-39; see also id. at Abstract, 4:28-33, 6:65-7:5, claim 3). And Beetcher's software prevents unwanted “patching” of these sub-objects by including entitlement verification triggering instructions 301 (Id. at 4:25-33, 11:11-39; see also id. at Abstract, 3:14-18).





The '842 Patent refers to sub-objects and a memory scheduler as examples of code resources ('842 Patent at 11:55-65, 15:36-42). APOSITA would have understood that Beetcher's module sub-objects are sub-objects (Silva Declaration at 65-66).

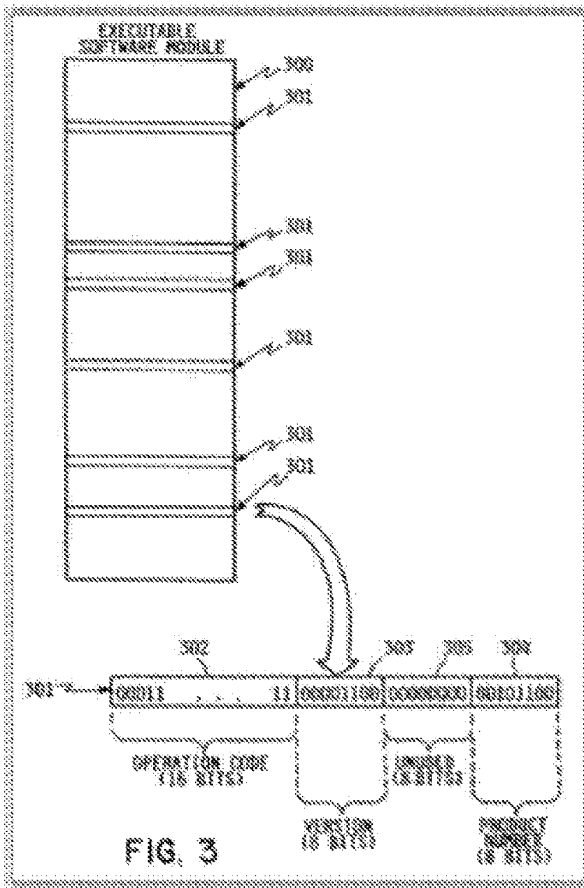
Based on Beetcher's description, a PGSITA would have understood that one sub-object in module 300 is a first code resource providing a specified underlying functionality when installed on the customer's computer system 101 and unlocked using the license information (key) (Id. at If 67).

d) Element 12.3: “encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key encoded software code”

Beetcher discloses element 12.3. Beetcher describes encoding its software code by the distributor system that includes development system 125 and marketing system 124, which may be “a single computer system performing both functions.”(Beetcher at 5:37-58, 6:41-65, 11:4-39) Specifically, Beetcher describes encoding a first license key into the software code where that key is used to authorize access to the software product:

Software module 300 is part of a program product in compiled object code form which executes on system 101.... [T]he actual executable code operates at executable code level 403, as shown by the box in broken lines. The executable code contains entitlement verification triggering instructions 301 (only one shown), which are executed by horizontal microcode check lock function 422. (Id. at 8:13-23; see also id. at 4:3-21, 6:20-55, 7:39-44, 8:58-67, 9:51-56, 10:22-38)

This encoding is illustrated in Figure 3:



The computer in Beetcher's development system 125 performs the encoding, as shown in Figure 7 at step 704, detailed as: "The program template serves as input to translator 127 at step 704, along with its product number and version number identification. Translator 127 automatically generates a substantial number of entitlement verification triggers, inserts them in random locations in the object code....." (Id. at 9:10-16; see also id< at 5:3'8-47, 9:1-10, 9:16-20, Fig. 7; Silva Declaration at 70-72)

Moreover, the computer in Beetcher's development system 125 uses an encoding algorithm to encode the first license key. Beetcher's system uses a set of instructions, as shown in Figure 7, to encode triggers into the software code to form the first license key : (Beetcher at 9:10-16: see also id. at 5 :38-47, 9:1-1Q, 9:16-20, Fig. 7; Silva Declaration at ¶ 73)

The compiler begins the process by producing a template (step 702), next the template is input into the translator (step 703), then the translator encodes the triggers/license keys into the code (step 704), and finally the translator resolves references after key insertion to produce the executable module (Beetcher at 9:6-20, Fig. 7). As such, a POSITA would have understood Beetcher's Figure 7 illustrates an encoding algorithm. (Silva Declaration at ¶74) Beetcher's encoding process is further described with respect to element 11.3.

Moreover, during the original prosecution, Patent Owner specified that "[e]ncoding using a key and an algorithm is known." (Ex. 2, Prosecution History at 519) As such, a POSITA would have understood that Beetcher's encoding technique necessarily includes a first license key and an encoding algorithm to form a first license key encoded software code (Silva Declaration at 70-75).

e) Element 12.4: “wherein, when installed on a computer system, said first license key encoded software code will provide said specified underlying functionality only after receipt of said first license key”

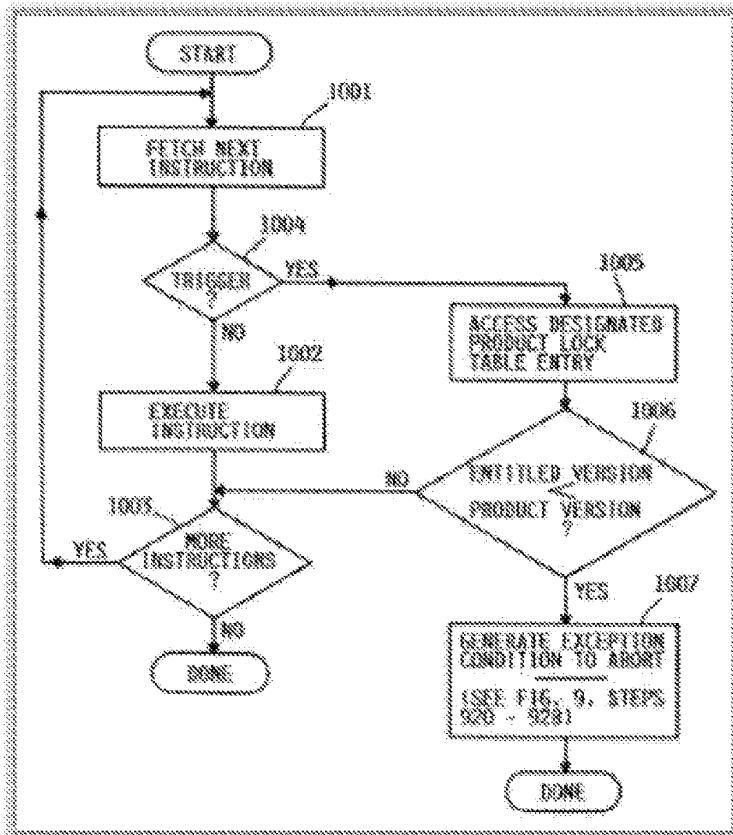
Beetcher discloses element 12.4. Specifically, Beetcher explains that its first license key encoded software code provides the specified underlying functionality only after receipt of the first license key. (Beetcher at 6:58-65, 11:4-39; see also id. at Abstract, 3:14-18, 4:25-33, 6:65-7:5, claim 3) For instance, Beetcher states:

For support of such a traditional compilation path where the object code format is known by customers, additional barriers to patching of the object code to nullify or alter the entitlement triggering instructions maybe appropriate. One such additional barrier would be to define the entitlement triggering instruction to simultaneously perform some other function. In this case, it is critical that the alternative function performed by the triggering instruction cannot be performed by any other simple instruction. The alternative function must be so selected that any compiled software module will be reasonably certain of containing a number of instructions performing the function. If these criteria are met, the compiler can automatically generate the object code to perform the alternative function (and simultaneously, the entitlement verification trigger) as part of its normal compilation procedure. This definition would provide a significant hairier to patching of the object code to nullify the entitlement triggering instructions. (Id. at 11:10-28)

And as described with respect to element 12.3, Beetcher teaches encoding the triggering instructions into the software code that is decoded via the first license key.

Beetcher’s Figure 10, as provided below, illustrates providing the software’s underlying functionality based on the first license key (trigger information). For instance, Beetcher explains:

System 101 executes the module by fetching (step 1001) and executing (step 1002) object code instructions until done (step 1003). If any instruction is an entitlement verification triggering instruction 301 (step 1004) check lock function 422 is invoked. Check lock function 422 accesses the product lock table entry 601 corresponding to the product number contained in the triggering instruction at step 1005. If the version number in product lock table 460 is equal to or greater than the version number 303 contained in triggering instruction 301, the software is entitled to execute (step 1006). (Id. at 10:49-60; see also id. at 10:48-49, 10:60-11:3; Silva Declaration at ¶ 78-82)



Claim 13

a) Preamble: “A method for encoding software code using a computer having a processor and memory, comprising”

As explained above, Beetcher discloses a method for encoding software using a computer with a processor and memory. As such, Beetcher teaches this preamble. (Silva Declaration at ¶ 85)

Beetcher details that the software distributor has “development computer system 125, which contains compiler 126 and translator 127” where “[t]he software modules are recorded on software recording media 112” and “entitlement key generator/encrypter 122 and a database 123 containing customer information.” (see Beetcher at 5:38-48; see also *id.* at 9:1-20). Beetcher specifies these compiling and key generating functions may be performed by a single computer (*Id.* at 5:51-58). Below annotated Figure 1 illustrates the distributor’s computer system distributing memory media 112 and compiling encoded software code:

declaration (Ex. 9), Beetcher's computer would necessarily include a processor and memory in order to function. (Silva Declaration at ¶¶156-59)

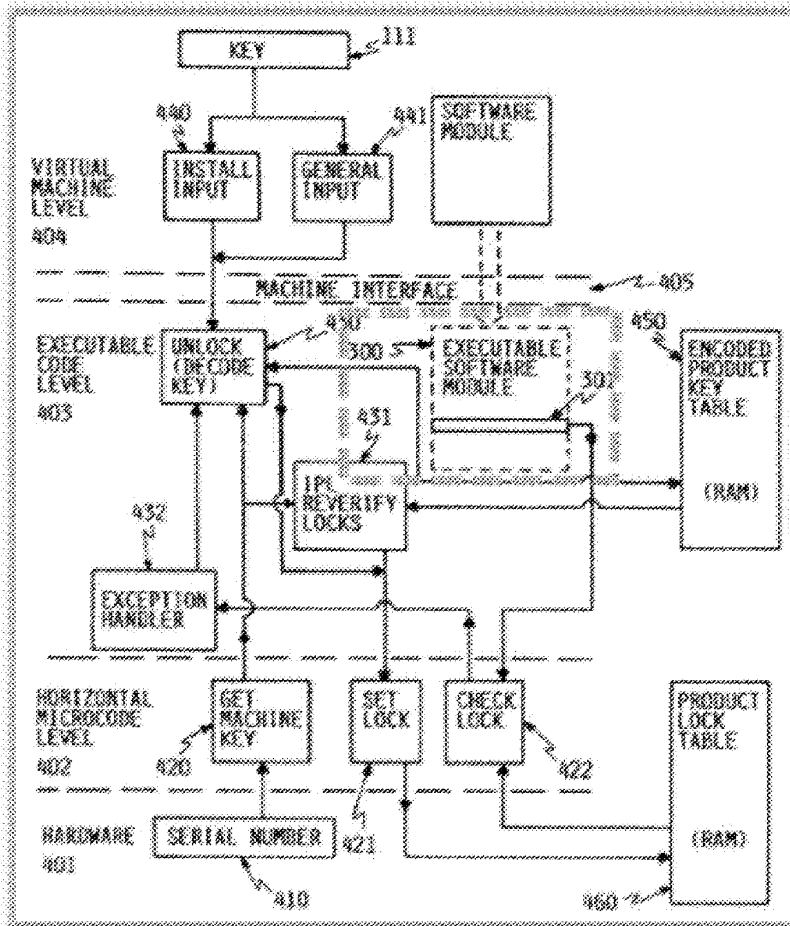
b) Element 13.1: "storing a software code in said memory"

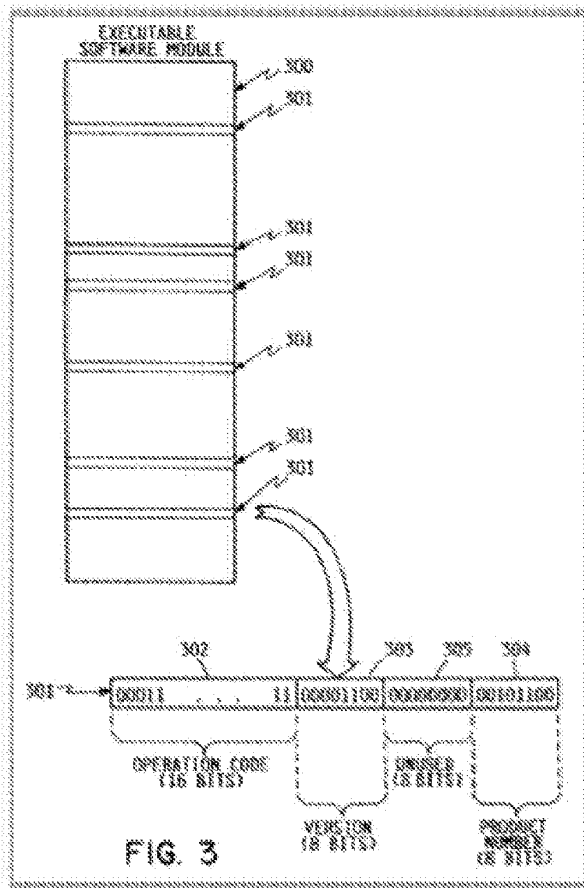
Beetcher discloses element 12.1. Specifically, Beetcher discloses a development system 125 for compiling and translating for the software code (Beetcher at 5:38-48, 9:1-20). Beetcher details that the software code is stored as disks 112 in warehouse 120. A POSITA would have understood that developer system 125 stores the compiled and translated code in memory and records that code onto disks 112 for distribution to customers. As expert Dr. Silva explains in his declaration (Ex. 9), Beetcher's computer would necessarily include store software code in memory in order to function. (Silva Declaration at ¶¶ 62 and 87)

c) Element 13.2: "wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system"

Beetcher discloses element 12.2. Specifically, Beetcher explains that its software code includes multiple code resources that include a first code resource. (Beetcher at 5:40-43, 6:1-15) Beetcher's code resources include software modules 300 (dashed box) including sub-objects within the code, as shown below in annotated Figure 4 and Figure 3 (Id. at 6:41-45, 8:14-17, Fig. 4; see also id. at 7:45-48, Fig. 3). These sub-objects control multiple functions of the software installed on the customer's computer system

101 (Id. at 6:58-65, 11:4-39; see also id. at Abstract, 4:28-33, 6:65-7:5, claim 3). And Beetcher's software prevents unwanted "patching" of these sub-objects by including entitlement verification triggering instructions 301 (Id. at 4:25-33, 11:11-39; see also id. at Abstract, 3:14-18).



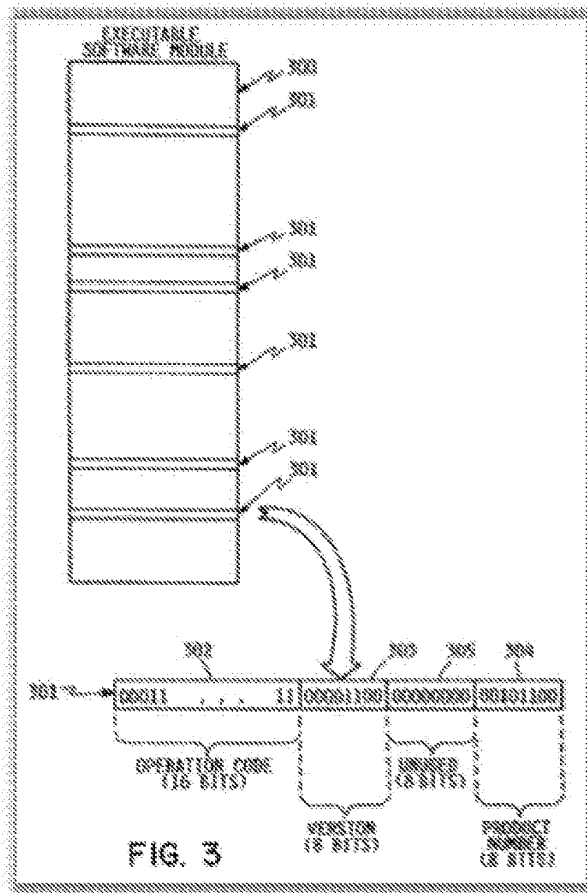


The '842 Patent refers to sub-objects and a memory scheduler as examples of code resources ('842 Patent at 11:55-65, 15:36-42). APOSITA would have understood that Beetcher's module sub-objects are sub-objects (Silva Declaration at ¶¶ 65-66).

Based on Beetcher's description, a PGSITA would have understood that one sub-object in module 300 is a first code resource providing a specified underlying functionality when installed on the customer's computer system 101 and unlocked using the license information (key) (Id. at ¶¶ 67 and 89).

d) Element 13.3: “modifying, by said computer, using a first license key and an encoding algorithm, said software code, to form a modified software code; and wherein said modifying comprises encoding said first code resource to form an encoded first code resource”

Beetcher discloses element 13.3. As described with respect to element 12.3 (supra), Beetcher’s distributor system includes a computer that encodes software code using a first license key (e.g., triggering information) and an encoding algorithm (e.g., Figure 7). And Beetcher’s encoding process modifies the software code by inserting triggering information into the code. (Beetcher at 8:13-23., 9:1-20; see also id at 5:38-47, 9:1-10, 9:16-20, Fig. 7; Silva Declaration at 91) For instance, Beetcher details that system inputs compiled software code into a translator which modifies the code by “automatically generat[ing] a substantial number of entitlement verification triggers” and “insert[ing] them in random locations in the object code.” as shown in Figure 7’s steps 703 and 704 (Beetcher at 9 :11-15). Figure 3 illustrates this modifying by inserting triggering information 301 to form a modified software code:



As described with respect to element 12.2 (supra), Beetcher's software code includes a series of code resources corresponding to sub-objects. And Beetcher teaches a code resource is modified to encode the first code resource via the triggering information (Id. at 4:25-33, 11:11-39; see also id. at Abstract, 3:14-18). For instance, Beetcher teaches:

For support of such a traditional compilation path where the object code format is known by customers, additional barriers to patching of the object code to nullify or alter the entitlement triggering instructions maybe appropriate. One such additional barrier would be to define the entitlement triggering instruction to simultaneously perform some other function. In this case, it is critical that the alternative function performed by the triggering instruction cannot be performed by

any other simple instruction. The alternative function must be so selected that any compiled software module will be reasonably certain of containing a number of instructions performing the function. If these criteria are met, the compiler can automatically generate the object code to perform the alternative function (and simultaneously, the entitlement verification trigger) as part of its normal compilation procedure. This definition would provide a significant barrier to patching of the object code to nullify the entitlement triggering instructions. (Id. at 11:10-28)

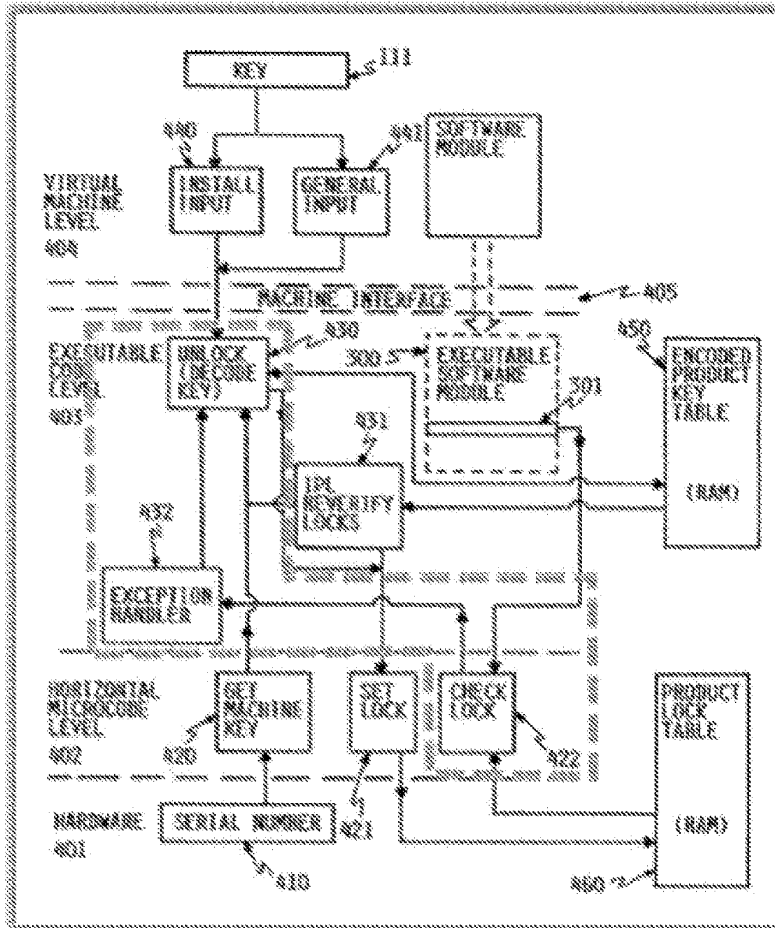
A POSITA would have understood that such modification results in an encoded first code resource (Silva Declaration at ¶ 92).

Moreover, during the original prosecution, Patent Owner specified that “[e]ncoding using a key and an algorithm is known.”¹⁵³ As such, a POSITA would have understood that Beetcher’s encoding technique necessarily includes a first license key and an encoding algorithm to form a modified encoded first code resource (Silva Declaration at ¶ 93).

e) Element 13.4: “wherein said modified software code comprises said encoded first code resource, and a decode resource for decoding said encoded first code resource”

Beetcher discloses element 13.4. Beetcher explains that its modified software code includes a decode resource for decoding the encoded first code resource. Specifically, Beetcher teaches that executing a trigger 301 invokes check lock function 422, which results in accessing “unlock (decode key)” function 430 upon confirmation that the customer possesses the software’s license key (Beetcher at 10:22-39, 10:52-

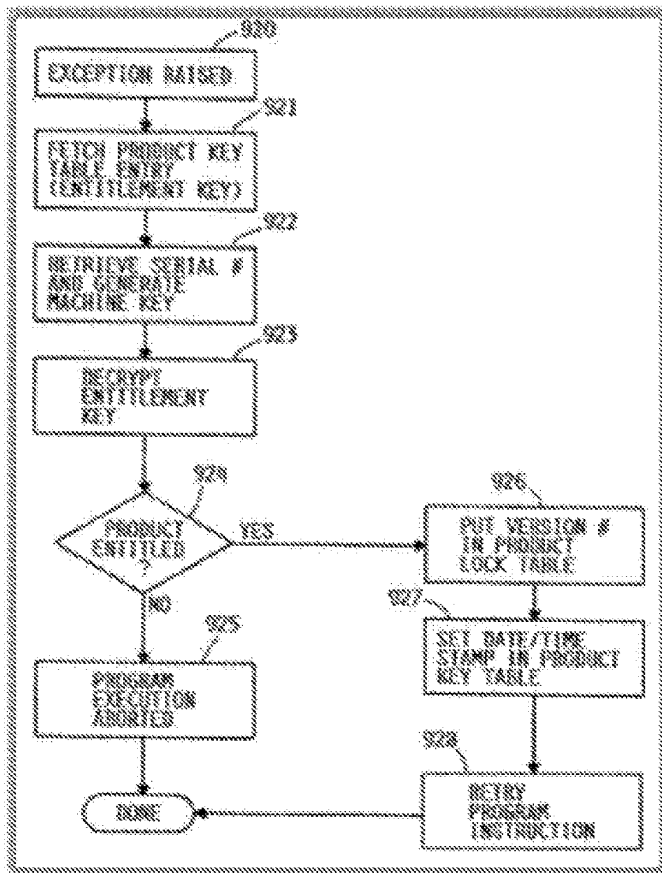
65, Figs. 9b, 10; see also id. at 7:16-38, 8:18-22,9:49-10:7). Beetcher's Figure 4, as annotated below, illustrates the decode resource (dashed perimeter) of the modified software code (Silva Declaration at ¶ 96):



f) Element 13.5 “wherein said decode resource is configured to decode said encoded first code resource upon receipt of said first license key”

Beetcher discloses element 13.5. Beetcher specifies that its decode resource decodes the encoded first Code resource upon receipt of the license key. Beetcher, for

example, states that unlock routine 430 “fetches the encrypted entitlement key from... table 450 ... and decodes the entitlement key The triggering instruction is then retried and program execution continues at step 928.” (Beetcher at 10:27-38) And Beetcher’s Figure 9b illustrates accessing the decode resource to decode the encoded first code resources based on the entitlement key, reflected in steps 921 to 928:



As such, a POSITA would have understood that Beetcher's decode resource is configured to decode the encoded first code resource based on first license key (Silva Declaration at 99-100). Accordingly, Beetcher discloses claim 13.

Claim 14

a) Preamble: “A method for encoding software code using a computer having a processor and memory, comprising”

Under the broadest reasonable construction, the preamble is non-limiting.

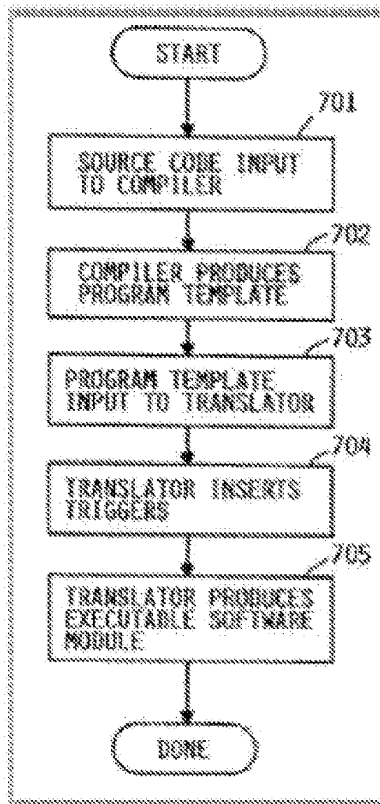
Nevertheless, Beetcher discloses claim 14’s preamble. Claim 14’s preamble is the same as each of claim 12 and 13’s preamble. As explained above, Beetcher discloses a method for encoding software using a computer with a processor and memory. As such, Beetcher teaches this preamble (Id. at ¶ 103).

b) Element 14.1: “storing a software code in said memory”

Beetcher discloses element 12.1. Specifically, Beetcher discloses a development system 125 for compiling and translating for the software code (Beetcher at 5:38-48, 9:1-20). Beetcher details that the software code is stored as disks 112 in warehouse 120. A POSITA would have understood that developer system 125 stores the compiled and translated code in memory and records that code onto disks 112 for distribution to customers. As expert Dr. Silva explains in his declaration (Ex. 9), Beetcher’s computer would necessarily include store software code in memory in order to function. (Silva Declaration at ¶ 62) (Id. at ¶ 105).

c) Element 14.2: “wherein said software code defines software code interrelationships between code resources that result in a specified underlying functionality when installed on a computer system”

Beetcher discloses element 14.2. Beetcher details that its software code is compiled into executable code by compiler 126. This compiler works with translator 127 to compile the software sub-objects and insert triggering information (Beetcher at 8:14-17). And Beetcher specifies that translator 127 “resolves references” in the software code, which corresponds to defining code interrelationships between code resources (Id. at 9:11-18; Silva Declaration at ¶ 107). As shown in steps 701 and 702 of Figure 7, Beetcher teaches its software code is input into compiler 126 that produces a template of the software code: (Beetcher at 8:14-17, 9:1-20, Fig. 7; see also id. at 5:37-39, 6:41-45, 7:63-66)



A POSITA would have understood that this software code template also defines the code interrelationships- between the code resources.164 As the Patent Owner specified during the original prosecution, software code interrelationships are defined during the compiling process of conventional software applications:

What the examiner has implied by alleging that the "specification ... fails to teach or mention 'software code interrelationships'" is that software-code interrelationships were somehow unknown in the art, which clearly is not the case. As admitted, in the specification at the beginning of paragraph [0051], an "application" comprises "sub-objects" whose "order in the computer memory is of vital importance" in order to perform an intended function. And as admitted further in paragraph [0051J]. "When a program is compiled, then, it consists of a collection of these sub-objects, whose exact order or arrangement in memory is not important, so long as any sub-object which uses another sub-object knows where in memory it can be found." Paragraph [0051] of course refers to conventional applications. Accordingly, that is admittedly a discussion of

what is already know by one skilled in the art. Accordingly, the examiner's statement that the specification lacks written description support for "software code interrelationships" is inconsistent with the fact that such interrelationships were explained in paragraphs [0051] and [0052] as a fundamental basis of preexisting modem computer programs. (Ex. 2, Prosecution History at 519)

Moreover, during the original prosecution, Patent Owner specified that "interrelationships between code resources are not that which is novel." (Id.) Based on Patent Owner's concessions, it is deemed that a POSITA would have understood that Beetcher's code necessarily defines code interrelationships between code resources (Silva Declaration at ¶ 109).

Beetcher further teaches that the code resource interrelationships specify the underlying application functionalities when installed on the customer's computer 101. For instance, Beetcher's software code includes multiple entitlement verification triggers (Beetcher at 4:15-33, 9:1-3, 10:22-34, Fig. 3; see also id. At 6:45-65, 8:19-22, 10:52-11:39). And Beetcher details that certain code resources include triggering instruction that controls the underlying functionalities of the software code:

[An] additional barrier would be to define the entitlement triggering instruction to simultaneously perform some other function.... The alternative function must be so selected that any compiled software module will be reasonably certain of containing a number of instructions performing the function. If these criteria are met, the compiler can automatically generate the object code to perform the alternative function (and simultaneously, the entitlement verification trigger) as part of its normal compilation procedure. This definition would provide a significant barrier to patching of the object code to nullify the entitlement triggering instructions. (Id. at 11:14-28; see also id. at 4:25-33, 6:58-65)

Beetcher further explains that “the triggering instruction is also a direct instruction to perform some other useful work [E]xecution of the triggering instruction causes system 101 to perform some other operation simultaneous with the entitlement verification.” (Id. at 6:58-65 (Beetcher specifies that these functions are those “which do not require that an operand for the action be specified in the instruction.”)) As such, a POSITA would have understood that the code interrelationships between Beetcher’s code resources result in a specified underlying functionality once installed. (Silvia Declaration at ¶¶ 110-11)

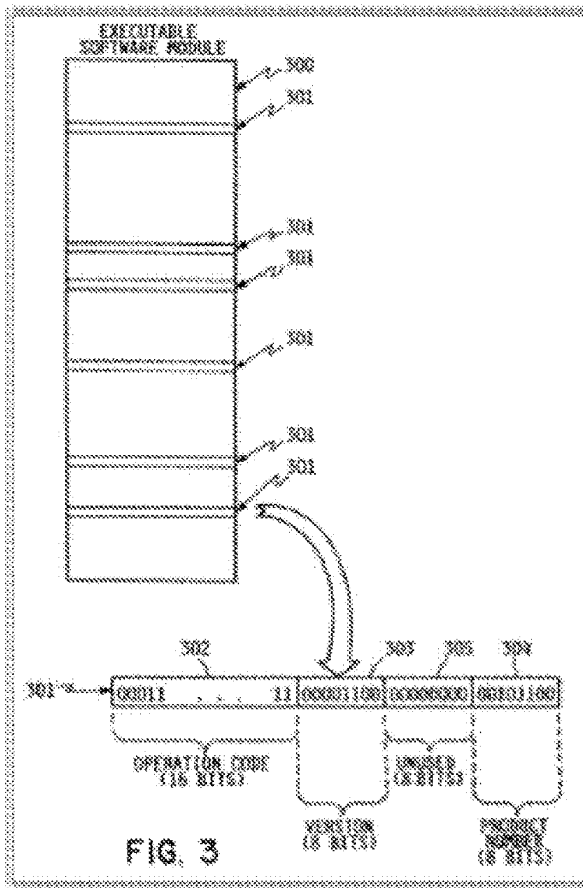
d) Element 14.3: “encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key encoded software code”

Beetcher discloses element 12.3. Beetcher describes encoding its software code by the distributor system that includes development system 125 and marketing system 124, which may be “a single computer system performing both functions.” (Beetcher at 5:37-58, 6:41-65, 11:4-39) Specifically, Beetcher describes encoding a first license key into the software code where that key is used to authorize access to the software product:

Software module 300 is part of a program product in compiled object code form which executes on system 101.... [T]he actual executable code operates at executable code level 403, as shown by the box in broken lines. The executable

code contains entitlement verification triggering instructions 301 (only one shown), which are executed by horizontal microcode check lock function 422. (Id. at 8:13-23; see also id. at 4:3-21, 6:20-55, 7:39-44, 8:58-67, 9:51-56, 10:22-38)

This encoding is illustrated in Figure 3:

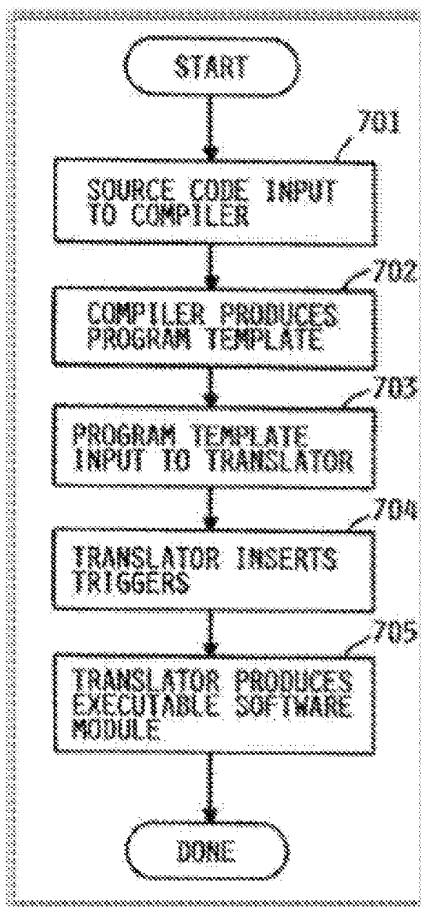


The computer in Beetcher's development system 125 performs the encoding, as shown in Figure 7 at step 704, detailed as: "The program template serves as input to translator 127 at step 704, along with its product number and version number identification. Translator 127 automatically generates a substantial number of

entitlement verification triggers, inserts them in random locations in the object code.....”

(Id. at 9:10-16; see also id< at 5:3'8-47, 9:1-10, .9:16-20, Fig. 7; Silva Declaration at 70-72)

Moreover, the computer in Beetcher's development system 125 uses an encoding algorithm to encode the first license key. Beetcher's system uses a set of instructions, as shown in Figure 7, to encode triggers into the software code to form the first license key : (Beetcher at 9:10-16: see also id. at 5 :38-47, 9:1-1Q, 9:16-20, Fig. 7; Silva Declaration at ¶ 73)



The compiler begins the process by producing a template (step 702), next the template is input into the translator (step 703), then the translator encodes the triggers/license keys into the code (step 704). And finally the translator resolves references after key insertion to produce the executable module (Beetcher at 9:6-20, Figure 7). As such, a POSITA would have understood Beetcher's Figure 7 illustrates an encoding algorithm. (Silva Declaration at ¶74) Beetcher's encoding process is further described with respect to element 11.3.

Moreover, during the original prosecution, Patent Owner specified that "[e]ncoding using a key and an algorithm is known." (Ex. 2, Prosecution History at 519) As such, a POSITA would have understood that Beetcher's encoding technique necessarily includes a first license key and an encoding algorithm to form a first license key encoded software code (Silva Declaration at 70-75 and 114-115).

e) Element 14.4: "in which at least one of said software code interrelationships are encoded"

Beetcher discloses element 14.4. As described with respect to element 14.2, Beetcher teaches that its software code defines code interrelationships between code resources and triggering information 301 in the code control certain underlying software functionality. And Beetcher details that triggering information 301 is encoded into the software code. (Beetcher at 4:25-33, 6:58-65, 11:4-39) For instance, Beetcher explains

that the triggering instructions will be encoded into the code resources controlling software functionality:

[An] additional barrier would be to define the entitlement triggering instruction to simultaneously perform some other function.... The alternative function must be so selected that any compiled software module will be reasonably certain of containing a number of instructions performing the function. If these criteria are met, the compiler can automatically generate the object code to perform the alternative function (and simultaneously, the entitlement verification trigger) as part of its normal compilation procedure. This definition would provide a significant barrier to patching of the object code to nullify the entitlement triggering instructions. (Id. at 11:14-28; see also id. at 4:25-33, 6:58-65)

And Beetcher details that “the triggering instruction is also a direct instruction to perform some other useful work [E]xecution of the triggering instruction causes system 101 to perform some other operation simultaneous with the entitlement verification.” (Id. at 6:58-65 (Beetcher specifies that these functions are those “which do not require that an operand for the action be specified in the instruction.”)) Accordingly, a POSITA would have understood that this encoded triggering information includes encoded code interrelationship of the coder resources. (Silva Declaration at ¶ 117-19)

Accordingly, Beetcher discloses claim 14.

Cooperman Reference

Claims 11, 12, and 13 are anticipated by Cooperman under 35 U.S.C. § 102(a)/(e).

In summary:

Cooperman teaches a method for protecting software through the use of license code. Cooperman accomplishes copyright protection by using a utility application to select 'essential code resources' and encodes the 'data resources' using a key in a stegacipher process. These 'essential code resources' are then not accessible just by installing the software, rather a key will be needed to access the software (see page 10, lines 7-29). Cooperman teaches that a mapping is added at assemble time which specifies which 'data resource' a particular 'code resources' is encoded as. When programs are first run, the system prompts the user to enter a license code, then generates a decoding key to access the 'essential code resource' (see page 11, lines 9-33).

Claim 11

a) Preamble: "A method for licensed software use, the method comprising"

Under the broadest reasonable construction, the preamble is non-limiting. Nevertheless, Cooperman discloses claim 11's preamble. Specifically, Cooperman describes a method for use of licensed software (Cooperman at 5:35-6:5, 11:24-33; see also id. at 3:24-31, 11:34-37, 12:13-35, claim 2). Cooperman, for instance, provides a

method of encoding a license key into software code where the code operates by “ask[ing] the user for personalization information, which include the license code.” (Id. at 11:24-33) And Cooperman specifies that, to extract a digital watermark essential to operate the software, “the user must have a key. The key, in turn, is a function of the license information for the copy of the software in question.” (Id. at 12:13-16)

As such, Cooperman teaches this preamble. (Silva Declaration at ¶ 208-10)

b) Element 11.1: “loading a software product on a computer, said computer comprising a processor, memory, an input, and an output, so that said computer is programmed to execute said software product”

Cooperman discloses element 11.1. Specifically, Cooperman’s system includes a computer having a processor, memory, input, and output. Cooperman initially recognizes that “[a] computer application seeks to provide a user with certain utilities or tools, that is, users interact with a computer or similar device to accomplish various tasks and applications provide the relevant interface.” (Cooperman at 3:16-20) And Cooperman discloses loading software object code into “computer memory for the purpose of execution.” (Id. at claim 5; see also id. at 13:31-36, claim 7) Cooperman further discusses that software products include functions made from executable object code whose “order in the computer memory is of vital importance.” (Id. at 7:1-5) Accordingly, a POSITA would have understood that Cooperman’s computer includes a processor and memory for executing the stored software code because, as expert Dr.

Silva explains, inclusion of a processor and memory is standard in such computers.

(Silva Declaration at ¶212)

Cooperman explains that the computer may “process [] a digital sample stream for the purpose of modifying it or playing the digital sample stream.” (Cooperman at claim 4; see also id. at claims 5, 6 (processing digital sample stream and a map list)) A POSITA would have understood that such digital sample stream processing is performed by a computer’s processor and an output plays the digital sample stream.

(Silva Declaration at ¶ 213)

Cooperman further describes loading a software product on the computer, so the computer can execute the software product. For instance, Cooperman further describes the operation of the disclosed software product requires:

1. Installing, i.e., loading, the software on the computer;
2. Asking the user to input a license code;
3. Generating, i.e., outputting, a decoding key after receiving the license code to access the software resources. (Cooperman at 11:24-34; Silva Declaration at ¶ 214)

c) Element 11.2: “said software product outputting a prompt for input of license information”

Cooperman discloses element 11.2. Specifically, Cooperman explains that its software product requests that the user input license information, i.e., a license key, into the computer before the product can be used. (Cooperman 11:24-33; see also id. at Abstract, 3:24-28, 5:35-6:5, 11:6-8, 12:10-16) For instance, Cooperman explains that the software product prompts the user to input license information: “1) when it is run for the first time, after installation, it asks the user for personalization information, which includes the license code. This can include a particular computer configuration.” (Id. at 11:25-28) Cooperman specifies that such license codes are entered by the user “when prompted at start-up.” (Id. at 1:25-28) A POSITA would have understood this request corresponds to the software product outputting a prompt to input license information. (Silva Declaration at ¶ 216)

During the original prosecution, Patent Owner confirmed that such a teaching disclosed by Cooperman meets element 11.2. For instance, Patent Owner’s May 14, 2012 Appeal Brief states that element 11.2 is taught by: “1) when it is run for the first time, after installation, it asks the user for personalization information, which includes the license code. This can include a particular computer configuration.” (Ex. 2, Prosecution History at 577 (original claim 58 issued as claim 11)) Cooperman includes this same teaching, and thus discloses element 11.2 (Silva Declaration at ¶ 217).

d) Element 11.3: “said software product using license information entered via said input in response to said prompt in a routine designed to decode a first license code encoded in said software product”

Cooperman discloses element 11.3. Specifically, Cooperman explains that its system includes a routine designed to decode a first license code encoded in the software product based on inputted license information. For instance, Cooperman states:

Given that there are one or more of these essential resources, what is needed to realize the present invention is the presence of certain data resources of a type which are amenable to the "stega-cipher" process described in the "Steganographic Method and Device" patent application. (Cooperman at 9:22-27; see also id. at 2:34-37, 4:7-17 (incorporating by reference U.S. Patent Application No. 08/489,172 entitled "Steganographic Method and Device"))

And Cooperman discloses: "3) Once it has the license code, it can then generate the proper decoding key to access the essential code resources." (Cooperman at 11:31-33)

As explained regarding element 11.2, Cooperman details that the user enters license information via an input in response to the prompt.

During the original prosecution, Patent Owner confirmed that such a teaching disclosed by Cooperman meets element 11.3. For instance, Patent Owner's May 14, 2012 Appeal Brief states that element 11.3 is taught by:

Given that there are one or more of these essential resources, what is needed to realize the present invention is the presence of certain data resources of a type which are amenable to the "stega-cipher" process described in the "Steganographic Method and Device" patent U.S. Pat. No. 5,613,004 [issued from U.S. Application No. 08/489,172],

* * * *

3) Once it has the license code, it can then generate the proper decoding key to access the essential code resources. (Ex. 2, Prosecution History at 577 (original claim 58 issued as claim 11); see also id. at 664 (Patent Owner explaining that element 11.3 is met by teachings corresponding to Cooperman at 10:7-11:33))

Cooperman includes these same teachings, and thus discloses element 11.3. (Silva Declaration at ¶ 220-23)

Accordingly, Cooperman discloses claim 11.

Claim 12

a) Preamble: “A method for encoding software code using a computer having a processor and memory, the method comprising”

Under the broadest reasonable construction, the preamble is non-limiting. (Claim 12’s preamble recites “a computer” and claim 12’s body recites “a computer system.” It is unclear whether those elements refer to the same or separate computing devices.

For purposes of this Request and using the broadest reasonable interpretation consistent with the specification, it is assumed that the “computer” recited in the preamble is a device separate from the “computer system.”) Nevertheless, Cooperman discloses claim 12’s preamble. Specifically, Cooperman describes a method for encoding software code using a computer with a processor and memory.

Cooperman details that, during the software code assembly, the computer system will “choose one or several essential code resources, and encode them into one or several

data resources using the stegacipher process.” (Cooperman at 10:13-16; see also id. at claim 6) As expert Dr. Silva explains, Cooperman’s computer would necessarily include a processor and memory in order to function. (Silva Declaration at ¶ 227)

As such, Cooperman teaches this preamble. Id. at ¶¶ 226-28

b) Element 12.1: “storing a software code in said memory”

Cooperman discloses element 12.1. Specifically, Cooperman describes techniques for randomizing the location of software code stored in memory. (Cooperman at 3:32-37; see also id at 4:1-6, 6:5-9, 13:23-46, 14:4-9) Cooperman explains that this randomization makes the software code more resistant to patching and memory capture analysis. (Id. at 3:13-16, 14:37-15:18, claim 7) As such, a POSITA would have understood that these techniques are used for code stored in memory because, as expert Dr. Silva explains, storage of code in memory is standard in computers like Cooperman’s. (Silva Declaration at ¶ 230)

Cooperman further explains that its software code is compiled and assembled: “When code and data resources are compiled and assembled into a precursor of an executable program the next step is to use a utility application for final assembly of the executable application.” (Cooperman at 10:8-11; see also id. at 7:1-21) Cooperman also states that code resources are stored separately from applications, i.e., software, code. (Id. at 7:26-30) A POSITA would have understood that Cooperman’s compiled and assembled application code is stored in memory. As Dr. Silva explains,

Cooperman's computer would necessarily include store software code in memory in order to function. (Silva Declaration at ¶ 231)

During the original prosecution, Patent Owner confirmed that such a teaching disclosed by Cooperman meets element 12.1. For example, Patent Owner's May 14, 2012 Appeal Brief states that element 12.1 is taught by: "When code and data resources are compiled and assembled into a precursor of an executable program the next step is to use a utility application for final assembly of the executable application." (Ex. 2, Prosecution History at 578 (original claim 59 issued as claim 12); see also *id.* at 415-16 (original claim 61, which issued as claim 13, includes the same limitation "wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system"); Silva Declaration at 232)

Cooperman includes this same teaching, and thus discloses element 12.1.

c) Element 12.2: "wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system"

Cooperman discloses element 12.2. Specifically, Cooperman explains that its software code includes multiple code resources that include a first code resource. (Cooperman at 10:11-29, 11:13-33; see also *id.* at Abstract, 7:26-30, 9:10-21, 13:31-36,

claim 6) And Cooperman discloses that its software code includes the code resources and provides an underlying functionality when installed on the computer. (Id. at 7:19-36, 11:24-37; see also id. at 8:30-33, 10:11-29) For instance, Cooperman states: "The basic premise for this scheme is that there are a certain sub-set of executable code resources, that comprise an application and that are 'essential' to the proper function of the application." (Id. at 8:30-33)

As another example, Cooperman details that software applications include code resources providing functionalities specified in the application:

The memory address of the first instruction in one of these sub-objects is called the "entry point" of the function or procedure. The rest of the instructions comprising that sub-object immediately follow from the entry point. Some systems may prefix information to the entry point which describes calling and return conventions for the code which follows, an example is the Apple Macintosh Operating System (MacOS). These sub-objects can be packaged into what are referred to in certain systems as "code resources," which may be stored separately from the application, or shared with other applications, although not necessarily. Within an application there are also data objects, which consist of some data to be operated on by the executable code. These data objects are not executable. That is, they do not consist of executable instructions. The data objects can be referred to in certain systems as "resources." (Id. at 7:19-36)

The '842 Patent refers to sub-objects and a memory scheduler as examples of code resources. ('842 Patent at 11:55-65, 15:36-42) In this additional and alternative way, a POSITA would have understood that Cooperman's sub-objects and code resources. (Silva Declaration at ¶ 235-36)

During the original prosecution, Patent Owner confirmed that such teachings disclosed by Cooperman meets element 12.2. For example, Patent Owner's May 14, 2012 Appeal Brief states that element 12.2 is taught by: "The basic premise for this scheme is that there are a certain subset of executable code resources, that comprise an application and that are 'essential' to the proper function of the application." (Ex. 2, Prosecution History at 578 (original claim 59 issued as claim 12)) As another example, Patent Owner's May 14, 2012 Appeal Brief states this element is taught by:

The memory address of the first instruction in one of these sub-objects is called the "entry point" of the function or procedure. The rest of the instructions comprising that sub-object immediately follow from the entry point. Some systems may prefix information to the entry point which describes calling and return conventions for the code which follows, an example is the Apple Macintosh Operating System (MacOS). These sub-objects can be packaged into what are referred to in certain systems as "code resources" which may be stored separately from the application, or shared with other applications, although not necessarily. Within an application there are also data objects, which consist of some data to be operated on by the executable code. These data objects are not executable. That is, they do not consist of executable instructions. The data objects can be referred to in certain systems as "resources." (Id. at 579-80 (original claim 61, which issued as claim 13, includes the same limitation "wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system"))

Cooperman includes these same teachings, and thus discloses element 12.2.

(Silva Declaration at ¶ 235-37)

d) Element 12.3: "encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key encoded software code"

Cooperman discloses element 12.3. Specifically, Cooperman describes encoding its software code to form a first license key encode software code. (Cooperman at 10:28-35, 11:6-15; see also id. at 2:27-31, 3:24-31, 12:13-23, claim 6) Cooperman details that this encoding uses a first license key and an encoding algorithm. (Id. at 10:13-16, 11:9-11, claim 6) For instance, Cooperman details that “[t]he assembly utility can be supplied with a key generated from a license code generated for the license in question.” (Id. at 11:9-11) And Cooperman states: “The utility will choose one or several essential code resources, and encode them into one or several data resources using the stegacipher process.” (Id. at 10:13-16; see also id. at claim 6)

During the original prosecution, Patent Owner confirmed that such a teaching disclosed by Cooperman meets element 12.3. For instance, Patent Owner’s May 14, 2012 Appeal Brief states that “encoding, by said computer using at least a first license key and an encoding algorithm, said software code” is taught by:

The assembly utility can be supplied with a key generated from a license code generated for the license in question.

* * * *

The utility will choose one or several essential code resources, and encode them into one or several data resources using the stegacipher process. (Ex. 2, Prosecution History at 578 (emphasis in original) (original claim 59 issued as claim 12))

As another example. Patent Owner's May 14, 2012 Appeal Brief states that "to form a first license key encoded software code" is taught by:

The purpose of this scheme is to make a particular licensed copy of an application distinguishable from any other. It is not necessary to distinguish every instance of an application, merely every instance of a license.

* * * *

3) Once it has the license code, it can then generate the proper decoding key to access the essential code resources. (Ex. 2, Prosecution History at 578-79 (original claim 59 issued as claim 12))

Cooperman includes this same teaching, and thus discloses element 12.3.

Moreover, during the original prosecution, Patent Owner specified that "[e]ncoding using a key and an algorithm is known." (Id. at 519) As such, a POSITA would have understood that Cooperman's encoding technique necessarily includes a first license key and an encoding algorithm to form a first license key encoded software code. (Silva Declaration at 240-43)

e) Element 12.4: "wherein, when installed on a computer system, said first license key encoded software code will provide said specified underlying functionality only after receipt of said first license key"

Cooperman discloses element 12.4. Specifically, Cooperman explains that its first license key encoded software code provides the specified underlying functionality

only after receipt of the first license key. (Cooperman at 10:28-35, 11:6-15; see also id. at 2:27-31, 3:24-31, 12:13-23, claim 6) For instance, Cooperman states: “Once it has the license code, it can then generate the proper decoding key to access the essential code resources. Note that the application.. .must contain the license code issued to the licensed owner, to access its essential code resources.” (Id. at 11:31-37) Cooperman describes that these essential code resources correspond to the underlying functionalities of the software program installed on the computer. (Id. at 5:35-6:9, 11:6-8, 11:31-37, 12:10-16; see also id. At 6:26-30, 7:1-5, 8:25-37,9:14-21; Silva Declaration at ¶ 246-47)

Accordingly, Cooperman discloses claim 12.

Claim 13

a) Preamble: “A method for encoding software code using a computer having a processor and memory, comprising”

Under the broadest reasonable construction, the preamble is non-limiting. Nevertheless, Cooperman discloses claim 13’s preamble. Claim 13’s preamble is the same as claim 12’s preamble. As explained above, Cooperman discloses a method for encoding software using a computer with a processor and memory. As such, Cooperman teaches this preamble. Silva Declaration at 249-50)

b) Element 13.1: “storing a software code in said memory”

Specifically, Cooperman describes techniques for randomizing the location of software code stored in memory. (Cooperman at 3:32-37; see also id at 4:1-6, 6:5-9, 13:23-46, 14:4-9) Cooperman explains that this randomization makes the software code more resistant to patching and memory capture analysis. (Id. at 3:13-16, 14:37-15:18, claim 7) As such, a POSITA would have understood that these techniques are used for code stored in memory because, as expert Dr. Silva explains, storage of code in memory is standard in computers like Cooperman’s. (Silva Declaration at ¶ 230)

Cooperman further explains that its software code is compiled and assembled: “When code and data resources are compiled and assembled into a precursor of an executable program the next step is to use a utility application for final assembly of the executable application.” (Cooperman at 10:8-11; see also id. at 7:1-21) Cooperman also states that code resources are stored separately from applications, i.e., software, code. (Id. at 7:26-30) A POSITA would have understood that Cooperman’s compiled and assembled application code is stored in memory. As Dr. Silva explains, Cooperman’s computer would necessarily include store software code in memory in order to function. (Silva Declaration at ¶ 231)

During the original prosecution, Patent Owner confirmed that such a teaching disclosed by Cooperman meets element 13.1. For example. Patent Owner’s May 14, 2012 Appeal Brief states that element 13.1 is taught by: “When code and data resources are compiled and assembled into a precursor of an executable program the

next step is to use a utility application for final assembly of the executable application.”
(Ex. 2, Prosecution History at 578 (original claim 59 issued as claim 12); see also *id.* at 415-16 (original claim 61, which issued as claim 13, includes the same limitation “wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system”); Silva Declaration at 232)

And during the original prosecution, Patent Owner confirmed that such a teaching disclosed by Cooperman meets element 13.1. For instance, Patent Owner’s May 14, 2012 Appeal Brief states that element 13.1 is taught by: “When code and data resources are compiled and assembled into a precursor of an executable program the next step is to use a utility application for final assembly of the executable application.” (Ex. 2, Prosecution History at 579 (original claim 61 issued as claim 13)) As explained with respect to element 12.1, Cooperman includes this same teaching. (Silva Declaration at 252)

c) Element 13.2: “wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system”

Cooperman discloses element 12.2. Specifically, Cooperman explains that its software code includes multiple code resources that include a first code resource. (Cooperman at 10:11-29, 11:13-33; see also *id.* at Abstract, 7:26-30, 9:10-21, 13:31-36,

claim 6) And Cooperman discloses that its software code includes the code resources and provides an underlying functionality when installed on the computer. (Id. at 7:19-36, 11:24-37; see also id. at 8:30-33, 10:11-29) For instance, Cooperman states: "The basic premise for this scheme is that there are a certain sub-set of executable code resources, that comprise an application and that are 'essential' to the proper function of the application." (Id. at 8:30-33)

As another example, Cooperman details that software applications include code resources providing functionalities specified in the application:

The memory address of the first instruction in one of these sub-objects is called the "entry point" of the function or procedure. The rest of the instructions comprising that sub-object immediately follow from the entry point. Some systems may prefix information to the entry point which describes calling and return conventions for the code which follows, an example is the Apple Macintosh Operating System (MacOS). These sub-objects can be packaged into what are referred to in certain systems as "code resources," which may be stored separately from the application, or shared with other applications, although not necessarily. Within an application there are also data objects, which consist of some data to be operated on by the executable code. These data objects are not executable. That is, they do not consist of executable instructions. The data objects can be referred to in certain systems as "resources." (Id. at 7:19-36)

The '842 Patent refers to sub-objects and a memory scheduler as examples of code resources. ('842 Patent at 11:55-65, 15:36-42) In this additional and alternative way, a POSITA would have understood that Cooperman's sub-objects and code resources. (Silva Declaration at ¶ 235-36)

During the original prosecution, Patent Owner confirmed that such teachings disclosed by Cooperman meets element 12.2. For example. Patent Owner's May 14, 2012 Appeal Brief states that element 12.2 is taught by: "The basic premise for this scheme is that there are a certain subset of executable code resources, that comprise an application and that are 'essential' to the proper function of the application." (Ex. 2, Prosecution History at 578 (original claim 59 issued as claim 12)) As another example, Patent Owner's May 14, 2012 Appeal Brief states this element is taught by:

The memory address of the first instruction in one of these sub-objects is called the "entry point" of the function or procedure. The rest of the instructions comprising that sub-object immediately follow from the entry point. Some systems may prefix information to the entry point which describes calling and return conventions for the code which follows, an example is the Apple Macintosh Operating System (MacOS). These sub-objects can be packaged into what are referred to in certain systems as "code resources" which may be stored separately from the application, or shared with other applications, although not necessarily. Within an application there are also data objects, which consist of some data to be operated on by the executable code. These data objects are not executable. That is, they do not consist of executable instructions. The data objects can be referred to in certain systems as "resources." (Id. at 579-80 (original claim 61, which issued as claim 13, includes the same limitation "wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system"))

Cooperman includes these same teachings, and thus discloses element 12.2.

(Silva Declaration at ¶¶ 235-237 and 254-55)

And during the original prosecution, Patent Owner confirmed that such a teaching disclosed by Cooperman meets element 13.2. For instance. Patent Owner's May 14, 2012 Appeal Brief states that element 13.2 is taught by:

The memory address of the first instruction in one of these sub-objects is called the "entry point" of the function or procedure. The rest of the instructions comprising that sub-object immediately follow from the entry point. Some systems may prefix information to the entry point which describes calling and return conventions for the code which follows, an example is the Apple Macintosh Operating System (MacOS). These sub-objects can be packaged into what are referred to in certain systems as "code resources," which may be stored separately from the application, or shared with other applications, although not necessarily. Within an application there are also data objects, which consist of some data to be operated on by the executable code. These data objects are not executable. That is, they do not consist of executable instructions. The data objects can be referred to in certain systems as "resources." (Ex. 2, Prosecution History at 579-80 (original claim 61 issued as claim 13))

As explained with respect to element 12.2, Cooperman includes this same teaching. (Silva Declaration at 254-55)

d) Element 13.3: "modifying, by said computer, using a first license key and an encoding algorithm, said software code, to form a modified software code; and wherein said modifying comprises encoding said first code resource to form an encoded first code resource"

Cooperman discloses element 13.3. Specifically, Cooperman describes modifying its software code using a license key and an encoding algorithm. (Cooperman at 3:10-31, 8:25-30, 10:8-31; see also id. at 2:19-37, 4:7-17, 11:6-24, claim 6) And Cooperman's modification includes encoding the first code resource to form an encoded first code resource. For instance, Cooperman teaches code modification using a "digital watermarking" process to encode a code resource: "The first method of the present invention described involves hiding necessary 'parts' or code 'resources' in digitized

sample resources using a 'digital watermarking' process, such as that described in the 'Steganographic Method and Device' patent application." (Id. at 8:25-30; see also id. at 2:34-37, 4:7-17 (incorporating by reference U.S. Patent Application No. 08/489,172 entitled "Steganographic Method and Device")) Cooperman further discloses "watermarking with 'keys' derived from license codes... and using the watermarks encoded with such keys to hide an essential subset for the application code resources." (Cooperman at 5:15-22) A POSITA would have understood that such modification results in a modified software code. (Silva Declaration at ¶ 257)

During the original prosecution, Patent Owner confirmed that such a teaching disclosed by Cooperman meets element 13.3. For instance, Patent Owner's May 14, 2012 Appeal Brief states that element 13.3 is taught by: "The first method of the present invention described involves hiding necessary 'parts' or code 'resources' in digitized sample resources using a 'digital watermarking' process, such as that described in the 'Steganographic Method and Device' patent application." (Ex. 2, Prosecution History at 580 (original claim 61 issued as claim 13)) Cooperman includes this same teaching, and thus discloses element 13.3. (Silva Declaration at ¶ 258)

Moreover, during the original prosecution, Patent Owner specified that "[e]ncoding using a key and an algorithm is known." (Ex. 2, Prosecution History at 519) As such, a POSITA would have understood that Cooperman's encoding technique necessarily includes a first license key and an encoding algorithm to form a modified encoded first code resource. (Silva Declaration at ¶ 259)

e) Element 13.4: “wherein said modified software code comprises said encoded first code resource, and a decode resource for decoding said encoded first code resource”

Cooperman discloses element 13.4. Specifically, Cooperman explains that its modified software code includes a decode resource for decoding the encoded first code resource. (Cooperman at 11:17-20, claim 6; see also id. 11:31-33, claim 5)

For instance, Cooperman describes the modified application code has a decoding resource: “Note further that the application contains a code resource which performs the function of decoding an encoded code resource from a data resource.” (Id. at 11:17-20) And Cooperman further discloses that “[o]nce [the application] has the license code, it can then generate the proper decoding key to access the essential code resources.” (Id. at 11:31-33; Silva Declaration at ¶ 262)

During the original prosecution, Patent Owner confirmed that such a teaching disclosed by Cooperman meets element 13.4. For instance, Patent Owner’s May 14, 2012 Appeal Brief states that element 13.4 is taught by: “Note further that the application contains a code resource which performs the function of decoding an encoded code resource from a data resource.” (Ex. 2, Prosecution History at 580 (original claim 61 issued as claim 13)) Cooperman includes this same teaching, and thus discloses element 13.4. (Silva Declaration at ¶ 263).

f) Element 13.5: “wherein said decode resource is configured to decode said encoded first code resource upon receipt of said first license key”

Cooperman discloses element 13.5. Cooperman specifies that its decode resource decodes the encoded first code resource upon receipt of the license key:

The application must also contain a data resource which specifies in which data resource a particular code resource is encoded. This data resource is created and added at assembly time by the assembly utility. The application can then operate as follows:

- 1) when it is run for the first time, after installation, it asks the user for personalization information, which includes the license code. This can include a particular computer configuration;
- 2) it stores this information in a personalization data resource;
- 3) Once it has the license code, it can then generate the proper decoding key to access the essential code resources. (Cooperman at 11:20-33; see also id. at claims 5 and 6)

During the original prosecution, Patent Owner confirmed that such a teaching disclosed by Cooperman meets element 13.5. For instance, Patent Owner’s May 14, 2012 Appeal Brief states that element 13.5 is taught by:

The application must also contain a data resource which specifies in which data resource a particular code resource is encoded. This data resource is created and added at assembly time by the assembly utility. The application can then operate as follows:

- 1) when it is run for the first time, after installation, it asks the user for personalization information, which includes the license code. This can include a particular computer configuration;
- 2) it stores this information in a personalization data resource;
- 3) Once it has the license code, it can then generate the proper decoding key to access the essential code resources. (Ex. 2, Prosecution History at 580-81 (original claim 61 issued as claim 13))

Cooperman includes this same teaching, and thus discloses element 13.5. (Silva Declaration at ¶ 266-68)

Accordingly, Cooperman discloses claim 13.

Hasebe Reference

Claims 12, 13, and 14 are anticipated by Hasebe under 35 U.S.C. § 102(e).

Claim 12

a) Preamble: “A method for encoding software code using a computer having a processor and memory, the method comprising”

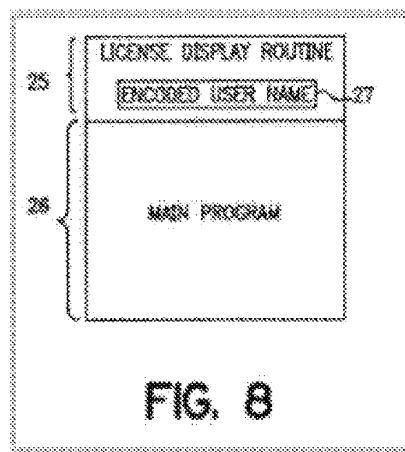
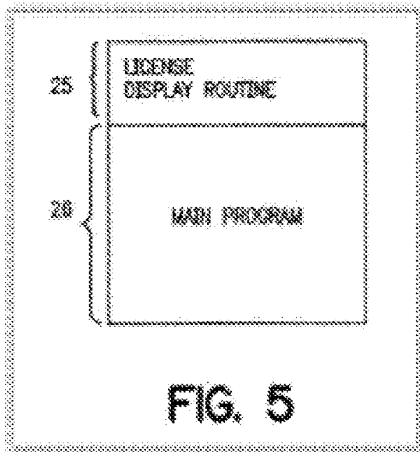
Under the broadest reasonable construction, the preamble is non-limiting. (Claim 12’s preamble recites “a computer” and claim 12’s body recites “a computer system.” It is unclear whether those elements refer to the same or separate computing devices. For purposes of this Request and using the broadest reasonable interpretation consistent with the specification, it is assumed that the “computer” recited in the preamble is a device separate from the “computer system.”) Nevertheless, Hasebe discloses claim 12’s preamble. Specifically, Hasebe describes a method for encoding software code using a computer with a processor and memory. Hasebe details that management center 32 generates the software code provided to the user via CD-ROM. (Hasebe at 1:9-14, 6:9-13, 9:22-26) Alternatively, Hasebe’s software code may be downloaded from the management center. (Id. at 9:60-64) And Hasebe explains that the link-up unit 15 of the management center performs “processing” reversed by separating unit 21. (Id. at 7:23-26, Fig. 1) As such, A POSITA would have understood that the management center includes a processor and memory to create these CD-

ROMs and to provide the downloading capability. As expert Dr. Silva explains, Hasebe's computer would necessarily include a processor and memory in order to function. (Silva Declaration at ¶ 314-17)

As such, Hasebe teaches this preamble.

b) Element 12.1: "storing a software code in said memory"

Hasebe discloses element 12.1. As described with respect to claim 12's preamble, Hasebe's management center 32 either generates a CD-ROM containing the software code or provides downloadable versions of the software code. (Hasebe at 6:9-13, 9:22-26, 9:60-64) A POSITA thus would have understood that Hasebe's management center stores the software code in its memory for CD-ROM generation or user downloading because, as Dr. Silva explains, storage of code in memory is standard in computers like Hasebe's. (Silva Declaration at ¶ 319) And as shown in Hasebe's Figure 1, as annotated below, management center 14 includes a software database 14 (dashed box) capable of software storage:



The '842 Patent refers to sub-objects, a memory scheduler, and data as examples of code resources. ('842 Patent at 11:55-65, 15:36-42) Hasebe's routine 25 consists of software code that controls access to the underlying functionality of the software's main program, or sub-objects. (Silva Declaration at 323-24) In this additional and alternative way; a POSIT A would have understood that Hasebe's routine 25 contains a first code resource. (Id. at 324)

Moreover, Hasebe's software code provides underlying functionalities when installed on the user's computer system (terminal 31). Hasebe, for instance, explains that the code's routine 25 provides access to the main program module 26 upon verification of the user's license information. (Hasebe at 7:65-8:9, 9:2Q-36; Silva Declaration at ¶ 325)

d) Element 12.3: “encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key encoded software code”

Hasebe discloses element 12.3. As discussed with respect to element 12.1, Hasebe’s management center 32 provides the user the software code via CD-ROM or download from the seller. (Hasebe at 6:9-13, 9:22-26, 9:60-64) Hasebe details that the management center 32 encodes the software code:

[I]t is also possible to make the software that is presented to the user encoded, and to make the conversion information for decoding the encoded software. Also, it is possible to employ, in such a licensee notification system, license information containing the user identification information in a form that cannot be separated without special information. For example, it is possible to employ information, as license information, which is the result of encoding the conversion information and user identification information, combined in integrated manner. (Id. at 4:48-58; see also id. at 7:32-38, 9:22-26)

It is also possible to constitute the system such that, instead of the user name and signature information, information representing the user name in encoded form is stored in the license file, and, when the installed software is executed, the information in the license file is decoded by the software and displayed. (Id. at 8:47-53)

With respect to the code illustrated in Figure 9, Hasebe explains that the customer’s computer system “effects installation by decoding the software in the CD ROM using the software decoding key, and generates the user name in encoded form by encoding the user name.” (Id. at 9:22-26)

Moreover, Hasebe describes its encoding technique uses a license key and an encoding algorithm. For instance, Hasebe states its system includes: “a DES (data encryption standard) algorithm [] employed for encoding and decoding.” (Id. at 6:48-50) And Hasebe details that the system uses a license key to encode the software code: “generat[ing] license information including user identification information encoded with a characteristic key of the software.” (Id. at 4:40-43; see also id. at 6:33-47, 7:33-38, 9:19-26) Figure 3, for example, illustrates a license key in the management center’s software database 14 used to encode the software:

FIG. 3

CONTENTS ID	DECODING KEY
ABC00001	XXXXXXXXXX

As such, a POSIT A would have understood that Hasebe’s encoded software code utilizes the encoded license information to generate the claimed “first license key encoded software code.” (Silva Declaration at ¶ 328-30)

Moreover, during the original prosecution, Patent Owner specified that “[e]ncoding using a key and an algorithm is known.” (Ex. 2, Prosecution History at 519) As such, a POSIT A would have understood that Hasebe’s encoding technique

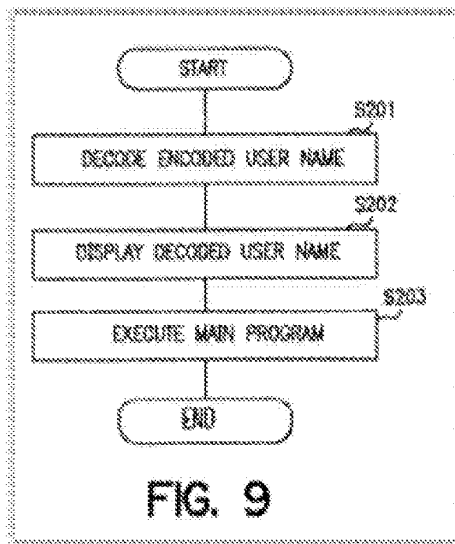
necessarily includes a first license key and an encoding algorithm to form a first license key encoded software code. (Silva Declaration at ¶ 331)

e) Element 12.4: “wherein, when installed on a computer system, said first license key encoded software code will provide said specified underlying functionality only after receipt of said first license key”

Hasebe discloses element 12.4. Hasebe describes the installation of the software code upon verification of the first license key by the user’s computer. (Hasebe at 3:5-15, 3:30-38, 9:19-39; see also id at 7:32-38, 8:47-53) For instance, Hasebe details the software code will provide access to specified underlying functionality of the code contained in main program routine 26 only after receipt of the first license key in license display routine 25:

- (a) decoding the license information, which includes the key and user name,
- (b) installing the encoded software using the decoded key,
- (c) writing the user name into the license display routine 25,
- (d) displaying the user name, and
- (e) executing the main portion routine 26 of software program. (Hasebe at 9:19-39)

And Hasebe’s Figure 9 illustrates the user’s Computer providing the underlying functionality of the main program routine 26 after the receipt and decoding of the first license key:



A POSIT A would have understood that Hasebe's main program routine 26 includes specified underlying functionality of the first license key encoded software code accessible via confirmation of the encoded license key. (Silva Declaration at 334-36)

Accordingly, Hasebe discloses claim 12.

Claim 13

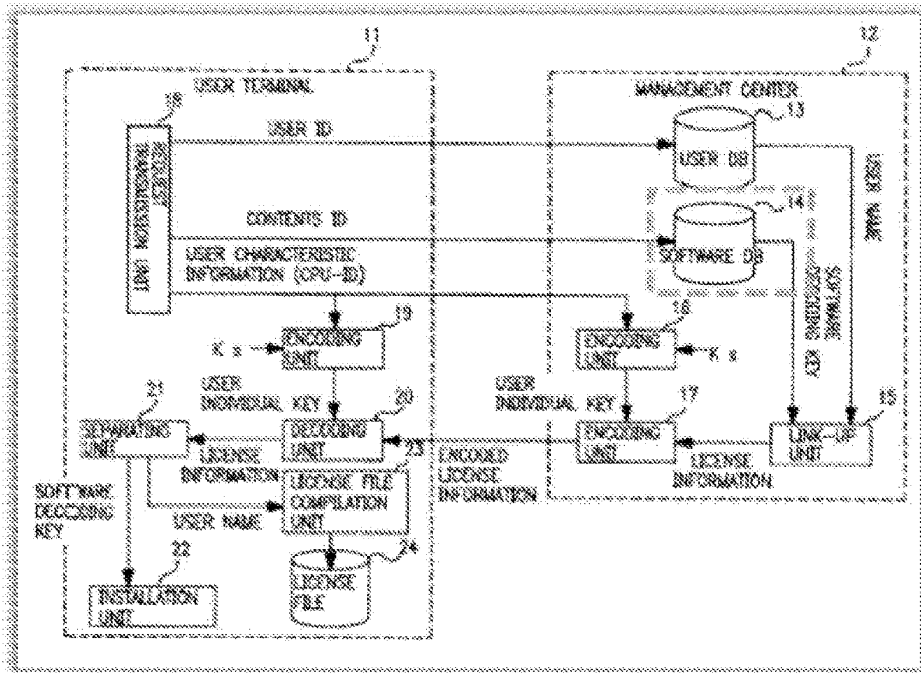
a) Preamble: "A method for encoding software code using a computer having a processor and memory, comprising"

Under the broadest reasonable construction, the preamble is non-limiting. Nevertheless, Hasebe discloses claim 13's preamble. Claim 13's preamble is the same as claim 12's preamble. As explained above, Hasebe discloses a method for encoding

software using a computer with a processor and memory. As such, Hasebe teaches this preamble. (Id. at ¶¶38-39)

b) Element 13.1: “storing a software code in said memory”

Hasebe discloses element 13.1. As described with respect to claim 13’s preamble, Hasebe’s management center 32 either generates a CD-ROM containing the software code or provides downloadable versions of the software code. (Hasebe at 6:9-13, 9:22-26, 9:60-64) A POSITA thus would have understood that Hasebe’s management center stores the software code in its memory for CD-ROM generation or user downloading because, as Dr. Silva explains, storage of code in memory is standard in computers like Hasebe’s. (Silva Declaration at ¶ 319) And as shown in Hasebe’s Figure 1, as annotated below, management center 14 includes a software database 14 (dashed box) capable of software storage:

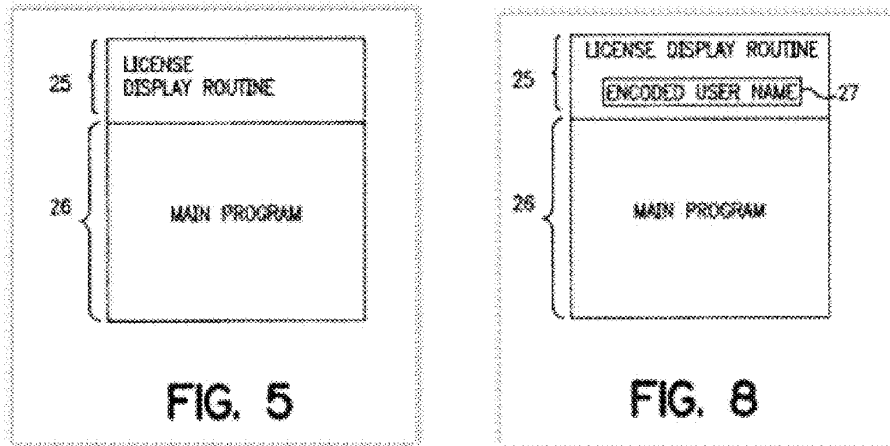


(Id. at ¶ 341)

c) Element 13.2: “wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system”

Hasebe discloses element 13.2. Hasebe teaches that its software code includes multiple code resources such as those used in license display routine 25. (Hasebe at 7:55-8:9, 9:25-35, Figs. 5, 8) Hasebe explains that routine 25 determines whether the user’s license information is legitimate and, if so, permits access to the main program routine 26. (Id. at 7:65-8:9) For instance, Hasebe states: “In the main program there are defined the operating procedures relating to the proper functions of this software; in license display routine 25, there is defined the content to be executed prior to execution

of main program 26. (Id at 7:55-60) Hasebe illustrates routines 25 and 26 of the software code in Figures 5 and 8:



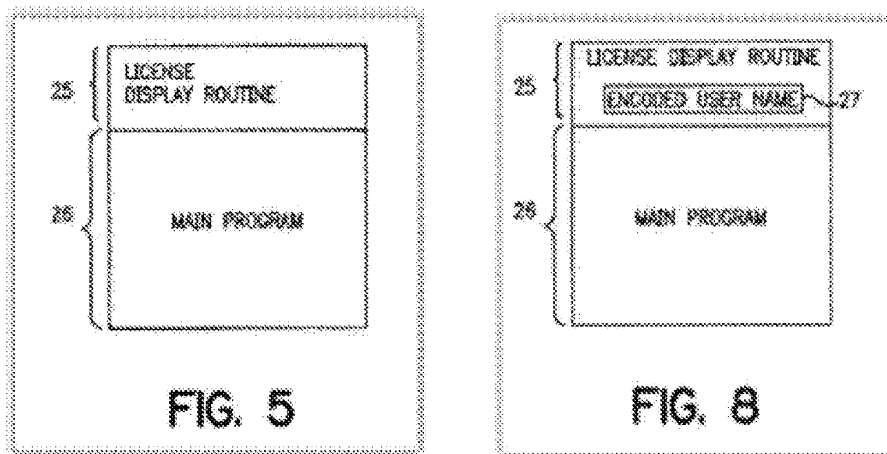
The '842 Patent refers to sub-objects, a memory scheduler, and data as examples of code resources. ('842 Patent at 11:55-65, 15:36-42) Hasebe's routine 25 consists of software code that controls access to the underlying functionality of the software's main program, or sub-objects. (Silva Declaration at 323-24) In this additional and alternative way; a POSIT A would have understood that Hasebe's routine 25 contains a first code resource. (Id. at 324)

Moreover, Hasebe's software code provides underlying functionalities when installed on the user's computer system (terminal 31). Hasebe, for instance, explains that the code's routine 25 provides access to the main program module 26 upon verification of the user's license information. (Hasebe at 7:65-8:9, 9:2Q-36; Silva Declaration at ¶ 325)

(Id. at ¶ 343)

d) Element 13.3: “modifying, by said computer, using a first license key and an encoding algorithm, said software code, to form a modified software code; and wherein said modifying comprises encoding said first code resource to form an encoded first code resource”

Hasebe discloses element 13.3. As described with respect to element 12.2, Hasebe’s system includes multiple code resources (e.g., license display routine 25) for accessing software functionality.⁴¹⁰ Hasebe illustrates routine 25 and main program routine 26 of the software code in Figures 5 and 8:



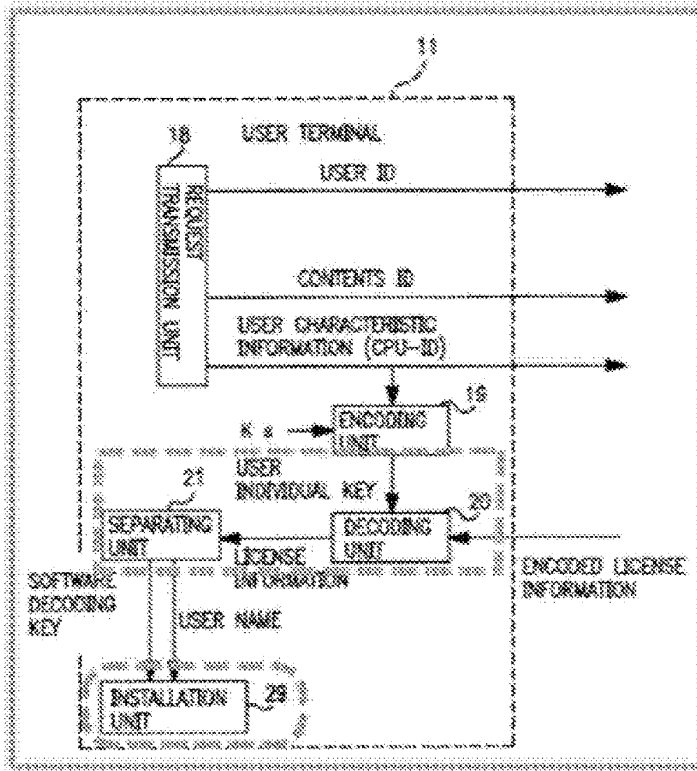
And as described with respect to element 12.3, Hasebe’s computer (Hasebe at 6:21-24) in management center 12 modifies the software code to form an encoded first code resource. (Id. at 4:48-58, 8:47-53; see also id. at 7:32-38, 9 22-26; Silva Declaration at ¶ 346) For example, Hasebe’s software code is modified to include routine 25 used for verification of the user’s license information, which permits execution of the software code. (Hasebe at 4:48-58, 8:47-53; Silva Declaration at ¶ 346)

Hasebe discloses that its code modification uses a license key and an encoding algorithm, as described with respect to element 12.3. (Hasebe at 6:48-50, 4:40-43, Fig. 3; see also *id.* at 6:33-47, 7:33-38, 9:19-26) Moreover, during the original prosecution, Patent Owner specified that “[e]ncoding using a key and an algorithm is known.” (Ex. 2, Prosecution History at 519) As such, a POSITA would have understood that Hasebe’s encoding technique necessarily includes a first license key and an encoding algorithm to form an encoded first code resource. (Silva Declaration at ¶ 347.)

e) Element 13.4: “wherein said modified software code comprises said encoded first code resource, and a decode resource for decoding said encoded first code resource”

Hasebe discloses element 13.4. As described with respect to element 13.3, Hasebe’s modified software code includes the encoded first code resource. And Hasebe details that user terminal 11 includes decode unit 20 and separating unit 21 to produce the decoding key for the relevant software code. (Hasebe at 7:17-31) Hasebe’s user terminal sends the decoding key to the software installation unit (Fig. 1’s unit 22 or Fig. 7’s unit 29), and “[i]n installation unit 29 effects installation by decoding the software in the CD ROM using the software decoding key, and generates the user name in encoded form by encoding the user name.” (*Id.* at 7:27-39, 9:22-26) As shown below in annotated Figure 7, Hasebe’s user terminal 11 includes a decode resource

including the separating and decoding units 20, 21 (dashed box) and installation unit 22 (dashed oval) to decode the encoded code resource for software execution:



As such, Hasebe teaches a decoding resource for decoding the encoded first code resource. (Silva Declaration at ¶ 350)

f) Element 13 Si “wherein said decode resource is configured to decode said encoded first code resource upon receipt of said first license key”

Hasebe discloses element 13.5. As described with respect to element 12.3.

Hasebe details that the system uses a license key to encode the software code:

“generating license information including user identification information encoded with a characteristic key of the software.” (Hasebe at 4:40-43; see also id. at 6:33-47, 7:33-38, 9:19-26, Fig. 3) And Hasebe specifies that its decode resource decodes the encoded first code resource upon receipt of the license key. For instance, Hasebe teaches that the user terminal receives the encoded license information at decoding unit 20, decodes the information to produce the decoding key, and decodes the encode first code resource (routine 25) “by decoding the software in the CD ROM using the software decoding key..”⁴²¹ Figure 7, as annotated below, shows the decode resource (dashed box) receiving the first license key (dashed Oval) to decode the encoded software including the encoded first code resource:

Accordingly, Hasebe discloses claim 13.

Claim 14

Claims 14 is anticipated by Hasebe under 35 U.S.C. § 102(e).

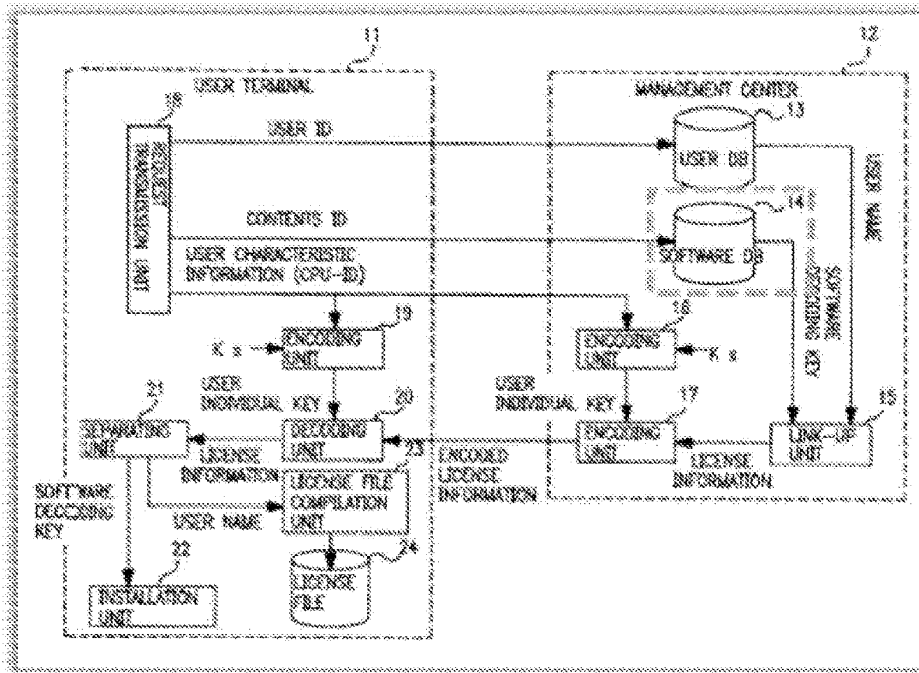
a) Preamble: “A method for encoding software code using a computer having a processor and memory, comprising”

Under the broadest reasonable construction, the preamble is non-limiting.

Nevertheless, Hasebe discloses claim 14's preamble. Hasebe discloses a method for encoding software using a computer with a processor and memory. As such, Hasebe teaches this preamble. (Silva Declaration at ¶¶ 356-57)

b) Element 14.1: "storing a software code in said memory"

Hasebe discloses element 14.1. As described with respect to claim 14's preamble, Hasebe's management center 32 either generates a CD-ROM containing the software code or provides downloadable versions of the software code. (Hasebe at 6:9-13, 9:22-26, 9:60-64) A POSITA thus would have understood that Hasebe's management center stores the software code in its memory for CD-ROM generation or user downloading because, as Dr. Silva explains, storage of code in memory is standard in computers like Hasebe's. (Silva Declaration at ¶¶ 319) And as shown in Hasebe's Figure 1, as annotated below, management center 14 includes a software database 14 (dashed box) capable of software storage:

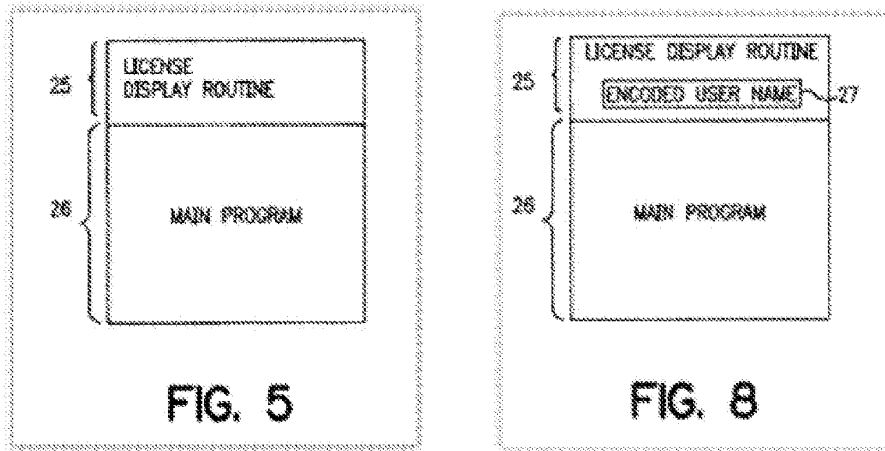


Hasebe teaches element 14.1. (Id. at ¶359)

c) Element 14.2: “wherein said software code defines software code interrelationships between code resources that result in a specified underlying functionality when installed on a computer system”

Hasebe discloses element 14.2. Hasebe explains that its software code interrelates code resources relating to routines 25 and 26 upon verification of the license key. (Hasebe at 7:55-8:9, Figs. 5, 8, 9) For instance, Hasebe details that its software code includes routine 25 which permits access to the main program routine 26 upon validation of user’s license information. (Id. at 7:65-8:9) Hasebe states: “In the main program there are defined the operating procedures relating to the proper functions of this software; in license display routine 25, there is defined the content to be

executed prior to execution of main program 26.” (Id. at 7:55-60) Hasebe illustrates routines 25 and 26 of the software code in Figures 5 and 8:



Moreover, the '842 Patent refers to sub-objects and a memory scheduler as examples of code resources. ('842 Patent at 11:55-65, 15:56-42) Hasebe's routine 25 contains a sub-object of the software code because it controls access to the underlying functionality of the software's main program. (428 Silva Declaration at ¶ 561-62) And Hasebe specifies routine 25 “directly rewrite[es]” the software code when the software code is decoded. (Hasebe at 5:10-32, 9:22-59) In this additional and alternative way, a POSIT A would have understood that Hasebe's routines 25 and 26 are code resources and that the software code defines software code interrelationships between these code resources. (Silva Declaration at ¶ 362) And a POSIT A would have understood that the interrelationship between Hasebe's routines 25 and 26 result in a specified underlying functionality upon code installation. (Id. At ¶ 362)

Moreover, during the original prosecution, Patent Owner specified that "interrelationships between code resources are not that which is novel." (Ex. 2, Prosecution History at 519) Patent Owner continued by conceding:

What the examiner has implied by alleging that the "specification ... fails to teach or mention 'software code interrelationships'" is that software code interrelationships were somehow unknown in the art, which clearly is not the case. As admitted, in the specification at the beginning of paragraph [0051], an "application" comprises "sub-objects" whose "order in the computer memory is of vital importance" in order to perform an intended function. And as admitted further in paragraph [0051], "When a program is compiled, then, it consists of a collection of these sub-objects, whose exact order or arrangement in memory is not important, so long as any sub-object which uses another sub-object knows where in memory it can be found." Paragraph [0051] of course refers to conventional applications. Accordingly, that is admittedly a discussion of what is already know by one skilled in the art. Accordingly, the examiner's statement that the specification lacks written description support for "software code interrelationships" is inconsistent with the fact that such interrelationships were explained in paragraphs [0051] and [0052] as a fundamental basis of preexisting modem computer programs. (Id.)

Based on the Patent Owner's concession, it is clear that a POSITA would have understood that Hasebe's code resources necessarily define code interrelationships resulting in specific underlying functionality once installed on a computer. (Silva Declaration at 363-64)

d) Element 14.3: "encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key encoded software code"

Element 14.3 is identical to element 12.3. As explained above, Hasebe discloses each limitation of element 12.3. For the same reasons, Hasebe teaches element 14.3.

(Id. at ¶ 367)

Moreover, during the original prosecution, Patent Owner specified that “[e]ncoding using a key and an algorithm is known” and that “an interrelationship in software code is necessarily defined by digital data, and digital data can obviously be encoded by an encoding process.” (Ex. 2, Prosecution History at 519) As such, a POSITA would have understood that Hasebe’s encoding technique necessarily includes a first license key and an encoding algorithm to form a first license key encoded software code. (Silva Declaration at ¶ 368)

e) Element 14.4: “in which at least one of said software code interrelationships are encoded”

Hasebe discloses element 14.4. As described with respect to element 14.2, Hasebe teaches that its software code defines code interrelationships between code resources and routine 25 control certain underlying software functionality. Hasebe further details that its software code is encoded:

[I]t is also possible to make the software that is presented to the user encoded, and to make the conversion information for decoding the encoded software. Also, it is possible to employ, in such a licensee notification system, license information containing the user identification information in a form that cannot be separated without special information. For example, it is possible to employ information, as license information, which is the result of encoding the conversion

information and user identification information, combined in integrated manner. (Hasebe at 4:48-58; see also id. at 7:32-38, 9:22-26)

It is also possible to constitute the system such that, instead of the user name and signature information, information representing the user name in encoded form is stored in the license file, and, when the installed software is executed, the information in the license file is decoded by the software and displayed. (Id. at 8:47-53)

And Hasebe states that the software code includes the code interrelationships between routines 25 and 26, all of which would be encoded as part of the software code. (Id. at 7:55-8:9, Figs. 5, 8, 9; Silva Declaration at ¶¶ 370-71)

Accordingly, Hasebe discloses claim 14.

Patent Owner Statement

The Patent Owner filed a Patent Owner Statement on 8/17/2018 arguing against the proposed application of references and further that some of the supplied references are not prior art.

I.1 and IV

With respect to arguments that “Cooperman and Hasebe references are not prior art”:

Patent Owner argues that *“If Mr. Cooperman concludes that he did so contribute, then the undersigned will petition for a certificate of correction to correct the inventorship of the patent by naming Mr. Cooperman, in which case the inventive entities of*

USP9104842 and USP5745569 will be the same, which would prove invention of claims 11-14 not later than 1/17/1996."

In response, the Examiner respectfully submits that USP5745569 is not a reference considered in the reexamination. The Examiner does appreciate the commonality of the references between USP5745569 and applied WO 97/26732. It is further noted that USP9104842 clearly contains more elaborate subject matter than does USP5745569 as the subject patent is twice the length, and previously had no note to nor reliance on USP5745569 for earlier date.

Patent Owner argues that *"If, on the other hand, Mr. Cooperman determines he is not an inventor of the subject matter of any claim, including claims 11-14 of USP9104842, then that is proof that the subject matter defined by claims 11-14, by its disclosure in USP5745569 and US application 08/587,793 is proof that Mr. Moskowitz invented that subject matter prior to the dates of availability of the Cooperman and Hasebe references."*

In response, the Examiner respectfully submits that the argument that if Mr. Cooperman state he is not an inventor of subject matter of the subject Patent (he was not named in) then the Patent Under Reexamination deserves a date prior to Hasebe and Cooperman, simply because of a common disclosure of a portion of the subject matter, is illogical. There is a procedure for correction of inventorship within Reexamination under *37 C.F.R. 1.530*. Those steps have not been followed.

Patent Owner has further not established common inventorship of the subject Patent with Cooperman or Moskowitz et al. (USPN 5,745,569). Patent Owner's statements asserting the commonality between the instant claims, USPN 5,745,569, and applied WO 97/26732 are insufficient alone.

Patent Owner argues that *"If Mr. Cooperman refuses to cooperate, then that fact would also indicate he is not an inventor of the subject matter defined by claims 11-14 in USP9104842."*

In response, the Examiner respectfully submits that Mr. Cooperman has been cooperating by the Patent Owner's own admission, as noted on page 1 of the Patent Owner Statement: *"Mr. Cooperman recently testified in the related district court litigation and the transcript of his testimony suggests that he was not an inventor of at least some of the subject matter defined by claims in the subject patent, USP9104842."* It is just that the testimony of Mr. Cooperman didn't serve the purpose of attaching him to the subject Patent. Additionally someone's refusal to cooperate does not alone indicate inventorship or preclude inventorship.

It is further unclear, even given the Patent Owner's reasoning, how the argued "proof" that the Patent Under reexamination is entitled to a date equal to that of WO97/26732 (1/17/1996), that this would throw out Hasebe et al. as prior art given Hasebe's claims to foreign priority bac to 8/31/1995. Patent Owner has not submitted

an appropriate oath or declaration to establish invention of the subject matter prior to the effective date of the reference (37 CFR 1.131).

I.2

With respect to arguments that “Beetcher497 and Beetcher072 reference do not disclose any claim”:

Beetcher teaches using the product # and version # used to build the template to further create the key, then utilizing this key to enable decoding of the code (see 9:1-60).

The process starts by placing **entitlement verification triggers** in the object code. The process involves creation of a program template that is input into a translator along with product numbers and version numbers. From this the encoded code is generated. (see column 9, lines 1-20)

Entitlement keys for accessing a subset of the encoded code are then generated using a product # and a version # that is accessible by the user. (see column 9, lines 21-48)

Then when a user desires access to the code they enter an entitlement key that is used in an unlock routine that handles decoding (rebuilding) the code, so that only permissible segments are executable. (see column 9, line 49 through column 10, line 39)

Patent Owner specifically argues that “The software modules are not encoded using the key.”

In response, the Examiner respectfully submits that the software modules are in fact created using information from the key, including version # and product #. (see column 9, lines 1-20)

Patent Owner specifically argues that “encrypting and decrypting the key is not encoding or decoding and software module...” and that “Beetcher does not disclose encoding or decoding of software using a key, or decoding license code encoded in software.”

In response, the Examiner respectfully submits that the described process starts by placing **entitlement verification triggers** in the object code. The process involves creation of a program template that is input into a translator along with product numbers and version numbers. From this the encoded code is generated, where this code is encoded with hidden functionality not available until it is decoded. (see column 9, lines 1-20) Subsequent the end device receiving the key the software code enters a “decoding process” whereby the table enabling access to software modules is rebuilt using key data (product numbers and version numbers) to enable execution of select modules. (see column 9, line 49 through column 10, line 39)

Moreover, during the original prosecution, Patent Owner specified that “[e]ncoding using a key and an algorithm is known.” (Ex. 2, Prosecution History at 519) As such, a POSITA would have understood that Beetcher’s encoding technique

necessarily includes a first license key and an encoding algorithm to form a first license key encoded software code (Silva Declaration at 70-75).

V. Specific arguments against Beetcher

V.1.1 - V.1.3

No specific arguments presented against the reference.

This section includes an important admission by the Patent Owner of what exactly Beetcher teaches. Specifically, Patent Owner notes that *“Beetcher discloses that set lock function 421, sets the version number in each entry in product lock table 460 based upon the entitlement flag and version number in entitlement key 111”* and that *“The entitlement flag and version number are data in the entitlement key 111. See fig. 2; version bits 202, entitlement flags 205. Thus, Beetcher discloses that the version data in product lock table 460 is derived only from the data in entitlement key 111.”*

V.1.4 - V.1.5

Though no specific arguments presented against the reference, this section includes a mischaracterization of the Beetcher reference. Specifically, the Patent Owner notes that *“Beetcher discloses the machine key is used to decrypt (decode) Beetcher’s encrypted entitlement key and is unrelated to and never applied to modify, encode, or decode, Beetcher’s software module.”* However, Beetcher notes use of “the

machine key to decode the entitlement key”, which is then used in the decoding process to unlock the software modules (see column 9, line 49 through column 10, line 39).

V.2 Arguments against Claim 11

V.2.1 - V.2.2

Patent Owner specifically argues that *“Beetcher explains that general input routine 441 is part of the operating system. USP5933497 col. 7:66 to 8:1 (“This operation system support at virtual machine level 404 contains two user interface routines needed to support input of the entitlement key.”) Beetcher explains that general input routine 441 handles input of entitlement keys, subsequent to installation of the operating system. USP5933497 col. 8:1-3. Therefore, other than for initial installation of Beetcher’s operating system, Beetcher does not disclose loading a software product on computer 101 in which the same software product uses an entitlement key 111 (licensing information) entered into a user interface in response to a prompt. In contrast, claim 11 requires “loading a software product on a computer, ... said software product using license information entered via said input in response to said prompt.” Instead, for all software loads subsequent-to-operating-system-installation, Beetcher discloses the operating system’s “[g]eneral input routine 441” using the license information.”*

In response, the Examiner respectfully submits that it is clear in Beetcher that the reference to “an initial installation” is that of installation of the software product / software module (see column 9, lines 1-61) not initial installation of the OS. That is

when the software which included encrypted portions that are only executable should the user supply an acceptable key is downloaded (initial installation) that is when a prompt is provided requesting said key. The software product contains verification triggers in its object code, where the corresponding program is unlocked via an unlock routine 430 that uses machine key to unlock an encrypted entitlement key in a key table, unlock routine 430 then rebuilds the encoded product key table 450. (see column 9, lines 1-61 and figure 9a) Additionally, products are unlocked on demand by unlock routine 430 by fetching the encrypted entitlement key from the appropriate entry in the encoded product key table 450, decodes the entitlement key, the key is then checked to confirm entitlement, and if so the program instructions are executed (see column 10, lines 20-39 and figure 9b).

V.2.3

Patent Owner specifically argues that “ *Beetcher does not teach "loading a software product" that then decodes a "license code encoded in said software product," as required by claim 11.*”

...

“Beetcher does not teach decoding a license code encoded in a software product. Beetcher does not disclose a license code encoded in a software product. Beetcher also does not disclose decoding a license code encoded in a software product. Instead, the only thing Beetcher teaches decoding, is the encrypted entitlement key 111. USP5933497 col. 10:27-31 ("Unlock routine 430 then fetches the encrypted entitlement key from the appropriate entry in encoded product key table 450 at step

921, obtains the machine key at step 922, and decodes the entitlement key at step 923.") Encrypted entitlement key 111 is not contained in the software product loaded onto the Beetcher's computer 101; that key is "separately distributed" from the software."

In response, the Examiner respectfully submits that as an initial matter the software module that is transmitted to the destination computer 101 operates (and includes code) that "receives, decodes and stores the entitlement key 111 and sets product lock table 460" (see column 8, lines 53-67). The entitlement triggers embedded within the software module, including the utilized product key table and product lock table act to take the key and use it to make the software modules executable program modules operational. Surely the entries in the product key table / product lock table that use the key to decode are considered license code encoded in the software product.

V.2.4

Patent Owner specifically argues that "the fact that the entitlement code is stored in computer system 101 in one or more tables, after the software is loaded on Beetcher's computer system 101, does not define a software program, as loaded onto computer system 101, encoding a license code."

In response, the Examiner respectfully submits that again the software module that is transmitted to the destination computer 101 operates (and includes code) that "receives, decodes and stores the entitlement key 111 and sets product lock table 460" (see column 8, lines 53-67). This code transmitted to the client 101 includes code that requires a key to operate. It is unclear how the Patent Owner doesn't view this a

license code that is decoded via the provided key (see column 9, lines 49-61). The Computer system 101 is said to “maintain product lock table 460” where content of the table is decoded via the entitlement key and the table is rebuilt right on computer system 101 (supra).

V.2.5

Patent Owner specifically argues that *“The Rexam request's next sentence at page 24: 12-14, states "Beetcher's Figures 4 and 9a, which are provided below, show the software using the key (i.e., license information) entered by the customer to decode a first license code encoded in the software product.”*

In response, that statement is also incorrect. Beetcher does disclose a license code encoded in the software product. Beetcher does not disclose decoding license code encoded in the software product.”

In response, the Examiner respectfully submits that the Patent Owner appear to be using the claim terms “license information” and “license code” synonymously. The claim however separately claims “license information” and “license code”. Since the terms are close the Examiner would like to further clarify how he is interpreting the claim terms.

License Information – which compares to Beetcher’s entitlement key - is a key entered by an individual at an end user device that allows a user access to a particular portion of a software product separately downloaded.

License code – which compares to Beetcher's software module - is code embedded within the downloaded software product that blocks usage of particular modules of that software product until a corresponding license information / entitlement key is entered granting access to the software module.

V.2.7

Patent Owner specifically argues that "The Rexam request admits that Beetcher's decoding refers to the decoding of encrypted entitlement key 111."

In response, the Examiner respectfully submits that though decoding occurs here as described, this is not the only decoding / decrypting occurring in Beetcher. Unlock routine is said to be what handles the decoding process, where unlock routine decodes the entitlement key and rebuilds encoded product key table (itself decoding) using the new key, which enables previously un-executable software modules. (See column 9, line 49 through column 10, line 39).

V.2.8

Patent Owner specifically argues that *"First, Beetcher's check lock function 422 is not part of Beetcher's software. Its microcode, that is, part of horizontal microcode 402. See USP5933497 Fig. 4 and col. 7:18-21 ("It [sic; horizontal microcode 402] is physically stored in control store 103, which in the preferred embodiment is a read-only memory (ROM) which is not capable of alteration by the customer."); and col. 8:21-22("horizontal microcode check lock function 422."). Therefore, check lock function 422 is not part of a software product, and certainly not part of the software products*

Beetcher explains is distributed by Beetcher's distributor to Beetcher's computer system 101.

Second, Beetcher does not disclose that check lock function 422 decodes a first license code encoded in a software product. Beetcher discloses that check lock function 422 reads an entry in product lock table 460 to verify entitlement. ("Check lock function 422 accesses product lock table 460 and reads one of the entries to verify entitlement.") Reading and verifying are not decoding."

In response, the Examiner respectfully submits that: First Beetcher's lock function is part of the downloaded software, as the software module that is transmitted to the destination computer 101 operates to and includes code that "receives, decodes and stores the entitlement key 111 and sets product lock table 460" (see column 8, lines 53-67). Second, license information is used while checking lock, license information (the key) is input to the product key table (from the software module) to reveal encrypted entitlement key where if entitlement is indicated unlock is granted via the product lock table (see column 9, line 49 through column 10, line 39).

V.2.8 – V.2.9

Patent Owner specifically argues that *"Beetcher col. 10:52-62 does not disclose changing the form of entitlement verification triggering instruction 301. Therefore, Beetcher col. 10:52-62 does not disclose decoding of entitlement verification triggering instruction 301."*

...

"Claim 11, in contrast, requires "said software product using... a routine designed to decode a first license code encoded in said software product." Beetcher discloses no routine to decode a first license code encoded in said software product. Accordingly, Beetcher does not disclose claim 11."

In response, the Examiner respectfully submits that the claims only requires "decode a first license code", this license code is not connected to the remainder of the claim except for being said to be "encoded in said software product". The software product includes code that "receives, decodes and stores the entitlement key 111 and sets product lock table 460" (see column 8, lines 53-67). The decoding involves using the key to access the original code by removing the encoded restrictions on access to verification trigger blocked portions of the code.

Examiner notes that a POSITA would define to ENCODE in computer art is to convert into a particular form (usually for transmission or security). Conversely to DECODE something that is encoded is to convert (revert) to the original (or ordinary) form. These definitions are in line with Dictionary.com definitions. Beetcher clearly teaches encoding in its taking of original code, insertion of entitlement verification triggers in code, and precluding using the original code until a key is provided, decoded, a table is rebuilt again making the code executable (see column 9, lines 1-20 and column 4, lines 3-46).

The compiler begins the process by producing a template (step 702), next the template is input into the translator (step 703), then the translator encodes the

triggers/license keys into the code (step 704), and finally the translator resolves references after key insertion to produce the executable module. As such, a POSITA would have understood Beetcher's Figure 7 illustrates an encoding algorithm.

Moreover, during the original prosecution, Patent Owner specified that "[e]ncoding using a key and an algorithm is known." (see Exhibit 2, Prosecution History at 519). A POSITA would have understood that Beetcher's encoding technique necessarily includes a first license key and an encoding algorithm to form a first license key encoded software code.

V.3 – V.3.1

Patent Owner specifically argues that *"The Rexam request pages 35-39 imply that "entitlement verification triggering instructions 301" correspond to claim 12's "license key" and that Beetcher Fig. 3's software module 300 is license key encoded software.*

In response, Beetcher does not disclose Claim 12's "encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key encoded software code. This is because Beetcher's "entitlement verification triggering instructions 301" is not a "license key." Instead, it's a lock. It is what prevents the functioning of Beetcher's software modules, unless and until a customer enters Beetcher's "entitlement key 111."

In response, the Examiner respectfully submits that the Request rather shows encoding the software code utilizing information from the key encoded as triggers, within the encoded software code. This code is the original code encoded so as to preclude usage of modules until a proper entitlement key is provided to unlock the original code (see column 9, lines 1-20 and column 4, lines 3-46).

Patent Owner specifically argues that *"The specification continues, stating "... 3) Once it has the license code, it can then generate the proper decoding key to access the essential code resources." USP9104842 Col. 13:65-67; USP5745569 col. 6:29-32. This means that the application can function as intended only after it has the decoding key which it derives from the license code or key. The key unlocks the software. "*

...

"The Rexam's request of equating that which locks the software so it cannot run (verification triggering instructions 301) with a license key, which is something that enables software to run, is contrary to both the specification of the subject patent, and the meaning of a license key in the computer arts, and inconsistent."

In response, the Examiner respectfully submits that like the Patent Beetcher teaches using license code information (the version number and product number) to generate the both the license key (enablement key) and the license key encoded software (software code encoded with key based verification triggers embedded within the software code) (see column 9, lines 1-48).

Patent Owner specifically argues that *"The reexamination request's argument is inconsistent because Beetcher does not disclose the installed version of software module 300 receiving an entitlement verification triggering instructions 301, and even if it did, that would not unlock Beetcher's software"*.

In response, the Examiner respectfully submits that this characterization is inconsistent with the rejection as the rejection makes clear that it is the received entitlement key that is used to unlock corresponding verification triggered sections of the encoded software module (see column 9, line 1 through column 10, line 39).

V.4 – V4.1

Patent Owner specifically argues that *"The Rexam request at pages 40-42 incorrectly asserts that Beetcher discloses claim 13's "modifying, by said computer, using a first license key and an encoding algorithm, said software code, to form a modified software code," arguing that Beetcher's "entitlement verification triggering instructions 301" correspond to the claimed "first license key." See Rexam request, page 40, last two lines, stating "Beetcher's distributor system includes a computer that encodes software code using a first license key (e.g., triggering information)."*

In response, the Examiner respectfully submits that again stating that *"Beetcher's distributor system includes a computer that encodes software code using a first license key (e.g., triggering information)" does not imply that a **first license key is trigger information** but rather that **software code that is encoded using a first license key results in triggering information embedded in that now encoded software code.***

V.4.2

Patent Owner specifically argues that *"The Rexam request at page 43-44 incorrectly alleges that Beetcher discloses "wherein said modified software code comprises said encoded first code resource, and a decode resource for decoding said encoded first code resource." The dashed lines contain the following and only the following elements from Fig. 4: "Unlock (Decode key) 430"; "Exception handler 432"; and "Check Lock 422."*

In response, this assertion is incorrect.

First, the Rexam request provides no explanation how those element meet the claim limitation.

Second, the Rexam request fails to account for the requirement in the claim recitation that the "modified software code comprises .. a decode resource for decoding said encoded first code resource;" The Rexam request fails to identify, and Beetcher does not disclose, the items in this dashed box functioning to decode anything in Beetcher's software module.

Third, Beetcher fails to disclose decoding anything in Beetcher's installed software module.

Fourth, "Check Lock 422" is not part of Beetcher's software module. Beetcher's "Check Lock 422" could not possibly be part of Beetcher's software module, because it is part of the microcode level 402, Fig. 4. And the microcode is part of the hardware level, in this case, embodied in ROM, and therefore not possibly part of Beetcher's installable software module."

In response, the Examiner respectfully submits that:

First, the Reexam request explains that the delivered software module is encoded to include verification triggers based upon key information. Then upon being supplied with an enablement key the encoded software is decoded to reveal the original code in its original executable form. The encoded software module is transmitted along with a product lock table showing correspondence between keys and the sub-modules they unlock (see column 8, lines 53-67), where the Request describes this unlock process and what resident elements carry it out along with the transmitted table.

Second, the Reexam request shows the modified code (software module) including a decode resource (product lock table) for decoding said encoded first code resource (supra), via the described hardware.

Third, Beetcher discloses decoding anything in the software module (supra).

Fourth, "Check Lock 422" is not argued to be part of Beetcher's software module, just a means for implementing it.

Patent Owner specifically argues that *"Beetcher does not disclose a decode resource configured to decode an encoded first code resource upon receipt of a license key"*

In response, the Examiner respectfully submits that the software product includes code that "receives, decodes and stores the entitlement key 111 and sets product lock table 460" (see column 8, lines 53-67). The decoding of involves using the key to access the original code by removing the encoded restrictions on access to verification trigger blocked portions of the code.

Examiner notes that a POSITA would define to ENCODE in computer art is to convert into a particular form (usually for transmission or security). Conversely to DECODE something that is encoded is to convert (revert) to the original (or ordinary) form. These definitions are in line with Dictionary.com definitions. Beetcher clearly teach encoding in its taking of original code, insertion of entitlement verification triggers in code, and precluding using the original code until a key is provided (see column 9, lines 1-20 and column 4, lines 3-46).

The compiler begins the process by producing a template (step 702), next the template is input into the translator (step 703), then the translator encodes the triggers/license keys into the code (step 704), and finally the translator resolves references after key insertion to produce the executable module. As such, a POSITA would have understood Beetcher's Figure 7 illustrates an encoding algorithm.

Moreover, during the original prosecution, Patent Owner specified that "[e]ncoding using a key and an algorithm is known." (see Exhibit 2, Prosecution History at 519). A POSITA would have understood that Beetcher's encoding technique necessarily includes a first license key and an encoding algorithm to form a first license key encoded software code.

V.5

Patent Owner specifically argues that the Request incorporated the rejection from 12.3 but left out a limitation specifically *“Rexam request section 12.3 dealt with the claim recitation “encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key encoded software code,” in claim 12. This claim 12 recitation does not also require encoding of “software code interrelationships,” which is required by claim 14.”*

In response, the Examiner respectfully submits that it is clear from the request that software code interrelationships exist many ways in Beetcher, an encoded entitlement key relates to a decoded entitlement key through use of machine key (see column 9, lines 49-60), a key table shows association between key and executable code (see column 9, line 60 through column 10, line 19), triggers are placed in code along with corresponding code the key triggers (see column 9, lines 1-20), a product lock table when supplied with a key unlocks a corresponding code for execution (see column 10, lines 20-47), etc. The broadly worded claims that use different claim wordings to essentially claim the same concept lend themselves to multiple options for which a reference can cover the claim.

Litigation Reminder

The Patent Owner is reminded of the continuing responsibility under 37 CFR 1.565(a) to apprise the Office of any litigation activity, or other prior or concurrent proceeding, involving Patent Number: 9,104,842 throughout the course of this reexamination proceeding. The third part requester is also reminded of the ability to

similarly apprise the Office of any such activity or proceeding throughout the course of this reexamination proceeding. See MPEP §§ 2207, 2282 and 2286.

Service of Papers

After filing of a request for ex parte reexamination by a third party requester, any document filed by either the patent owner or the third party requester must be served on the other party (or parties where two or more third party requester proceedings are merged) in the reexamination proceeding in the manner provided in 37 CFR 1.248. The document must reflect service or the document may be refused consideration by the Office. See 37 CFR 1.550(f).

Response to this Action

In order to ensure full consideration of any amendments, affidavits, or declarations, or other document as evidence of patentability, such documents must be submitted in response to this Office Action. Submissions after the next Office Action, which is intended to be a Final Action, will be governed by the requirements of 37 CFR 1.116, after final rejection and 37 CFR 41.33 after appeal, which will be strictly enforced.

Conclusion

Extensions of time under 37 CFR 1.136(a) do not apply in reexamination proceedings. The provisions of 37 CFR 1.136 apply only to "an applicant" and not to parties in a reexamination proceeding. Further, in 35 U.S.C. 305 and in 37 CFR

1.550(a), it is required that reexamination proceedings "will be conducted with special dispatch within the Office."

The patent owner is reminded of the continuing responsibility under 37 CFR 1.565(a) to apprise the Office of any litigation activity, or other prior or concurrent proceeding, involving the Millington patent throughout the course of this reexamination proceeding. The requester is also reminded of the ability to similarly appraise the Office of any such activity or proceeding throughout the course of this reexamination proceeding. See MPEP § § 2207, 2282, and 2286.

All correspondence relating to this *ex parte* reexamination proceeding should be directed:

By Mail to: Mail Stop Ex Parte Reexam
Central Reexamination Unit
Commissioner for Patents
United States Patent & Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450

By FAX to: (571) 273-9900
Central Reexamination Unit

By hand: Customer Service Window

Randolph Building
401 Dulany Street
Alexandria, VA 22314

By EFS-Web:

Registered users of EFS-Web may alternatively submit such correspondence via the electronic filing system EFS-Web, at

EFS-Web offers the benefit of quick submission to the particular area of the Office that needs to act on the correspondence. Also, EFS-Web submissions are "soft scanned" (i.e., electronically uploaded) directly into the official file for the reexamination proceeding, which offers parties the opportunity to review the content of their submissions after the "soft scanning" process is complete.


Any inquiry concerning this communication or earlier communications from the Reexamination Legal Advisor or Examiner, or as to the status of this proceeding, should be directed to the Central Reexamination Unit at telephone number (571) 272-7705.

/DENNIS G BONSHOCK/
Primary Examiner, Art Unit 3992

Conferee:

/William H. Wood/
Primary Examiner, Art Unit 3992

/ALEXANDER J KOSOWSKI/
Supervisory Patent Examiner, Art Unit 3992

<i>Search Notes</i> 	Application/Control No. 90/014,138	Applicant(s)/Patent Under Reexamination 9104842
	Examiner DENNIS G BONSHOCK	Art Unit 3992

CPC - Searched*		
Symbol	Date	Examiner

CPC Combination Sets - Searched*		
Symbol	Date	Examiner


US Classification - Searched*			
Class	Subclass	Date	Examiner

* See search history printout included with this form or the SEARCH NOTES box below to determine the scope of the search.

Search Notes		
Search Notes	Date	Examiner
Previous Prosecution Searched	12/6/2018	DGB
litigation Searched	12/6/2018	DGB

Interference Search			
US Class/CPC Symbol	US Subclass/CPC Group	Date	Examiner

--	--

Reexamination 	Application/Control No. 90/014,138	Applicant(s)/Patent Under Reexamination 9104842
	Certificate Date	Certificate Number

Requester Correspondence Address: Patent Owner Third Party

Joseph F. Edell
 Fisch Sigler LLP
 5301 Wisconsin Ave, NW
 Fourth Floor
 Washington, DC 20015

LITIGATION REVIEW <input checked="" type="checkbox"/>	/DGB/ (examiner initials)	2018-06-13T00:00:00 (date)
Case Name		Director Initials
6:17cv16		
6:18cv174		
6:18cv181		
6:18cv195		
6:18cv223		

COPENDING OFFICE PROCEEDINGS	
TYPE OF PROCEEDING	NUMBER

--	--

Reexamination control number: 90014138

Confirmation number: 7638

RE: United States patent 9104842

37 CFR 1.530 Patent Owner Response
to the Non Final Office Action Dated 12/12/2028

Table of Contents

I.	Evidence Previously submitted	Page 1 of 33
II.	Evidence Submitted with this Response	Page 1 of 33
III.	Summary of Office Action	Page 2 of 33
III.1	Summary of Rejections	Page 2 of 33
III.2	Relevant Dates of References Relied Upon in Rejections	Page 2 of 33
IV.	Summary of Response To Rejections	Page 3 of 33
IV.1	Summary as to Beetcher	Page 3 of 33
II.2	Summary as to Cooperman and Hasebe	Page 3 of 33
V.	Mooting Anticipated Rejections	Page 3 of 33
VI.	The Beetcher Rejections Should be Withdrawn Because They are Improperly Vague	Page 4 of 33
VII.1	The Beetcher Rejections, Whatever Their Basis, are Wrong, and Should be Withdrawn	Page 4 of 33
VII.2	Rejected Claim 11	Page 4 of 33
VII.2.1	Description of Beetcher's Unencrypted entitlement key 200	Page 4 of 33
VII.2.2	Description of How Beetcher's Marketing System 124 and Customer Computer 101 Use Beetcher's UnEncrypted Entitlement Key 200 and Encrypted Entitlement Key 111	Page 5 of 33
VII.2.3	Beetcher's Entitlement Key is Not Part of Beetcher's Software Module	Page 6 of 33
VII.2.4	Beetcher's Entitlement Key is Not Used to Decode Any Instruction In Beetcher's Software Module	Page 6 of 33
VII.2.5	Description of Beetcher's entitlement verification instruction 301	Page 6 of 33
VII.2.6	Description of How Customer Computer 101 Uses Beetcher's entitlement verification instruction 301	Page 7 of 33
VII.2.7	Beetcher Discloses that Check lock function 422 compares data in entry 601 in product lock table 460 to the product number contained in the Product Number field 304 in instruction 301	Page 7 of 33
VII.2.8	Beetcher's Customer Computer 101 Does Not Decode Beetcher's entitlement verification instruction 301	Page 8 of 33
VII.2.9	Beetcher's Customer Computer 101 Does Not Have a Routine to Decode Beetcher's entitlement verification instruction 301	Page 9 of 33
VII.2.10	Beetcher's Customer Computer 101 Does Not Use Information from the Encrypted Entitlement Key 111 to Decode Beetcher's entitlement verification instruction 301	Page 9 of 33
VII.3	Claim 11: "loading a software program ... said software product outputting a prompt for input of license information; and "	Page 9 of 33
VII.3.1	Beetcher Discloses Customer Computer 101 Prompts for Input of Encrypted Entitlement Key 111	Page 9 of 33
VII.3.2	Claim 11 defines a process in which the licensing information input in response to the prompt is not part of the loaded software product	Page 10 of 33
VII.3.3	Beetcher's Encrypted Entitlement Key 111 Is Input In Response to a Prompt, Does Not Correspond to Claim 11's Installed Software Product	Page 10 of 33
VII.4	Claim construction, Claim 11: " <i>encode ... in a routine designed to decode</i> "	

.....	<u>Page 11 of 33</u>
VII.5 Response to the NFOA’s Assertions Regarding Beetcher	<u>Page 15 of 33</u>
claim 11's "said software product using license information entered via said input in response to said prompt <i>in a routine designed to decode</i> a first license code <i>encoded</i> in said software product.	<u>Page 15 of 33</u>
VII.5.1 Contrary to the NFOA, Beetcher <i>does not</i> disclose using the entitlement key (111, 200; encrypted, unencrypted) in a routine designed to decode anything in Beetcher's software, as required by claim 11	<u>Page 15 of 33</u>
VII.5.2 Contrary to the NFOA, Beetcher <i>also does not</i> disclose using the entitlement key in a routine designed to decode entitlement verification triggering instruction 301; does not disclose a routine designed to decode instruction 301; and does not disclose decoding instruction 301, all of which are required by claim 11	<u>Page 16 of 33</u>
VII.5.2.1 Contrary to the NFOA, Beetcher Does Not Disclose Using the Key’s Version and Product Number to Decode a License Code	<u>Page 17 of 33</u>
III.5.2.2 Entitlement verification triggering instructions	<u>Page 17 of 33</u>
VII.5.2.3 Triggering Instruction 301	<u>Page 17 of 33</u>
VII.5.2.4 Key table 460	<u>Page 18 of 33</u>
VII.5.2.5 Check Lock Function 422	<u>Page 18 of 33</u>
VII.5.2.6 Execution of Instruction 301	<u>Page 20 of 33</u>
VII.7 Contrary to the NFOA, Patent Owner’s Description of Beetcher in the Patent Owner’s Statement, Was Accurate	<u>Page 21 of 33</u>
VII.8. Claim 12 Claim Construction, “encoding using at least a first license key and an encoding algorithm”	<u>Page 23 of 33</u>
VII.9 Beetcher does not disclose “encoding using at least a first license key and an encoding algorithm.”	<u>Page 24 of 33</u>
VII.10 Claim 12 Claim construction, “only after receipt of said first license key.”	<u>Page 25 of 33</u>
VII.11 Beetcher Does not Disclose Claim 12s “only after receipt of said first license key.”	<u>Page 25 of 33</u>
VII.12 Claim 12 Claim Construction “first license key”	<u>Page 25 of 33</u>
VII.13 Beetcher Does Not Disclose Instruction 301 is a “first license key,” as Defined by Claim 12	<u>Page 26 of 33</u>
VII.14 Claim 13	<u>Page 26 of 33</u>
VII.15 Contrary to the NFOA, Beetcher does not disclose Claim 13’s “wherein said modified software code comprises said encoded first code resource, and a decode resource for decoding said encoded first code resource”	<u>Page 26 of 33</u>
VIII.16 Contrary to the NFOA, Beetcher Does Not Anticipate Claim 14 ..	<u>Page 29 of 33</u>
Further Comments	<u>Page 30 of 33</u>
XI.1 Regarding the NFOA’s Understanding of Beetcher	<u>Page 30 of 33</u>
XI.2 Regarding Cooperman	<u>Page 31 of 33</u>
XI.3 Regarding Mr. Marc Cooperman	<u>Page 31 of 33</u>
XI.4 Regarding Hasebe’s Prior Art Date	<u>Page 32 of 33</u>

I. Evidence Previously submitted

Petition for Reexamination, Exhibits 1-11:

Exhibit 1, US Patent No. 9,104,842 to Moskowitz

Exhibit 2, Prosecution History of the '842 Patent

Exhibit 3, US Patent No. 5,933,497 ("Beetcher")

Exhibit 4, Japanese Patent Application Publication No. H05334072 ("Beetcher '072")

Exhibit 5, English Translation of Beetcher '072

Exhibit 6, PCT Application Publication No. WO 97126732 ("Cooperman")

Exhibit 7, US Patent No. 5,935,243 ("Hasebe")

Exhibit 8, None

Exhibit 9, Declaration of Dr. Claudio Silva ("Silva Declaration")

Exhibit 10, Curriculum Vitae of Dr. Silva

Exhibit 11, Plaintiff Blue Spike LLC's Proposed Terms for Construction, Pursuant to Patent Rule (P.R.) 4-2 in Blue Spike, LLC v. Juniper Networks, Inc., Case No. 6:17-cv-16-KNM (E.D. Tex.)

Patent Owner Statement, Attachments 1-10:

Attachment 1, Pages 1-28 from the file history of application 08/587,793 (issued as USP5745569) (27 Pages)

Attachment 2, USP5745569 (6 pages)

Attachment 3, Pages 1-12 and 172-192 of the transcript of the deposition of Marc S. Cooperman, May 17, 2018, in Blue Spike LLC v. Juniper Networks, Inc., civil case 6:17-cv-00016-KNM. (33 Pages)

Attachment 4, Comparison of Disclosures of US application 08/587,943; WO 97/26732; USP5,745,569; USP9021602; and USP9104842. (5 pages)

Attachment 7, 2002 Settlement Agreement (37 pages)

Attachment 9, Title Abstract for application, 08587943, now USP5745569. (2 Pages)

Attachment 10, a claim support chart showing support in 08587943 filed 1/17/1996; USP5745569; and USP9104842 for the citations used by the reexam request to show disclosure of the claimed subject matter by the Cooperman reference. (6 pages)

The exhibits and attachments references above and the patent owner response are relied upon in this response and are incorporated by reference herein.

II. Evidence Submitted with this Response

Attachment 11, a copy of the screen image showing the Google search query and quote definitions.

Attachment 12, the definition of encoding and decoding, at <https://searchnetworking.techtarget.com/definition/encoding-and-decoding>.

Attachment 13, the definition of "Object Code" from the Technopeida website.

Attachment 14, the definition of "routine" from the Webopedia website.

Attachment 15, definition of "microcode" from Wikipedia.

Attachment 16, 37 CFR 1.131 declaration of Scott Moskowitz
Attachment 17, 37 CFR 1.132 declaration of Scott Moskowitz
Attachment 23, SM_MC_1993Agreement.pdf Agreement between Marc Cooperman and Scott Moskowitz
Attachment 24, Page1_CoopermanLetter_11-18-1996.pdf Letter from Marc Cooperman to Scott Moskowitz
Attachment 25, 11-15-95EmailDisclosure.pdf Email dated 11/15/1995 from Marc Cooperman to Scott Moskowitz
Attachment 26, DraftApplicationDated12-22-95.pdf Draft patent application dated 12/22/1995.
Attachment 27, DraftApplicationDated01-03-96.pdf Draft patent application dated 1/3/1996.

III. Summary of Office Action

The Non Final Office Action Dated 12/12/2028 (NFOA) stated that claims 1-10 were not subject to reexamination; that claims 11-14 were subject to reexamination; and rejected claims 11-14. NFOA PTOL-466, Summary page.

III.1 Summary of Rejections

The NFOA, page 6, rejects claims 11-14 under 102(a) based upon Beetcher.

The NFOA, page 47, rejects claims 11, 12, and 13 under 102(e) based upon Cooperman.

The NFOA, page 69, rejects claims 12, 13, 14, under 102(e), based upon Hasebe.

III.2 Relevant Dates of References Relied Upon in Rejections

The NFOA, page 2, identifies the applied references. These references and their relevant dates are as follows.

USP 5933497 to Beetcher (Beetcher, Exhibit 3) issued 8/3/1999, from application 08/011,042, filed 1/29/1993. Ex 3, page 1. Beetcher, Exhibit 3 has a 102(a) date of 8/3/1999. Beetcher has a 102(e) prior art date of 1/29/1993. **Beetcher has a 102(b) date of 8/3/2000.**

WO97/26732 to Cooperman (Cooperman, Exhibit 6) published 7/24/1997, from PCT application PCT/US97/00651, filed 1/16/1997. Cooperman, Exhibit 6 claims Paris priority to US application 08/587,943, filed 1/17/1996. Ex. 6, page 1. See MPEP706.02(f)(1)(C)(3)(b) ("For U.S. application publications and WIPO publications directly resulting from international applications under PCT Article 21(2), never apply these references under pre-AIA 35 U.S.C. 102(e). These references may be applied as of their publication dates under pre-AIA 35 U.S.C. 102(a) or pre-AIA 35 U.S.C. 102(b);"). Thus, Cooperman, Exhibit 6 has no 102(e) date.

Cooperman, Exhibit 6 has a 102(a) date of 7/24/1997.

USP 5935243 to Hasebe (Hasebe, Exhibit 7) issued 8/10/1999, from application 08/673,108, filed 7/1/1996. Hasebe, Exhibit 7 claims Paris priority to Japanese patent application 7-224338, filed 8/31/1995. Hasebe, page 1. Paris priority claims do not qualify as prior art under pre-AIA 35 USC 102, except under the narrow circumstance of a 102(g) interference proceeding, per the *Hilmer* doctrine. See "Viability of the Hilmer Doctrine" Rick Neifeld, JPTOS, 81 (July 1999), 544, also available at URL: https://www.neifeld.com/pubs/2000_newpatentlaws.html. Hasebe has a 102(a) date of 8/10/1999. **Hasebe has a 102(e) date of 7/1/1996.**

The prior art dates for these references are summarized in Table 1.

Table 1

Reference	102(b)	102(a) date	102(e) date
USP 5933497 to Beetcher (Beetcher, Exhibit 3)	YES	8/3/1999	1/29/1993
WO97/26732 to Cooperman (Cooperman, Exhibit 6)	NO	7/24/1997	None.
USP 5935243 to Hasebe (Hasebe, Exhibit 7)	NO	8/10/1999	7/1/1996

IV. Summary of Response To Rejections

IV.1 Summary as to Beetcher

The rejections of claims 11-14 under 102(a) based upon Beetcher should be withdrawn because Beetcher does not anticipate any claim. The examiner erred in concluding that Beetcher discloses decoding of a license code encoded in an installed software product. Instead, Beetcher disclose decrypting a *separately provided* encrypted key, and sequentially *fetching instructions* from the object of the installed software. As explained below, neither of those actions corresponds to the claimed decoding of a license code encoded in an installed software product.

II.2 Summary as to Cooperman and Hasebe

The rejections of claims 11, 12, and 13, under 102(e), based upon Cooperman, should be withdrawn because Cooperman has no 102(e) date. MPEP706.02(f)(1)(C)(3)(b); see discussion above.

The rejections of claims 12, 13, 14, under 102(e), based upon Hasebe, should be withdrawn because Hasebe's 102(a) and 102(e) dates are after the date of invention of the rejected claims. Mr. Moskowitz's 37 CFR 1.131 declaration submitted with this response shows that he and Mr. Cooperman primarily constructively reduced to practice inventions, including those defined by claims 11-14, by filing the disclosure in 08/587,943, filed Jan. 17, 1996, which issued 4/28/1998, as USP5745943. Mr. Moskowitz's notes show additional, pre-filing work on the inventions. These show reduction to practice prior to the 102 dates of Hasebe. Accordingly, the rejections relying upon Hasebe are improper.

Moreover, rejected claims are entitled under 35 USC 120 to the benefit of the filing date of 3/24/1998, and therefore neither Cooperman nor Hasebe is a 102(b) statutory bar.

V. Mooting Anticipated Rejections

Rejection based upon Cooperman under 102(a) would be improper because of the proof submitted herewith of invention of claims 11-14, prior to Cooperman's publication date.

Rejection based upon USP5745943 under 102(a) or (e) would be improper because of the proof submitted herewith of invention of claims 11-14, prior to USP5745943's filing date.

Rejection based upon USP5745943 under 102(f) would be improper because the fact that Mr. Cooperman was a named inventor on USP5745943 does not make a *prima facie* case that Mr. Moskowitz is not the sole inventor of USP 9104842's claims 11-14. *Cf. Aktiebolaget Karlstads Mekaniska Werkstad v. ITC*, 82-21, 705 F. 2d 1565, 1574 (Fed. Cir. 4/18/1983)("there is no presumption ... to assume, that everything disclosed in a patent specification has been invented by the patentee"); and *In re DeBaun*, 82-530, 687 F. 2d 459, 463 (CCPA 1982)("The '678 patent is silent with respect to who invented the basic equalizer honeycomb section itself,

and we do not presume that it is the invention of appellant and Noll jointly or of either of them. *** the board erred in upholding the rejection based on the '678 patent in view of appellant's showing that the basic equalizer honeycomb section is appellant's own invention.").

VI. The Beetcher Rejections Should be Withdrawn Because They are Improperly Vague

The NFOA's rejections based upon Beetcher do not explain the rejection. All they do is state "here is a bunch of stuff in Beetcher" in subsections titled with quotes from the rejected claims. This fails to identify a correspondence of disclosure in Beetcher to claim limitations. That is insufficient to explain the basis for the rejection. Merely parroting the statements and conclusions in the reexamination request, without explanation, is insufficient to provide notice of the basis for the Beetcher rejections. Attachment 17, 37 CFR 1.132 Declaration of Scott Moskowitz (herein after 132Dec)¶3.

The NFOA in this regard parallels the biased litigation work product in the request for reexamination. There are in fact block quotes restated in the NFOA that are taken, without attribution, from the request for reexamination, and at least one such block quote is an inaccurate rendition of text in the request for reexamination. 132Dec¶4

The patentee has had to speculate what might be the basis for rejection. If the examiner maintains that the claims are anticipated by Beetcher, then the examiner should impose a new **non-final** rejection that explains the basis for the rejections with clarity, expressly identifying what correspondence exists between claim limitations, and disclosure in Beetcher. 132Dec¶5

VII.1 The Beetcher Rejections, Whatever Their Basis, are Wrong, and Should be Withdrawn

It's not clear from the NFOA why the claims are rejected based upon Beetcher. However, no correspondence of claimed limitations to Beetcher's disclosures results in anticipation of claim 11. Claim 11 is shown below, followed by description of Beetcher's unencrypted entitlement key 200, encrypted entitlement key 111, and entitlement verification instruction 301, and Beetcher's disclosure how they are used. 132Dec¶6

VII.2 Rejected Claim 11

Claim 11 reads:

11. A method for licensed software use, the method comprising:
loading a software product **on a computer**, said computer comprising a processor, memory, an input, and an output, so that said computer is programmed to execute said software product;
said software product outputting a prompt for input of license information;
and
said software product using license information **entered via said input in response to** said prompt **in a routine designed to decode a first license code encoded in said software product**. 132Dec¶7

VII.2.1 Description of Beetcher's Unencrypted entitlement key 200

What does Beetcher Fig. 2 show? Beetcher col. 4:52-53 states that "FIG. 2 shows the unencrypted contents of an entitlement key according to the preferred embodiment of this invention." Beetcher's Fig. 2 shows entitlement key 200 including 128 bits in five enumerated fields, 201 to 205. Fields 201 to 205 are described in the paragraph at Beetcher col. 6:20-40, which state that there are: 4 bits defining a charge group field 201; 8 bits defining software version field 202; 8 bits defining key type field 203; 28 bits defining machine serial number field 204; and 80 bits defining product entitlement flags 205. 132Dec¶8, 9

Beetcher col. 6:36-40 specifies that entitlement flags 205 each correspond to a product number, and each of those flags is set to "1" if the product number is entitled to run (aka licensed for use on that computer) and otherwise set to "0":

...Product entitlement flag 205 is an 80-bit field containing 80 separate product flags, each corresponding to a product number. The bit is set to '1' if the corresponding product number is entitled; otherwise it is set to "0". 132Dec¶10

Beetcher also discloses that the marketing system 124 generates the entitlement key 200 specific to a customer's order, including the serial number 105, 410 of the customer's computer, and generates encrypted entitlement key 111 from entitlement key 200, such that the serial number 105 is required to decrypt encrypted entitlement key 200. Beetcher's paragraph at col. 9:21-47 describes how Beetcher's entitlement key is generated and then encrypted. Beetcher discloses that the marketing system 124, Fig. 1, receives a customer order, and a key generator/encryptor 122 on marketing system 124 retrieves information about the customer from its database 123, and generates the unencrypted entitlement key 200. Beetcher col.8:22-36. 132Dec¶11

Beetcher discloses unencrypted entitlement key 200 includes the serial number of the customer computer 101 stored in the customer computer. Abstract ("The key includes the serial number of the computer for which the software is licensed"); col. 4:6-7 ("The key includes the serial number of the machine for which the software is licensed"); which customer computer 101 uses to decrypt encrypted entitlement key 111, see col. 4:9-13 ("The decryption mechanism fetches the machine serial number and uses it as a key to decrypt the entitlement key.") 132Dec¶12

VII.2.2 Description of How Beetcher's Marketing System 124 and Customer Computer 101 Use Beetcher's UnEncrypted Entitlement Key 200 and Encrypted Entitlement Key 111

Beetcher col. 8:39-40 then explains that "Key generator/encryptor 122 then encrypts the key," forming encrypted entitlement key 111. Beetcher col. 8:42-43 then explains that "The resultant encrypted entitlement key 111 is then transmitted to the customer at step 805." 132Dec¶13

Beetcher's paragraph at col. 9: 49 to 10:19 explains that Beetcher's customer/user's computer (Fig. 1, element 101) decrypts the entered and encrypted entitlement key 111, and stores entitlement data in memory. Beetcher col. 9:51-52 discloses that the customer enters encrypted entitlement key 111 into computer system 101. Beetcher col. 9:55-60 discloses that unlock routine 430 uses the serial number of computer 101 to "decode" [sic; decrypt] encrypted entitlement key 111. Beetcher col. 9:56 to 10:39 then describes the process of storing new

unencrypted entitlement key data in product key table 450. Beetcher col. 9:67 to 10:2 discloses this process includes determining if the decoded [sic; decrypted] entitlement flag for a product is "1". If so, Beetcher col. 10:3-8 discloses that unlock routine 430 updates entry 501 of product key table 450 with the "version number, and charge group value" contained in the entitlement key; and sets entitlement bit field 506 of entry 501 of product key table 450 (Fig. 5) to "1."132Dec¶14

Beetcher col. 10: then discloses that, if the entitlement flag for a product is "0", unlock routing 430 also sets the version number in product lock table 460 to "0" (indicating no entitlement). 132Dec¶15

Finally, Beetcher col. 10:18-19 disclose that, once table 450 is rebuilt, its contents are "saved in storage at step 909." Presumably, this reference to storage refers to the "plurality of storage devices 106, 107, 108." attached to the customer computer system 101 shown in Fig. 1 and identified at col. 5:21-22. Presumably, this statement refers to the transmission of table 450 from RAM 101 of the customer computer 101 to one of these storage devices, so that table 450 will survive reinitialization of the customer's computer. This presumption is in view of Beetcher's discussion at col. 8:23-27 that product key table 450 is contained in RAM 104, but duplicated in a non-volatile storage device so it can be recovered in case the computer system is powered down. It is well known in the art that data in RAM is lost when the RAM is powered down, and that RAM is powered down upon re-initialization of a computer system. 132Dec¶16

In summary, Beetcher discloses that Beetcher's customer computer system 101 receives encrypted entitlement key 111, decrypts the key using the customer computer 101's hard coded machine identification, and saves entitlement data in memory in records in tables 450 and 460. 132Dec¶17

VII.2.3 Beetcher's Entitlement Key is Not Part of Beetcher's Software Module

Beetcher contains no disclosure indicating that Beetcher's entitlement key is part of Beetcher's installed software module. Beetcher notes that the encrypted key is sent separately from the software modules, from the marketing system 124 to the customer computer 101. Abstract and col. 4:4-5 ("separately distributed encrypted entitlement key"). Beetcher Fig. 3 and col. 6:41-43 discloses that the software modules are "compiled object code". Beetcher's entitlement key 111, 200 is not object code. 132Dec¶18

VII.2.4 Beetcher's Entitlement Key is Not Used to Decode Any Instruction In Beetcher's Software Module

Beetcher contains no disclosure indicating that Beetcher's entitlement key, or any data stored in customer computer 101 derived from Beetcher's entitlement key, is used to decode an instruction in Beetcher's object code. Beetcher does not disclose that its instruction 301 is encrypted or encoded using Beetcher's entitlement key. Therefore, there would be no basis to decrypt or decode instruction 301 using Beetcher's entitlement key. 132Dec¶19

VII.2.5 Description of Beetcher's entitlement verification instruction 301

Beetcher col. 6:47-48 notes that all the instructions 301 in a software module are identical. Beetcher discloses that instruction 301 is a 48 bit instruction consisting of four fields: Operation; Unused; Version; and Product Number. Beetcher discloses that Fig. 3 shows contents of a typical executable software module (Beetcher col. 4:54-55, Brief description of Fig. 3). Beetcher Fig. 3 and col. 6:41-46 disclose that Fig. 3 shows software module 300 "comprises a

plurality of object code instructions capable of executing on computer system 101." Fig. 3, lower portion, also shows the contents of instruction 301. Beetcher's Fig. 3, lower portion, shows that each instruction 301 consists of four fields 302 to 305, totaling 48 bits. These are the Operation code field 302 (16 bits); the Version field 303 (8 bits); the Unused field 305 (8 bits); and the Product Number field 305 (8 bits). Beetcher col. 6:50-55 describes these four fields as follows:

...Field 305 is unused. Operation code 302 is the verb portion of the object code instruction, identifying the operation to be performed. Version 303 identifies the version level of the software module. Product number 304 identifies the product number associated with the software module. 132Dec¶20

VII.2.6 Description of How Customer Computer 101 Uses Beetcher's entitlement verification instruction 301

Beetcher explains how customer computer system 101 executes instruction 301 in connection with Fig. 10. Beetcher's Brief description of Fig. 10 states: "FIG. 10 is a block diagram of the steps required to verify entitlement during execution of a software module according to the preferred embodiment." Beetcher Fig. 10 shows that Beetcher's computer program determines if a fetched instruction, is an instance of instruction 301. See steps 1001, "Fetch next instruction" and step 1004, "Trigger?". Beetcher col. 10:51-54 explains this process, stating:

The process for executing a software module according to the preferred embodiment is shown in FIG. 10. System 101 executes the module by fetching (step 1001) and executing (step 1002) object code instructions until done (step 1003). If any instruction is an entitlement verification triggering instruction 301 (step 1004) check lock function 422 is invoked. 132Dec¶21

Thus, Beetcher discloses that instruction 301 is executed, and instruction 301 causes customer computer 101 to invoke check lock function 422. 132Dec¶22

VII.2.7 Beetcher Discloses that Check lock function 422 compares data in entry 601 in product lock table 460 to the product number contained in the Product Number field 304 in instruction 301

Beetcher discloses that check lock function 422 compares data in entry 601 in product lock table 460 to the product number contained in the Product Number field 304 in instruction 301. See Beetcher col. 10:54-65, which states:

...Check lock function 422 accesses the product lock table entry 601 corresponding to the product number contained in the triggering instruction at step 1005. If the **version number in product lock table 460** is equal to or greater than the **version number 303 contained in triggering instruction 301**, the software is entitled to execute (step 1006). In this case, check lock function 422 takes no further action, and the system proceeds to execute the next object code instruction in the software module. If the software is not entitled, check lock

function generates an exception condition, causing control to pass to exception handler 432, which will terminate program execution (step 1007). 132Dec¶23

While not essential to this analysis, for completeness, note that Beetcher indicates that it is Operation code 302 of instruction 301 that instructs customer computer 101 to invoke check lock function 422. This is because Beetcher col. 6:51-52 states that "Operation code 302 is the verb portion of the object code instruction, identifying the operation to be performed." The operation performed by step 1004, is "Trigger?", which is shown to be a decisional step, that is, a result of a comparison to something. That requires comparison of a value in the compiled object code to a predetermined value. Operation code 302 is the only set of bits of instruction 301 whose function is not specified for some other purpose, therefore the only portion of instruction 301 that can have such a predefined value. Moreover, "operation to be performed" is consistent identifying some predefined value against which a comparison to some other value is performed. 132Dec¶24

Beetcher Fig. 10 shows that Beetcher's computer program determines whether the instruction 301 indicates entitlement, by comparison of the version information in instruction 301, with the corresponding version number obtained from the entitlement key, in step 1006, "Entitlement Version < Product Version?". Beetcher states this expressly, at col. 10:56-65, which reads:

...If the **version number in product lock table 460** is equal to or greater than the **version number 303 contained in triggering instruction 301**, the software is entitled to execute (**step 1006**). ... If the software is not entitled, check lock function generates an exception condition, causing control to pass to exception handler 432, which will terminate program execution (**step 1007**). 132Dec¶25

As explained above, Beetcher discloses that product lock table 460 stores data obtained from the encrypted entitlement key 111. 132Dec¶26

Finally, Fig. 10 shows that step 1003 "More instruction?" executes when the software is determined in step 1006 to be entitled to run, and that this step merely loops back to instruction 1001, which fetches the next instruction in the object code, or terminates if there are no next instruction. And Fig. 10 shows that step 1007 "...Condition to Abort" terminates that program, if the software is determined in step 1006 to not be entitled to run. 132Dec¶27

VII.2.8 Beetcher's Customer Computer 101 Does Not Decode Beetcher's entitlement verification instruction 301

As shown above, instruction 301 is executed, just like any other instruction. Beetcher does not disclose decoding instruction 301, and Beetcher does not disclose a decoding routine to do so. 132Dec¶28

Beetcher also does not disclose using licensing information input in response to the prompt, that is, information contained in encrypted entitlement key 111, in a decoding routing to decode instruction 301. Therefore, instruction 301 does not correspond to claim 11's "first license code encoded in said software product." Therefore, Beetcher's entry of encrypted

entitlement key 111 and decryption of that key do not correspond to "using license information entered via said input in response to said prompt in a routine designed to decode a first license code encoded in said software product." 132Dec¶29

VII.2.9 Beetcher's Customer Computer 101 Does Not Have a Routine to Decode Beetcher's entitlement verification instruction 301

As noted above, Beetcher discloses executing instructions 301. Just like any other instruction, execution of instruction 301 does not involve decoding. Beetcher contains no disclosure of a routine designed to decode instruction 301. 132Dec¶30

VII.2.10 Beetcher's Customer Computer 101 Does Not Use Information from the Encrypted Entitlement Key 111 to Decode Beetcher's entitlement verification instruction 301

As noted above, Beetcher discloses customer computer 101 decrypting and storing data from encrypted entitlement key 111, and executing instructions 301. The only relation between the entitlement data in key 111 and the instruction 301 that Beetcher discloses is the comparison of bits of instruction 301 to bits derived from encrypted entitlement key 111. So Beetcher does not disclose using the encrypted entitlement key or information contained therein to decode instruction 301. 132Dec¶31

VII.3 Claim 11: "loading a software program ... said software product outputting a prompt for input of license information; and "

Beetcher describes a system in which the customer computer 101 (on which the Beetcher's software module is installed) prompts the user for Beetcher's encrypted entitlement key 111, and must receive that key in order to run. 132Dec¶32

VII.3.1 Beetcher Discloses Customer Computer 101 Prompts for Input of Encrypted Entitlement Key 111

Beetcher discloses the computer prompting for encrypted entitlement key 111. Beetcher discloses that the user inputs the encrypted form of that key, encrypted entitlement key 111. For example, Beetcher's Summary of the Invention section begins at col. 4:1-9, which state:

Software is distributed according to the present invention without entitlement to run. **A separately distributed encrypted entitlement key** enables execution of the software. The key includes the serial number **of the machine** for which the software is licensed, together with a plurality of entitlement bits indicating which software modules are entitled to run on the machine. 132Dec¶33

And Beetcher explains in the next sentences at col. 4:9-14 that this encrypted entitlement key is decrypted *after being input* to the computer:

A secure decryption mechanism is contained **on the machine**. The decryption mechanism fetches the machine serial number and uses it as a key to **decrypt the entitlement key**. The entitlement information is then stored in a product lock table in memory.

This passage clarifies that what the computer receives is the encrypted key, by explaining that the encrypted key is decrypted after receipt in the computer. 132Dec¶34

Further, Beetcher clarifies that the user of the computer receives only the encrypted entitlement key, not the unencrypted entitlement key. That is, Beetcher col. 5:59-61 explains that "Encrypted entitlement key 111 is sent from the software distributor to the customer by mail, telephone, or other appropriate means." And Beetcher clarifies that the computer has a mechanism for decrypting the key. That is, Beetcher col. 6:66-67 then explains that "Computer system 101 contains means for receiving and decoding [sic; decrypting] encrypted entitlement key 111." And Beetcher's summary of the 4 steps of Beetcher's invention at col. 8:53-67, specifies that the computer receives and decrypts the entitlement key. That is, Beetcher states at col. 8:60-62 that "In the third part, computer system 101 *receives, decodes* [sic; decrypts] *and stores entitlement key 111*, and sets product lock table 460." 132Dec¶35

Beetcher describes no mechanism for handling receipt of unencrypted entitlement key 200. Beetcher never suggests that the user input unencrypted entitlement key 200. 132Dec¶36

Furthermore, NFOA page 12:12-14 states "*Beetcher details that the customer enters entitlement key 111, i.e., license information, in response to the prompt.*" Thus, the NFOA asserts that encrypted entitlement key 111 is the licensing information. 132Dec¶37

Thus, Beetcher describes a system in which the customer computer 101 (on which the Beetcher's software module is installed) prompts the user for Beetcher's encrypted entitlement key 111, and must receive that key in order to run. 132Dec¶38

VII.3.2 Claim 11 defines a process in which the licensing information input in response to the prompt is not part of the loaded software product

Claim 11 requires "license information" be "input in response to said prompt" by the software product loaded on the computer. That follows from claim 11's limitation "**loading** a software product **on a computer.**" It follows necessarily because the sequence of claimed steps, loading, prompting, and then using, are predicated on the occurrence of the earlier steps. The software product cannot prompt, unless it has been loaded. The software product cannot use information input in response to a prompt, unless the prompt has already occurred which means the software has already been loaded. Therefore, claim 11 defines a process in which the licensing information input in response to the prompt is not part of the loaded software product. 132Dec¶39

VII.3.3 Beetcher's Encrypted Entitlement Key 111 Is Input In Response to a Prompt, Does Not Correspond to Claim 11's Installed Software Product

Like claim 11's prompt, Beetcher's encrypted entitlement key 111 is input in response to Beetcher's prompt. 132Dec¶40

Beetcher defines his installed software module as object code. See Beetcher's Brief Description of Fig. 3, col. 4:54-55 ("FIG. 3 shows the contents of a typical executable software

module") and description at col. 6:41-45:

In the preferred embodiment, software modules are distributed as **compiled object code**. A typical software module 300 is shown in FIG. 3. The software module **comprises a plurality of object code instructions capable of executing on computer system 101**. 132Dec¶41

Object code is a set of instruction codes that is understood by a computer, as defined in Attachment 13, which is the definition of "Object Code" from the Technopeida website. Beetcher's encrypted entitlement key 111 is not object code. Beetcher's encrypted entitlement key 111 is not a set of instructions that is understood by a computer. 132Dec¶42

Moreover, Beetcher col. 8:60-65 describes his entitlement key as distinct from his software modules, at col. 8:60-65:

...In the **third part**, computer system 101 receives, decodes and stores entitlement key 111, and sets product lock table 460. In the **fourth part**, the software module executing on system 101 causes system 101 to verify entitlement upon encountering an entitlement verification instruction.

Thus, Beetcher's installed software module does not include encrypted entitlement key 111. Therefore, Beetcher's encrypted entitlement key 111 (input in response to Beetcher's prompt) and its unencrypted version of the key and information therefrom that is stored in memory do not correspond to claim 11's loaded software product. 132Dec¶43

VII.4 Claim construction, Claim 11: "encode ... in a routine designed to decode"

The construction of "encode ... in a routine designed to decode" in claim 11 is relevant to the rejections. The subject patent makes it perfectly clear (1) that "encode" and "decode" have the meanings associated with those terms in the digital data arts, and (2) that "routine" refers to the part of the installed software code. Software function only in digital processing. In digital processing, encode and decode mean changes in digital representation of information. Consequently, claim 11's encode and decode refer to changes in digital representations of information. This is clear from the specification, the claims, and the general definitions of encode and decode. 132Dec¶44

Specification

The Abstract of the subject patent begins:

An apparatus and method for **encoding and decoding** additional information **into a digital information** in an integral manner. More particularly, the invention relates to a method and device for data protection. 132Dec¶45

The "Field of the Invention" section, first sentence, states that:

The invention relates to the protection of *digital* information. 132Dec¶45

The Background of the Invention, first sentence, states:

Increasingly, commercially valuable information is being created and stored in "**digital**" form. 132Dec¶46

This clearly specifies the invention is directed to encoding and decoding of information represented in **digital** form. And the specification repeatedly refers to decoding using a key, which requires a digital decoding algorithm, and that the software may include the encode and decode functions. For example, the specification states:

To decode the information, a predetermined key is used **before playing the digital information** at steps 140 and 150.

A key-based decoder can act as a "**plug-in**" **digital player** of broadcast signal streams without foreknowledge of the encoded media stream. Moreover, the data format orientation is used to partially scramble data in transit to prevent unauthorized descrambled access by **decoders that lack authorized keys**. A distributed key can be used to unscramble the scrambled content because a decoder would understand how to process the key.

The same keys can be used to later validate the embedded digital signature, or even fully decode the digital watermark if desired 132Dec¶47

Moreover, the specification refers to "code resource" of an "application" which means software. The specification states:

Note further that the **application contains a code resource which performs the function of decoding an encoded code resource from a data resource**. *** 3) Once it has the license code, it can then generate the proper decoding key to access the essential code resources.

Software necessarily functions on digital computers and therefore is limited to digital processing. 132Dec¶48

Claims

Claim 11 is specifically limited to software, and therefore necessarily has the meaning for encode and decode specific to digital representations. That meaning is one of representation of information digitally. 132Dec¶49

Moreover, claim 11 recites "first license code encoded in said software product." In this regard, the specification col. 13:36-48 states "This method, then, is to choose **the key** so that it corresponds, **is equal to**, or is a function of, **a license code** .. Alternatively, the key, possibly random, can be stored as a data resource and **encrypted** with a derivative of the license code." Thus, a corresponding disclosure to what claim 11 recites appearing in the specification refers to encoding as changing the digital representation of data. 132Dec¶50

Further, claim 11 requires software having "a *routine* designed to decode" encoded information. The specification explains that the claimed routine is part of the software

"application [that] contains a code resource which performs the function of decoding." That is, claim 11 defines a **software routine designed to decode the license code encoded in the software product**. Software acts on digital data. As such, the encoding and decoding defined by claim 11 are limited to digital encoding, which means changing the digital representation of information. 132Dec¶51

Example of encoding/decoding based upon the specification

Moreover, the specification mentions both ASCII and binary encoding. Therefore, to make the concept of the claimed "encode... routine designed to decode," more concrete, and the meaning of digital representation clear, consider the following example of a digital encoding representing the number ten, in base 10 and base 2, using the 7 bit ASCII specification.

132Dec¶52

Assuming the ASCII 7 bit specification, "10" in base 10 would be the ASCII code for numeral "1", which are "011 0001", followed by the ASCII code for numeral "0", which are "011 0000." So the number ten in base 10 represented in 7 bit ASCII code would appear in digital memory to be the following two sets of 7 digits: "011 0001", "011 0000". 132Dec¶53

But representing the number "10" in base 2 results in the digital bits "1010". So when the number 10 is encoded in base 2, and represented in the 7 bit ASCII specification, there are a sequence of four characters in sequence. So representing "10" in binary, in 7 bit ASCII would result on the following four sets of 7 digits: "011 0001", "011 0000", "011 0001", "011 0000".

132Dec¶54

Converting from binary to decimal, and from binary and decimal to 7 bit ASCII are examples of what encoding and decoding mean in the digital arts. That is how the same value (that is information) is represented differently. Encoding, in this patent, means changing the digital representation of information. 132Dec¶55

Definition in the Art of encoding and decoding

Moreover, encoding and decoding have clear meanings in the digital computer arts. A search on "what does encoding mean in the computer" using the Google search engine, returns the result stating:

In computers, encoding is the process of putting a sequence of characters (letters, numbers, punctuation, and certain symbols) into a specialized format ...Decoding is the opposite process -- the conversion of an encoded format back into the original sequence of characters. Nov 14, 2005

A copy of the screen image showing the Google search query and quote definitions is Attachment 11. 132Dec¶56

The URL link cited by the Google search is:

<https://searchnetworking.techtarget.com/definition/encoding-and-decoding>. 132Dec¶57

A copy of the web page provided by

<https://searchnetworking.techtarget.com/definition/encoding-and-decoding> Attachment 12.

132Dec¶58

Attachment 12 provides definitions of encoding and decoding. Attachment 12, and is

quoted below (emphasis added):

Definition encoding and decoding
Posted by: Margaret Rouse
WhatIs.com

In computers, encoding is the process of putting a sequence of characters (letters, numbers, punctuation, and certain symbols) **into a specialized format** for efficient transmission or storage. **Decoding is the opposite process** -- the conversion of an encoded format **back into the original sequence of characters**. Encoding and decoding are used in data communications, networking, and storage. The term is especially applicable to radio (wireless) communications systems.

The **code used by most computers for text files is known as ASCII** (American Standard Code for Information Interchange, pronounced ASK-ee). ASCII can depict uppercase and lowercase alphabetic characters, numerals, punctuation marks, and common symbols. Other commonly-used codes include Unicode, BinHex, Uuencode, and MIME. In data communications, Manchester encoding is a special form of encoding in which the binary digits (bits) represent the transitions between high and low logic states. In radio communications, numerous encoding and decoding methods exist, some of which are used only by specialized groups of people (amateur radio operators, for example). The oldest code of all, originally employed in the landline telegraph during the 19th century, is the Morse code.

The terms encoding and decoding are often used in reference to the processes of analog-to-digital conversion and digital-to-analog conversion. In this sense, these terms can apply to any form of data, including text, images, audio, video, multimedia, computer programs, or signals in sensors, telemetry, and control systems. Encoding should not be confused with encryption, a process in which data is deliberately altered so as to conceal its content. **Encryption can be done without changing the particular code** that the content is in, and encoding can be done without deliberately concealing the content. 132Dec¶59

This passage, in the last paragraph, explains that encoding relates to both digital and analog. While generally true, the subject patent is specifically directed to digital information, and the claims are limited to software programs, which only functions on digital information. The specification and claims therefore excludes claim constructions reading on analog encoding and decoding. 132Dec¶60

Definition of “routine”

Claim 11 recites “routine”. To avoid doubt, in the software arts, a “routine” defines a section of a computer program that performs a particular task, as stated in Attachment 14, which is the definition of “routine” from the Webopedia website. 132Dec¶61

The definitions of decode are important because Beetcher does not disclose that its software modules 300 loaded on customer computer 101 contains a routine designed to decode

instructions when 301; to change the digital representation or instruction 301. 132Dec¶62

VII.5 Response to the NFOA's Assertions Regarding Beetcher

The NFOA concludes that Beetcher discloses claim 11's "said software product using license information entered via said input in response to said prompt *in a routine designed to decode* a first license code *encoded* in said software product." NFOA page 12:7 to 17:1. This conclusion is incorrect. The NFOA lacks application of asserted facts to this claim language and contains irrelevant and incorrect statements of fact regarding Beetcher, discussed below. 132Dec¶63

VII.5.1 Contrary to the NFOA, Beetcher *does not* disclose using the entitlement key (111, 200; encrypted, unencrypted) in a routine designed to decode anything in Beetcher's software, as required by claim 11

NFOA page 12:15-17, states "After entering that key, Beetcher teaches that the customer's computer uses a decode key to initiate unlock routine 430 **to decode the license code encoded in the software product.**" 132Dec¶64

This assertion of fact is incorrect. Beetcher does not disclose that unlock routine 430 decodes a license code encoded in the software product. Beetcher discloses that **unlock routine 430 acts only on the encrypted entitlement key, not Beetcher's software module 300**, as shown in the following passages: 132Dec¶65

Beetcher col. 7:39-42:

...Unlock routine 430 uses the unique machine key to decodes entitlement key 111, and **stores encrypted entitlement key 111** in encoded product key table 450.

Beetcher col. 9:55-60:

The entitlement key is passed to unlock routine 430, which handles the decoding process. Unlock routine 430 causes get machine key function 420 to retrieve the machine serial number and generate the machine key at 902. Unlock routine 430 then uses the machine key to decode the entitlement key 111 at step 903.

Beetcher col. 9:67 to 10:5:

...Unlock routine 430 scans each product entitlement flag 205 in the decoded key (step 904). If the product entitlement flag is set to '1' (indicating entitlement) at step 905, the corresponding entry in product key table 450 is replaced with the new entitlement key [sic; key's], version number, and charge group value at step 905. 132Dec¶66

These passages show that Beetcher's unlock routine 430 performs the decryption of

encrypted entitlement key 111 and stores certain information (version numbers and charge groups) read from the decrypted entitlement key, in computer memory (key table 450 and product lock table 460). As explained above, encrypted entitlement key 111 and its decrypted version are not part of Beetcher's software module. Therefore, Beetcher's entitlement key does not correspond to the installed software product defined by claim 11. Therefore, Beetcher's "decoding" [sic; decrypting] of the encrypted entitlement key by routine 430 is not decoding of a license code encoded in Beetcher's installed software product. As explained herein above, the fact that Beetcher's computer decrypts and then stores in memory in tables 450, 460, certain information contained in the entitlement key, does not make that information "encoded" in Beetcher's software module. 132Dec¶67

In any case, Beetcher discloses the decryption and storing of data contained in the entitlement key as separate from Beetcher's software module. See col. 8:60-67:

In the third part, computer system 101 receives, decodes and stores entitlement key 111, and sets product lock table 460. **In the fourth part, the software module executing on system 101** causes system 101 to verify entitlement upon encountering an entitlement verification instruction. The first two parts are performed under the control of the software distributor. The last two are performed on the customer's system 101.

Thus, the decrypting and storing of information contained in encrypted entitlement key 111 could not correspond to the decoding of a "license code encoded *in said software product*," as required by claim 11. Thus, storing information contained in Beetcher's encrypted entitlement key 111 in memory of customer computer 101 does result in that information becoming "encoded **in said software product**," as required by claim 11. 132Dec¶68

Further, Beetcher does not disclose unlock routine 430 acting on instruction 301. Accordingly, Beetcher cannot disclose unlock routine 430 decoding instruction 301.

In summary, Beetcher does not disclose that unlock routine 430 decodes a license code encoded in the software product. 132Dec¶69

VII.5.2 Contrary to the NFOA, Beetcher also does not disclose using the entitlement key in a routine designed to decode entitlement verification triggering instruction 301; does not disclose a routine designed to decode instruction 301; and does not disclose decoding instruction 301, all of which are required by claim 11

The NFOA page 12:7 to page 16 bottom concludes that Beetcher discloses claim 11's "software product using license information entered via said input in response to said prompt in a routine designed to decode a first license code encoded in said software product." 132Dec¶70

Specifically, **NFOA, page 12:18-20** assert that:

...Beetcher's Figures 4 and 9a, which are provided below, show the software using the key (i.e., license information) entered by the customer to decode a first license code encoded in the software product.

This statement is incorrect. 132Dec¶72

Beetcher states that “FIG. 4 shows the hardware and software structures required on the customer's computer system to support the software protection mechanism according to the preferred embodiment.” Beetcher Fig. 4 does not show software. Nor does it show anything entered by the customer decoding anything in Beetcher’s software module 300. 132Dec¶73

Beetcher states that “FIG. 9 is a block diagram of the steps required to decode an entitlement key and maintain a record of entitlement status on the customer's computer system, according to the preferred embodiment.” Fig. 9a shows the process of receipt by customer computer 101 of encrypted entitlement key 111 (step 901) through the storage of information contained in the key in tables 450, 460 (step 904 for table 450 and step 907 for table 460). Decoding of the entitlement key and storing in memory the information contained in that key is not a disclosure of decoding a license code **in the software product**. The entitlement code, as explained in Beetcher and discussed above, is not part of Beetcher’s software module 300. 132Dec¶74

VII.5.2.1 Contrary to the NFOA, Beetcher Does Not Disclose Using the Key’s Version and Product Number to Decode a License Code

The NFOA, page 14:12-13 states “Beetcher's software product uses the key's version and product number fields to decode a license code.” 132Dec¶75

This statement lacks citation to Beetcher. Therefore, it should be given no weight. Moreover, this statement is incorrect. Beetcher does not disclose its software product decoding a license code. As explained above, Beetcher discloses customer computer 101 decrypting the encrypted entitlement key received by that computer, and then comparing information contained in the entitlement key to version number contained in instruction 301. Nowhere does Beetcher disclose decoding of an instruction in software module 300. 132Dec¶76

III.5.2.2 Entitlement verification triggering instructions

The NFOA, page 15:1-5, states:

...When compiling and translating the software code, Beetcher explains that the code includes entitlement verification triggering instructions encoded into the software. (Id. at 6:41 -58. 11 :4-59; see also id. at 4:14-23, 8:5-22, 8:56-9:20) Beetcher's triggering instructions are encoded into the software when the software code is compiled and translated, as shown in Figure 3 provided below: 132Dec¶77

This statement in the NFOA does not state that Beetcher discloses decoding entitlement verification triggering instructions. This assertion in the NFOA does not support rejection of claim 11. 132Dec¶78

VII.5.2.3 Triggering Instruction 301

The NFOA **page 15:4-5** states that "Beetcher's triggering instructions are encoded into the software when the software code is compiled and translated, as shown in Figure 3." This statement is not relevant because claim 11 requires **decoding** of a license code encoded in the software product, not encoding. Claim 11 defines using the input licensing information “to **decode** a first license code encoded in said software product.” Executing instruction 301, which invokes product lock table 422, is not decoding. 132Dec¶79

VII.5.2.4 Key table 460

Next, the NFOA, page 15:6 to 16:1, states:

Beetcher explains that its software code verifies the customer is entitled to use the software when the code encounters a triggering instruction. When it encounters one of these instructions, Beetcher's code accesses the license key information stored in the key table 460. (Id. at 10:48-11:39; See also id, at Abstract, 8:14-22, 8:53-9:20, Fig. 10). 132Dec¶80

This statement in the NFOA does not state that Beetcher discloses decoding entitlement verification triggering instructions. Key table 460 contains information received from the encrypted entitlement key 111, as discussed above. This key and the information contained in key table 460 are not part of Beetcher's software module 300. Beetcher's software module accessing this information is not decoding license information contained **in the software product**, as required by claim 11. Thus, the assertions at NFOA, page 15:6 to 16:1 do not support rejection of claim 11. 132Dec¶81

VII.5.2.5 Check Lock Function 422

The NFOA **page 16:3-5 concludes** that "As such, a POSITA would have understood that Beetcher uses its license information in a routine, such as check lock function 422, designed to decode a first license code encoded in a software product via the triggering instructions." 132Dec¶82

In response, the NFOA's conclusion that Beetcher discloses that check lock function 422 is designed to decode a first license code encoded in a software product is supported no reasoning. And it is wrong. Beetcher does not disclose that check lock function 422 is designed to decode a first license code encoded in a software product. 132Dec¶83

Instead, Beetcher discloses that check lock function 422 compares data in entry 601 in product lock table 460 to the product number contained in the Product Number field 304 in instruction 301. See Beetcher col. 10:54-65, quoted above in the description of check lock function 422. Performing a comparison is not decoding. 132Dec¶84

Beetcher discloses that check lock function 422 functions by *reading* or *accessing* values from lock table 422. See Beetcher:

...Check lock function 422 accesses product lock table 460 and *reads* one of the entries to verify entitlement. [Beetcher, col. 7:29-31; bold added for emphasis.]

...The executable code contains entitlement verification triggering instructions 301 (only one shown), which are *executed* by horizontal microcode check lock function 422. [Beetcher, col. 8:19-22; bold added for emphasis.]

Check lock function 422 *accesses* the product lock table entry 601 corresponding to the product number contained in the triggering instruction at step 1005. [Beetcher, col. 8:54-56; bold added for emphasis.] 132Dec¶85

The NFOA, page 16:5-11, continues, presenting the following inaccurate and uncited block quote, apparently lifted from the reexamination request, but inaccurately reproduced in the NFOA.

If any instruction is an entitlement verification triggering instruction 301 (step 1004) check lock function 422 is invoked. Check lock function 422 accesses the product lock table entry 601 corresponding to the product number contained in the triggering instruction at step 1005. If the version number in product lock table 460 is equal to or greater than the version number 303 contained in triggering instruction 301, the software is entitled to execute (step 1006). (Id. at 10:52-62, Fig. 10; Silva Declaration at ¶ 46-51). 132Dec¶88

This block quote contains no citation. Moreover, nothing in this block quote identifies the source of "Id." appearing therein. 132Dec¶87

After searching the reexam request, the undersigned found this quote to be an inaccurate copy of a passage in the Reexam request, page 39. Reexam request page 39:6-12 states the following:

If any instruction is an entitlement verification triggering instruction 301 (step 1004) check lock function 422 is invoked. Check lock function 422 accesses the product lock table entry 601 corresponding to the product number contained in the triggering instruction at step 1005. If the version number in product lock table 460 is equal to or greater than the version number 303 contained in triggering instruction 301, the software is entitled to execute (step 1006).

At the end of this text from the reexam request is a cite to footnote 144. Footnote 144 appears in the bottom margin of page 39 of the reexam request. Footnote 144 states "¹⁴⁴ Id. at 10:49-60; see also id. at 10:48-49, 10:60-11:3; Silva Declaration at ¶¶78-82." The "Id." in the reexamination request, refers to footnote 144, which refers to Beetcher. So this passage cites, for support, Beetcher 10:49-60; 10:48-49; and 10:60-11:3. These passages in Beetcher do not support the rejection. 132Dec¶88

Beetcher 10:48-49 reads "The process for executing a software module according to the preferred embodiment is shown in FIG. 10." All Fig. 10 shows is that the "trigger" in step 1004 is compared to the product lock table entry, in step 1006. Nothing here discloses decoding a trigger. Therefore, Fig. 10 does not disclose claim 11's "**routine designed to decode**" such a trigger "**encoded in said software product**." 132Dec¶89

Beetcher 10:49-60 reads:

System 101 executes the module by fetching (step 1001) and executing (step 1002) object code instructions until done (step 1003). If any instruction is an

entitlement verification triggering instruction 301 (step 1004) check lock function 422 is invoked. Check lock function 422 *accesses* the product lock table entry 601 corresponding to the product number contained in the triggering instruction at step 1005. If the version number in product lock table 460 is *equal to or greater than* the version number 303 contained in triggering instruction 301, the software is entitled to execute (step 1006). 132Dec¶90

All Beetcher 10:49-60 discloses is accessing entitlement verification triggering instruction 301 and comparing that to product lock table data. Nothing here discloses decoding an entitlement verification triggering instruction 301. Therefore, Beetcher 10:49-60 does not disclose claim 11's "**routine designed to decode**" such a triggering instruction "**encoded in said software product.**" 132Dec¶91

Beetcher 10:60-11:3 reads:

In this case, check lock function 422 takes no further action, and the system proceeds to execute the next object code instruction in the software module. If the software is not entitled, check lock function generates an exception condition, causing control to pass to exception handler 432, which will terminate program execution (step 1007). The system does not save the results of an entitlement check which shows that the software is entitled. Therefore, when a triggering instruction is again encountered in the software module, the system again verifies entitlement as described above. 132Dec¶92

All Beetcher 10:60-11:3 discloses is the results of the comparison shown in Fig. 10, step 1006. The results of the comparison do not related to operations performed on the data used in the comparison. Therefore, Beetcher 10:60-11:3 does not disclose claim 11's "**routine designed to decode**" anything "**encoded in said software product.**" 132Dec¶93

In summary, contrary to the NFOA, Beetcher **does not** disclose that check lock function 422 is designed to decode a first license code encoded in a software product via the triggering instructions. 132Dec¶94

VII.5.2.6 Execution of Instruction 301

Next, the NFOA, page 16:12-26, states that:

Moreover, Beetcher teaches that the triggering instructions will be encoded into the code resources controlling software functionality: [quote lines 17-22, *lacking citation*, omitted] And Beetcher details that "the triggering instruction is also a direct instruction to perform some other useful work I [sic] Execution of the triggering instruction causes system 101 to perform some other operation simultaneous with the entitlement verification." (Id. at 6:58-65 (Beetcher specifies that these functions are those "which do not require that an operand for the action be specified in the instruction."); Silva Declaration at ¶ 52-53). 132Dec¶95

These statements fail to assert that Beetcher discloses a "**routine designed to decode**" such a triggering instruction "**encoded in said software product.**" The fact that Beetcher uses portions of triggering instructions for purposes of effecting the functionality of Beetcher's software does not indicate encoding or decoding; it merely indicates execution of the instruction. 132Dec¶96

VII.7 Contrary to the NFOA, Patent Owner's Description of Beetcher in the Patent Owner's Statement, Was Accurate

The NFOA contains a section responding to the Patent Owner Statement. See NFOA pages 89-108. This section contains the following passage at NFOA 93:6-15:

Patent Owner specifically argues that "encrypting and decrypting the key is not encoding or decoding and [sic; the] software module ..." and that "Beetcher does not disclose encoding or decoding of software using a key, or decoding license code encoded in software."

In response, the Examiner respectfully submits that the described process starts by placing entitlement verification triggers in the object code. The process involves creation of a program template that is input into a translator along with product numbers and version numbers. From this the encoded code is generated, where this code is encoded with hidden functionality not available until it is decoded. (see column 9, lines 1-20). 132Dec¶97

The Examiner's statements of "responses" implies that the Examiner disagrees with the Patent Owner's statements of fact. The Examiner's statement, like the statement of the rejection, leaves the reader at sea as to the meaning of these statements. These statements are improper because they imply that the Patent Owner's assertion of fact are incorrect, without expressly stating so. In any subsequent office action the Examiner is requested to expressly withdraw this "response" and to clarify that the Patent Owner's Statements of fact are accurate. 132Dec¶98

The implications based upon the foregoing "response" of the Examiner are incorrect. Patent Owner's statement in the Patent Owner Statement were and remain correct. Specifically: 132Dec¶99

It is a fact that "encrypting and decrypting the key is not encoding or decoding and [sic; the] software module ..." 132Dec¶100

It is a fact that "Beetcher does not disclose encoding or decoding of software using a key." 132Dec¶101

And it is a fact that "Beetcher does not disclose decoding a license code encoded in software." 132Dec¶102

The Examiner's response cites Beetcher column 9:1-20. That passage does not contradict the Patent Owner's statements of fact. That passage reads:

When the software distributor compiles a software module, it must place the entitlement verification triggers in object code. A typical such process, which takes place on development system 125, is shown in FIG. 7. Source code is generated by a programmer in the normal fashion, without inclusion of

entitlement verification triggers. The source code is input into compiler 126 at step 701 to produce a program template at 702. The program template comprises machine instructions at virtual machine level 404 (i.e. above machine interface 405). The program template serves as input to translator 127 at step 704, along with its product number and version number identification. Translator 127 automatically generates a substantial number of entitlement verification triggers, inserts them in random locations in the object code, and resolves references after the triggers have been inserted. The resultant executable object code form of the software module which is output at 705 contains the embedded triggers. This executable form of the module comprises object code instructions at executable code level 403. 132Dec¶102

Beetcher column 9:1-20 does not disclose or suggest that the entitlement key (either in its unencrypted form, unencrypted entitlement key 200, see col. 9:34-36, col. 9:42-48) is involved in the process of encoding the software module. 132Dec¶104

In fact, Beetcher explains that the compiling of the software code discussed in the paragraph at col. 9:1-20 occurs in a distinct part from the generation of the entitlement key 200 discussed in the paragraph at col. 9:21-48. Specifically, Beetcher provides an overview of the four steps involved in his process in the paragraph at col. 8:53-67. There, in pertinent part, Beetcher states:

The operation of a software module on computer system 101 in accordance with the preferred embodiment of the present invention will now be described. There are four parts to this operation. **In the first part**, a plurality of entitlement verification triggering instructions are placed in the executable object code form of the software module. **In the second part, an encrypted entitlement key 111 authorizing access to the software module is generated.** In the third part, computer system 101 receives, decodes and stores entitlement key 111, and sets product lock table 460. In the fourth part, the software module executing on system 101 causes system 101 to verify entitlement upon encountering an entitlement verification instruction.

In this passage, Beetcher indicates no relation between placing the entitlement verification triggering instructions 301 in the object code and generating and entitlement key 111 authorizing access to the software module. Moreover, the description of the compiling in the "first part" suggests that the generation of the encryption key in the "second part" occurs after compiling. In which case, there is no encryption key in existence at the time of compiling. 132Dec¶105

Beetcher's disclosure clearly indicates that instruction 301 is not a function of entitlement key 200. That means instruction 301 is not encrypted using entitlement key 200 and therefore cannot be decrypted or encoded using the entitlement key. 132Dec¶106

In fact, Beetcher teaches a method that improves upon the prior art by providing *exactly the same software module to all users*, but different encrypted versions of entitlement key 200 to each user. Beetcher teaches that, by explaining that its invention provides the benefit of a simplified method of distribution of the same software module to all of its customers; not

software product that are individually encoded or encrypted based upon each customer's unique encryption key. See, e.g., col. 2:49-52 in the Background identification of the prior art ("Another restricted entitlement method is to encode *user or machine specific information* in the software itself.") and col. 3:51-53 in the Summary of the Invention section indicating one of Beetcher's goals is to avoid that prior art ("Another object of this invention is to *simplify the distribution system* of a distributor of software protected against unauthorized use"); followed by Beetcher's express disclosure that each of its software modules distributed to all of its users are identical at col. 4:34-39 ("Because the software itself does not contain any entitlement, **no restrictions on the distribution are necessary**. In the preferred embodiment, the distributor of software may record multiple software modules on a single generic medium, and **distribute the same** recorded set of **modules to all its customers**.") 132Dec¶107

Thus, Beetcher clearly discloses that its software module are neither encoded nor encrypted using Beetcher's encryption key 200. 132Dec¶108

VII.8 Claim 12

Claim 12 reads:

12. A method for encoding software code using a computer having a processor and memory, comprising:
storing a software code in said memory;
wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system; and
encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key encoded software code; and
wherein, when installed on a computer system, said first license key encoded software code will provide said specified underlying functionality only after receipt of said first license key. 132Dec¶109

VII.8. Claim 12 Claim Construction, “encoding using at least a first license key and an encoding algorithm”

Claim 12 recites “encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key encoded software code;” 132Dec¶110

Claim 12 defines encoding “software code” by using both the first license key and the encoding algorithm to encode the software code, stating “encoding using at least a first license key and an encoding algorithm.” The specification col. 13: 8-49 explains this process, stating:

One method of the present invention is now discussed. When code and data resources are compiled and assembled into a precursor of an executable program the next step is to use a utility application for final assembly of the executable application. The programmer marks several essential code resources in a list displayed by the utility. The utility will choose one or several essential code

resources, and **encode them** into one or several data resources **For the encoding of the essential code resources, a "key" is needed.** ... The purpose of this scheme is to make a particular licensed copy of an application **distinguishable from any other.** ... The assembly utility can be supplied with a key generated from a license code generated for the license in question. Alternatively, the key, possibly random, can be stored as a data resource and encrypted with a derivative of the license code. **Given the key, it encodes one or several essential resources into one or several data resources.** 132Dec¶111

This passage clarifies that the key chosen for the encoding process is what makes the resulting code “distinguishable from any other” copy. That is, the key is an input to the encoding algorithm. This is the conventional meaning of a key in the encoding arts. It is that which is used in the encoding algorithm to encode some other data. 132Dec¶112

To avoid doubt, this passage does not describe encoding software code that includes a key in the software code. Instead, it clearly describes conventional key encoding of something else, and the something else in this case are essential code resources of the software code. 132Dec¶113

Consistent with that description in the specification, the broadest reasonable construction of claim 12's “**encoding using at least a first license key** and an encoding algorithm” refers to an encoding process in which the encoding algorithm uses a key to encode something else. And claim 12 defines that the encoding is performed on “said software code.” Claim 12 therefore defines using a key based encoding algorithm and its key to encode software code. 132Dec¶114

Using a key based encoding algorithm and a particular key for encoding is referred to herein below as key based encoding. 132Dec¶115

VII.9 Beetcher does not disclose “encoding using at least a first license key and an encoding algorithm.”

Beetcher does not disclose key based encoding to form its software module. Instead, Beetcher discloses the following process of forming its distributed software module.

When the software distributor compiles a software module, it must place the entitlement verification triggers in object code. A typical such process, which takes place on development system 125, is shown in FIG. 7. Source code is generated by a programmer in the normal fashion, without inclusion of entitlement verification triggers. The source code is input into compiler 126 at step 701 to produce a program template at 702. The program template comprises machine instructions at virtual machine level 404 (i.e. above machine interface 405). The program template serves as input to translator 127 at step 704, along with its product number and version number identification. Translator 127 automatically generates a substantial number of entitlement verification triggers, inserts them in random locations in the object code, and **resolves references after the triggers have been inserted.** The resultant executable object code form of the software module which is output at 705 contains the embedded triggers. This

executable form of the module comprises object code instructions at executable code level 403. 132Dec¶116

So Beetcher teaches conventional compiling of its high level code, followed by inserting instructions 301 at various locations therein, and then followed by resolving the references (addresses) in the compiled code caused by the insertion of instances of instructions 301. Beetcher does not teach key based encoding to form its software module. Accordingly, Beetcher does not anticipate claim 12. 132Dec¶117

The NFOA, page 25 argues that Beetcher discloses and “encoding algorithm to encode the first license key” and corresponds the key to instruction 301. For the foregoing reasons, instruction 301 does not correspond to the key required by claim 12 for encoding the software program. 132Dec¶118

VII.10 Claim 12 Claim construction, “only after receipt of said first license key.”

Claim 12 recites “wherein, when installed on a computer system, said first license key encoded software code will provide said specified underlying functionality only after receipt of said first license key.” “Only after receipt of” clearly indicates that the first license key must be received by the software after the software is installed on the computer. 132Dec¶119

VII.11 Beetcher Does not Disclose Claim 12s “only after receipt of said first license key.”

Beetcher’s software module 300 does not receive Beetcher’s instruction 301 after being installed on Beetcher’s customer computer 101. Therefore, Beetcher’s instruction 301 does not correspond to claim 12’s receipt of a first license key. 132Dec¶120

The NFOA, page 26 states that “Beetcher discloses element 12.4. Specifically, Beetcher explains that its first license key encoded software code provides the specified underlying functionality only after receipt of the first license key. ... Beetcher teaches **encoding the triggering instructions into the software code** that is decoded **via the first license key.**” 132Dec¶121

This statement is logically flawed because it is inconsistent with the NFOA’s correspondence of license key to instruction 301 in the prior section of the NFOA. The claimed license key cannot correspond to both instruction 301 and the encrypted entitlement key 111. Moreover, what Beetcher discloses as being received after the software is installed is only the encrypted entitlement key. The NFOA conveniently fails to mention that the encrypted entitlement key is what the NFOA previously equated to the licensing information in discussing claim 11. 132Dec¶122

Claim 12 requires that “said first license key encoded software code will provide said specified underlying functionality only after receipt of said first license key.” So the first license key can only correspond to encrypted entitlement key 111. In that case, Beetcher fails to disclose claim 12’s “encoding, by said computer using at least a first license key” because Beetcher does not disclose encoding using encrypted entitlement key 111. 132Dec¶123

VII.12 Claim 12 Claim Construction “first license key”

The specification col. 13:27 to 14:3 states “For the encoding of the essential code resources, a ‘key’ is needed. ... The purpose of this scheme is **to make a particular licensed**

copy of an application distinguishable from any other. It is not necessary to distinguish every instance of an application, merely every instance of a license. A **licensed user may then wish to install multiple copies** of an application, legally or with authorization. ... Note that the application can be copied in an uninhibited manner, but must contain the license **code issued to the licensed owner**, to access its essential code resources.” This passage clarifies that the license code is specific to the licensed owner. And the specification col. 13:36-37 then explains “This method, then, is to choose the key so that it corresponds, is equal to, or is a function of, a license code...” 132Dec¶124

So the specification explains that the key is also specific to the licensed owner. Claim 12 therefore defines the “first license key” to be specific to the licensed owner. 132Dec¶125

VII.13 Beetcher Does Not Disclose Instruction 301 is a “first license key,” as Defined by Claim 12

Beetcher discloses that instruction 301 stores software version information, not a license key. And Beetcher discloses that the same software module 300 is distributed to all of its customers containing this generic instruction 301. Accordingly, corresponding instruction 301 to a license key that must be received by the installed software program is incorrect for this additional reason. 132Dec¶126

The NFOA, for example at page 23 identifies instruction 301 as corresponding to the claimed “licensing key.” For the reason just noted, that correspondence is incorrect. 132Dec¶127

VII.14 Claim 13

Claim 13 reads as follows:

13. A method for encoding software code using a computer having a processor and memory, comprising:
storing a software code in said memory;
wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system; and
modifying, by said computer, using a first license key and an encoding algorithm, said software code, to form a modified software code; and
wherein said modifying comprises encoding said first code resource to form an encoded first code resource;
wherein said modified software code comprises said encoded first code resource, and a decode resource for decoding said encoded first code resource;
wherein said decode resource is configured to decode said encoded first code resource upon receipt of said first license key. 132Dec¶128

VII.15 Contrary to the NFOA, Beetcher does not disclose Claim 13's “wherein said modified software code comprises said encoded first code resource, and a decode resource for decoding said encoded first code resource”

The NFOA page 35:15 to page 16:3 alleges the claimed decode resource of the modified software code, for decoding said encoded first code resource, corresponds to the elements inside the dashed perimeter of the copy of Beetcher Fig. 4 in paragraph 96 of the Silva declaration.

Those elements consist of; unlock decode key **430**; exception handler **432**; check lock **422**. 132Dec¶129

The NFOA is wrong, and the Silva declaration's conclusion is wrong. Moreover, the conclusion does not follow from the Silva declaration's statements paragraph 96's statements of fact. Paragraph 96 of the Silva declaration states:

... Beetcher discloses that executing a trigger 301 invokes check lock function 422, which results in accessing "unlock (decode key)" function 430 upon confirmation that the customer possesses the software's license key.

Based upon these statements, Mr. Silva concludes in paragraph 96 that "Beetcher explains that its modified software code includes a decode resource for decoding the encoded first code resource." 132Dec¶130

Mr. Silva does not state any fact showing decoding of Beetcher's installed software module 300. And Mr. Silva and the NFOA are wrong. 132Dec¶131

First, Mr. Silva and the NFOA are wrong in concluding that check lock function 422 is part of installed software module 300. Beetcher, Fig. 4, clearly shows that check lock function 422 is microcode and therefore not part of installed software module 300. Moreover, Beetcher expressly states that this microcode is hardware, not software, and therefore could not be part of an installed Software module. See Beetcher col. 7:16-31 which states:

Horizontal microcode 402 contains microcode entries interpreting the executable instruction set. It is physically stored in control store 103, which in the preferred embodiment is a **read-only memory (ROM)** which is not capable of alteration by the customer. **Entries in horizontal microcode support get machine key 420, set lock 421, and check lock 422 functions.** Get machine key function 420 fetches a unique identifier, which in the preferred embodiment is based on the system serial number, from some permanent hardware location. Set lock function 421 accesses product lock table 460 and alters an entry in the table. The set lock function is the only microcode function capable of altering product lock table 460. Check lock function 422 accesses product lock table 460 and reads one of the entries to verify entitlement. 132Dec¶132

"Microcode" refers to "microcode is a layer of hardware-level instructions that implement higher-level machine code instructions," as explained in the definition of Microcode in Wikipedia, Attachment 15. Beetcher does not disclose including in its software module 300 hardware-level instructions that implement higher-level machine code instructions. Microcode is not part of Beetcher's installed software module 300. 132Dec¶133

Second, Mr. Silva and the NFOA are wrong in concluding that exception handler 432 is part of the installed software module 300, and they are wrong in concluding that exception handler 432 is part of a decode resource for decoding said encoded first code resource of the installed software. 132Dec¶134

Beetcher explains that function of exception handler 432 as follows. 132Dec¶135

First, Beetcher Fig. 4 shows only one arrow pointing into exception handler 432, which

identifies the one source of input to exception handler 432 to be microcode check lock 422. Beetcher col. 7:43-46 states “Exception handler 432 responds to exception conditions raised by functions in horizontal microcode 402.” ” Fig. 7 and this passage show that exception handler 432 receives input only from microcode check lock function 422. 132Dec¶136

Third, Beetcher col. 10:20-41 states that ‘Upon first execution of a previously unentitled software product, an exception is generated by the system when a triggering instruction is encountered. **Exception handling routine 432 then calls unlock routine 430** to attempt to unlock the product at step 920. Unlock routine 430 then fetches the encrypted entitlement key from the appropriate entry in encoded product key table 450 at step 921, obtains the 30 machine key at step 922, and **decodes the entitlement key** at step 923. ... **The triggering instruction is then retried** and program execution continues at step 928. If entitlement is not indicated at step 924, program execution aborts at step 925.132Dec¶137

This passage specifies that exception handling routine 432 calls unlock routine 430. It does not specify that exception handling routine 432 decodes the code of software module 300. Moreover, decoding [sic; decrypting] encrypted entitlement key 111 is not decoding software module 300. This is because encrypted entitlement key 111 is not part of software module 300. Furthermore, retrying triggering instruction 301 (“**The triggering instruction is then retried** and program execution continues...””) is not decoding the triggering instruction. Retrying merely means executing the instruction again to see if the updated entry 601 will result in entitlement of the program to run. 132Dec¶138

Finally, Mr. Silva and the NFOA are wrong in concluding that “unlock (decode key)” 430 is part of a decode resource for decoding the modified software code. 132Dec¶139

Beetcher described unlock routine 430 at col. 7:38-42, stating “Unlock routine 430 uses the unique machine key to decodes [sic] entitlement key 111, and stores encrypted entitlement key 111 in encoded product key table 450.” As explained herein above in the discussion of claim 11, unlock routing 430 stores license information contained in the encrypted entitlement key in memory, in tables 450, 460. Unlock routine 430 does not act on or modify instruction 301. Accordingly, unlock routine 430 is not a “decode resource for decoding” Beetcher’s instruction 301. Accordingly, Mr. Silva and the NFOA are wrong. 132Dec¶140

VIII.16 Claim 14

Claim 14 reads:

14. A method for encoding software code using a computer having a processor and memory, comprising:
storing a software code in said memory;
wherein said software code defines software code interrelationships between code resources that result in a specified underlying functionality when installed on a computer system;
and
encoding, by said computer using at least a first license key and an encoding algorithm, said software code,
to form a first license key encoded software code in which at least one of said software code interrelationships are encoded. 132Dec¶141

VIII.16 Contrary to the NFOA, Beetcher Does Not Anticipate Claim 14

Claim 14 recites “encoding, by said computer using at least a first license key and an encoding algorithm, said software code.” 132Dec¶142

For the same reasons stated for claim 12, this recitation defines key based encoding; using the key as in input to the encoding function, to encode software code. 132Dec¶143

For the same reasons discussed for claim 12, Beetcher does not disclose this encoding, and therefore does not disclose the claimed “first license key encoded software code” encoding at least one “code interrelationship.” 132Dec¶144

IX. The Rejections Based Upon Cooperman

The rejections of claims 11, 12, and 13, under 102(e), based upon Cooperman, should be withdrawn because Cooperman has no 102(e) date. MPEP706.02(f)(1)(C)(3)(b); see discussion above.

To put it in plain language, Cooperman is a publication of a PCT application. Under applicable law, a PCT application is unavailable under 35 USC 102, as of its filing date. This was a consequence of the US PCT reservations when the US joined the PCT, which precluded a PCT from entitlement to a 102(e) and the concurrent amendment to 35 USC 363 which excepted PCT applications from having 102(e) effect.

At the time relevant to Cooperman, the US law did not accord a PCT application prior art status under 102 based upon the PCT application’s filing date. Subsequent revisions of US law did, but only for PCT application filed after a critical date. MPEP706.02(f)(1)(C)(3)(b) provides the instruction to the examining corps to not assert a PCT application as prior art under 102(e) for that reason. That section of the MPEP and applicable US law upon which it is based apply to Cooperman, because Cooperman, is WO97/26732, which is the international publication of to PCT/US97/00651, filed 1/16/1997. For PCT applications filed 1/16/1997, US law provides no 102(e) date.

Cooperman has no 102(e) date and therefore the 102(e) rejection based upon Cooperman is improper and must be withdrawn.

X. The Rejections Based Upon Hasebe

The rejections of claims 12, 13, 14, under 102(e), based upon Hasebe, should be withdrawn because Hasebe's 102(e) date is after the date of invention of the rejected claims.

Mr. Moskowitz's 37 CFR 1.131 declaration submitted with this response shows that he constructively reduced to practice inventions, including those defined by claims 12, 13, and 14, by pre-filing activity followed by filing the disclosure in 08/587,943, filed Jan. 17, 1996, which issued 4/28/1998, as USP5745943.

Mr. Moskowitz is named as an inventor on over 100 US patents directed to digital and computer technologies. 131Dec¶1.

Mr. Moskowitz identifies pre filing conception and drafting activities. 131Dec¶¶2-7.

Mr. Moskowitz shows actual reduction to practice on 1/3/1996. 131Dec¶¶9-11.

Mr. Moskowitz shows that his **08/587,943 application filed 1/17/1996**, issued as a patent, and provides written description support for claims 11-14 issued in USP 9104842. 131Dec¶¶12-16.

Application 08/587,943 issued as a patent, and claims 11-14 subsequently issued as a

patent. That is all that is required to show reduction to practice pursuant to rule 131.

Accordingly, Hasebe is not prior art and therefore the rejections based upon Hasebe under 102(a) should be withdrawn.

XI. Further Comments on Statements in the NFOA

XI.1 Regarding the NFOA's Understanding of Beetcher

The patent owner and Mr. Moskowitz's 132 declaration explain herein above why Beetcher does not anticipate. However, it may be helpful to review the following statements in the NFOA and patent owner's responses thereto, to clarify apparent misunderstandings in the NFOA, of Beetcher.

The NFOA, page 103:1-5, referring to Beetcher and the reexam request, states that:

In response, the Examiner respectfully submits that the Request rather shows encoding the software code utilizing information from the key encoded as triggers, within the encoded software code. This code is the original code encoded so as to preclude usage of modules until a proper entitlement key is provided to unlock the original code (see column 9, lines 1-20 and column 4, lines 3-46).

The statement "preclude usage of modules until a proper entitlement key is provided to unlock the original code" is not accurate. Beetcher's code runs when loaded. However, it is designed to terminate execution each time instruction 301 executes. Beetcher explains that if the product lock table entry 601 fails to store a version number at least as great as the version number stored in instruction 301, Beetcher's instruction 301 in module 300 causes Beetcher's software module 300 terminates. That is what step 1006, 1007, Fig. 10 show, and how they are explained at col. 10:54-65. Beetcher does not require a change in form or representation of instruction 301; instead Beetcher discloses merely the instruction 301 is executed. Hence, Beetcher does not disclose "decoding" instruction 301.

The NFOA, page 106:2-8, referring to Beetcher, states that (bold added for emphasis):

First, the Reexam request explains that the delivered software module is encoded to include verification triggers based upon key information. Then upon being supplied with an enablement key the encoded software is **decoded to reveal the original code in its original executable form**. The encoded software module is transmitted along with a product lock table showing correspondence between keys and the sub-modules they unlock (see column 8, lines 53-67), where the Request describes this unlock process and what resident elements carry it out along with the transmitted table.

There is no basis for the statement that the reexam request discloses that "software is **decoded to reveal the original code in its original executable form**." That statement is also not what Beetcher discloses. That statement is wrong.

Beetcher discloses software module 300 comprising object code; a series of instructions that customer computer 101 can execute. Some of those instructions are instruction 301.

According to Beetcher, software module 300 is executed, not decoded. The NFOA assertion references Beetcher col. 8:53-67 as supporting its conclusion. Beetcher col. 8:53-67 does not support the NFOA's conclusion.

Beetcher col. 8:53-67 states in part that the "computer system 101 receives, decodes and stores **entitlement key 111**." But that decoding is not decoding of the software module.

Beetcher col. 8:53-67 states that "the software module executing on system 101 causes system 101 to **verify entitlement** upon encountering an entitlement verification instruction." And that verification is not decoding of the software module. That verification refers to execution of verification triggering instructions 301, as explained at col. 10:51-65; and shown at steps 1004 to 1007 in Fig. 10.

XI.2 Regarding Cooperman

The NFOA, page 91:1-4 states:

Patent Owner has further not established common inventorship of the subject Patent with Cooperman or Moskowitz et al. (USPN 5,745,569). Patent Owner's statements asserting the commonality between the instant claims, USPN 5,745,569, and applied WO 97/26732 are insufficient alone.

That statement is incorrect. Mr. Moskowitz is named as an inventor on both USP 5,745,569 and USP 9104842. Thus, there is an inventor in common. That is common inventorship. The NFOA presumably meant that the inventorship named in USP 5,745,569 and USP 9104842 are not identical, is true.

XI.3 Regarding Mr. Marc Cooperman

The NFOA, at page 91:5-15 states that:

Patent Owner argues that "If Mr. Cooperman refuses to cooperate, then that fact would also indicate he is not an inventor of the subject matter defined by claims 11-14 in USP9104842."

In response, the Examiner respectfully submits that Mr. Cooperman has been cooperating by the Patent Owner's own admission, as noted on page 1 of the Patent Owner Statement: "Mr. Cooperman recently testified in the related district court litigation and the transcript of his testimony suggests that he was not an inventor of at least some of the subject matter defined by claims in the subject patent, USP9104842." It is just that the testimony of Mr. Cooperman didn't serve the purpose of attaching him to the subject Patent. Additionally someone's refusal to cooperate does not alone indicate inventorship or preclude inventorship.

With due respect, the examiner's conclusion that the fact that Mr. Cooperman recently testified in a court proceeding is an admission that Mr. Cooperman is cooperating with the patent owner is a non sequitur. Exhibit 3 clearly shows that Mr. Cooperman's testimony was taken by

“defendant” Juniper Networks, Inc. Exhibit 3, page 1. And that page also shows that the deposition was pursuant to a “Notice”, presumably an subpoena. Mr. Cooperman’s testimony was not at the request of, and not for the benefit of patent owner, and is not an admission that Mr. Cooperman “has been cooperating” with patent owner.

It is true that the long ago settlement agreement in Attachment 7, Section 2, obligated Mr. Cooperman to sign certain assignment documents and otherwise assigned all rights relating to IP in that dispute to the patent owner. However, the undersigned has repeatedly emailed and telephoned Mr. Cooperman, over a dozen times between the summer of 2018 and the present, requesting that he determine if he believes himself to be an inventor of any claim in USP9104842, and notify the patent owner of that determination. In fact, the undersigned emailed short alternative statements to Mr. Cooperman, for his signature and return. One form stating he was an inventor and one form stating he was not an inventor. To date, Mr. Cooperman has not returned a signed form. (If Mr. Cooperman does notify the undersigned that he believes himself to be an inventor of any claim in SP9104842 during the pendency of this reexam, the undersigned will at a minimum notify the examiner of that fact.)

Mr. Cooperman long ago surrendered any rights he might have in USP9104842 to the patent owner, pursuant to the settlement agreement in attachment 7. Therefore, whether or not Mr. Cooperman should be named as inventor is not relevant to ownership.

Given his lack of response, to date, on this issue, to say Mr. Cooperman “has been cooperating by the Patent Owner” is not correct.

XI.4 Regarding Hasebe’s Prior Art Date

The NFOA page 91:16 to 92:3 states that:

It is further unclear, even given the Patent Owner's reasoning, how the argued "proof" that the Patent Under reexamination is entitled to a date equal to that of W097/26732 (1/17/1996), that this would throw out Hasebe et al. as prior art given Hasebe's claims to foreign priority back to 8/31/1995. Patent Owner has not submitted an appropriate oath or declaration to establish invention of the subject matter prior to the effective date of the reference (37 CFR 1.131).

The NFOA’s conclusion that “Hasebe's claims to foreign priority back to 8/31/1995” is relevant to Hasebe’s prior art date is incorrect. Hasebe’s Paris priority date does not entitle Hasebe to a 35 USC 102(e) date. See for example *In re McKellin*, 529 F.2d 1324, ____, 188 USPQ 428, 434 (CCPA 1976)(expanded panel)

With respect to section 102(e), the effective date of the disclosure of the Maltha patent vis-a-vis an application filed in the United States is the United States filing date of Maltha. *In re Hilmer*, 53 CCPA 1288, 1318-20, 359 F.2d 859, 882-83, 149 USPQ 480, 499-500 (1966), hereinafter Hilmer (I). The United States filing date of the Maltha patent is subsequent to appellants' effective date. Therefore the disclosure of the Maltha patent cannot be prior art under section

102(e). [In re McKellin, 529 F.2d 1324, ____, 188 USPQ 428, 434 (CCPA 1976)(expanded panel).]

And 102(g) does not apply because there is no interference between Hasebe and the USP9104842. Accordingly, Hasebe's claims to foreign priority back to 8/31/1995 do not provide Hasebe a prior art date earlier than the US application date of Hasebe.

Truly, /RichardNeifeld/
Richard Neifeld, Ref. No. 35,299
Attorney of record, for patentee

Y:\Clients\SCOT Scott A Moskowitz and Wistaria Trading, Inc\90014138, USP9104842,
SCOT0014-4\Drafts\2019-01-28_ResponseToNFOA_90014138.wpd



what does encoding mean in the computer



Sign in

All News Images Shopping Videos More Settings Tools

About 117,000,000 results (0.49 seconds)

In **computers**, **encoding** is the process of putting a sequence of characters (letters, numbers, punctuation, and certain symbols) into a specialized format for efficient transmission or storage. Decoding is the opposite process -- the conversion of an **encoded** format back into the **original** sequence of characters. Nov 14, 2005

What is encoding and decoding? - Definition from WhatIs.com
<https://searchnetworking.techtarget.com/definition/encoding-and-decoding>

About this result Feedback

People also ask

- What do you mean by encoding? ▼
- What are the 3 types of encoding? ▼
- What is an example of encoding? ▼
- How is encoding done? ▼

Feedback

What is Encoding? - Definition from Techopedia
<https://www.techopedia.com/definition/948/encoding>

In **computer technology**, **encoding** is the process of applying a specific code, such as letters, symbols and numbers, to data for **conversion** into an equivalent cipher. In electronics, **encoding** refers to analog to digital conversion.

What is encoding and decoding? - Definition from WhatIs.com
<https://searchnetworking.techtarget.com/definition/encoding-and-decoding>

Nov 14, 2005 - In **computers**, **encoding** is the process of putting a sequence of characters (letters, numbers, punctuation, and certain symbols) into a specialized format for efficient transmission or storage. **Decoding** is the opposite process -- the conversion of an **encoded** format back into the original **sequence** of characters.

Encoding Definition - The Tech Terms Computer Dictionary
<https://techterms.com/definition/encoding>

Sep 23, 2010 - **Encoding** is the process of converting data from one form to another. While "**encoding**" can be used as a verb, it is often used as a noun, and ...

What is Encoding? Webopedia Definition
<https://www.webopedia.com/TERM/E/encoding.html>

In **computer technology**, **encoding** is the process of putting a sequence of characters into a special format for transmission or storage purposes.

Who does encoding and decoding in a computer? - Quora
<https://www.quora.com/Who-does-encoding-and-decoding-in-a-computer>

Apr 10, 2015 - In **Computer Science**, **encoding** is the process of putting a sequence of characters (letters, numbers, punctuation, and certain symbols) into a specialized format for efficient transmission or storage. **Decoding** is the opposite process -- the conversion of an **encoded** format back into the original sequence of characters.

- What does decode mean in computer science? May 12, 2018
 - What is it meant by **encoding** and decoding in a computer science ... Sep 10, 2017
 - Why is encoding needed? Dec 24, 2014
- More results from www.quora.com

encoding system Definition from PC Magazine Encyclopedia
<https://www.pcmag.com/encyclopedia/term/42501/encoding-system>

PC Magazine Tech Encyclopedia Index - Definitions on common technical and computer related terms.

Why is encoding and decoding needed for any programming language ...
<https://stackoverflow.com/.../why-is-encoding-and-decoding-needed-for-any-program...>

2 answers

Apr 30, 2017 - **Computer can only deal with bytes so Encoding and decoding is necessary. ...** from <http://searchnetworking.techtarget.com/definition/encoding-and-decoding..>

What is base 64 encoding used for? 16 answers Mar 11, 2015

What is the difference between encode/decode? 7 answers Dec 21, 2014

What is the real purpose of Base64 encoding? 3 answers Apr 25, 2012

Difference between encoding and encryption 6 answers Nov 19, 2011

More results from stackoverflow.com

What is Encode? - Computer Hope

<https://www.computerhope.com> - Dictionary > E - Definitions

Apr 10, 2017 - **Computer dictionary definition for what encode** means including related links, information, and terms.

Character encoding - Wikipedia

https://en.wikipedia.org/wiki/Character_encoding

Character **encoding** is used to represent a repertoire of characters by some kind of **encoding** ... The low cost of digital representation of data in modern **computer** systems allows more elaborate character codes (such as Unicode) which **Definition from The Tech Terms Dictionary**; ^ Tom Henderson (17 April 2014). "Ancient ...

Character encodings for beginners - World Wide Web Consortium

<https://www.w3.org/International/questions/qa-what-is-encoding>

What is a character **encoding**, and why should I care? ... Not only **does** lack of character **encoding** information **spoil** the readability of displayed text, but it may **mean** that your ... The characters **are** stored in the **computer** as one or more bytes .

Searches related to what does encoding mean in the computer

- encoding definition in communication
- encoding definition speech
- decoding definition
- encoding examples
- difference between encoding and decoding
- encoding function
- encoding definition psychology
- computer coding decoding

1 2 3 4 5 6 7 8 9 10 Next

22312, Springfield, VA - From your Internet address - Use precise location - Learn more

Help Send feedback Privacy Terms

[Home](#) > [Open source](#)

Search the TechTarget Network

DEFINITION

encoding and decoding

Posted by: [Margaret Rouse](#) [WhatIs.com](#)

In computers, encoding is the process of putting a sequence of [characters](#) (letters, numbers, punctuation, and certain symbols) into a specialized format for efficient transmission or storage. Decoding is the opposite process -- the conversion of an encoded format back into the original sequence of characters. Encoding and decoding are used in data communications, networking, and storage. The term is especially applicable to radio ([wireless](#)) communications systems.

The code used by most computers for text files is known as [ASCII](#) (American Standard Code for Information Interchange, pronounced ASK-ee). ASCII can depict uppercase and lowercase alphabetic characters, numerals, punctuation marks, and common symbols. Other commonly-used codes include [Unicode](#), [BinHex](#), [Uuencode](#), and [MIME](#). In data

Attachment 12 Page 1 of 7

communications, Manchester encoding is a special form of encoding in which the binary digits (bits) represent the transitions between high and low logic states. In radio communications, numerous encoding and decoding methods exist, some of which are used only by specialized groups of people (amateur radio operators, for example). The oldest code of all, originally employed in the landline telegraph during the 19th century, is the Morse code.

The terms encoding and decoding are often used in reference to the processes of analog-to-digital conversion and digital-to-analog conversion. In this sense, these terms can apply to any form of data, including text, images, audio, video, multimedia, computer programs, or signals in sensors, telemetry, and control systems. Encoding should not be confused with encryption, a process in which data is deliberately altered so as to conceal its content. Encryption can be done without changing the particular code that the content is in, and encoding can be done without deliberately concealing the content.



Margaret Fouse asks:

What type of text encoding do you use - ASCII, Unicode, MIME or something else



[Join the Discu](#)

This was last updated in November 2005

Continue Reading About encoding and decoding

Attachment 12 Page 2 of 7

- J. J. Weimer outlines encoding and decoding for data transmission from the Mac user's point-of-view.
- Hughs Ominous Valve Works discusses encoding and decoding methods, from traditional to exotic, used in radio communications.

Related Terms

ONOS (Open Network Operating System)

ONOS (Open Network Operating System) is an operating system (OS) designed for network service providers to help build ... See complete definition

OpenFlow Consortium



The OpenFlow Consortium was a collaborative group of university researchers and network administrators that developed the ... See complete definition



Trema


Trema is an open source framework for developing OpenFlow controllers for software-defined networking in the Ruby and C ... See complete definition

➤ Dig Deeper on Open source networking

ALL NEWS GET STARTED EVALUATE MANAGE PROBLEM SOLVE

 **Cumulus Linux, NetQ available for Lenovo RackSwitch**  **ONAP Casablanca release targets cross-carrier deployment, 5G**

 **Pica8 wooing campus with white box network switch software**  **Helping enterprises adapt to open source switching**

 **Join the conversation**  **7 comments**

Share your comment

Send me notifications when other members comment.

Add My Comment

Oldest ▼

[] Margaret Rouse - 14 Nov 2005 7:29 AM

What type of text encoding do you use - ASCII, Unicode, MIME or something else?

[] fahid786 - 4 Mar 2014 6:24 PM

No video sellite

[] manjibu - 5 Mar 2014 8:57 AM

UTF8 mostly

[] trying2learN - 17 Mar 2014 1:49 PM

I'm not sure, don't know where to look for it, I have a few word files that won't open and when it does it comes up with the code for it I believe that is what it is. I am trying to learn but am getting frustrated

[] GopikaGN - 1 May 2014 2:59 AM

ASCII code

[] GopikaGN - 1 May 2014 3:01 AM

I don't about what type of coding is used by my system.

[] Ramesh223 - 28 May 2017 9:57 AM

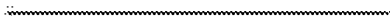
How an image is encoded in to bits?



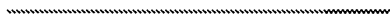
ADS BY GOOGLE



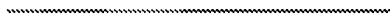
Latest TechTarget resources



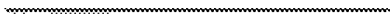
SDN



ENTERPRISE WAN



UNIFIED COMMUNICATIONS



MOBILE COMPUTING

SearchSDN



Interop 2014: New Broadcom processor boosts NFV deployments



The new multi-core Broadcom processor will enable better processing and agility when

DATA CENTER

IT CHANNEL

simplifying NFV and SDN deployments.



Meet five software-defined networking research rock stars

From programming languages to partial SDN deployments, current SDN research allows for true innovation in the field as proven by ...

About Us

Contributors

Features

Meet The Editors

Reprints

Guides

Contact Us

Archive

Opinions

Privacy Policy

Site Map

Photo Stories

Advertisers

Answers

Quizzes

Business Partners

Definitions

Tip

Media Kit

E-Products

Tutorials

Corporate Site

Events

Videos

All Rights Reserved. Copyright 2000 - 2019, TechTarget

ALERT [FREE WEBINAR | How to Continuously Monitor and Analyze MySQL and MariaDB with IDERA's SQL Diagnostic Manager \(https://www.techopedia.com/reg/how-to-continuously-monitor-and-analyze-mysql-and-mariadb-with-ideras-sql-diagnostic-manager/33743?utm_source=techopedia&utm_medium=hellobar\)](https://www.techopedia.com/reg/how-to-continuously-monitor-and-analyze-mysql-and-mariadb-with-ideras-sql-diagnostic-manager/33743?utm_source=techopedia&utm_medium=hellobar)

Object Code

Definition - What does *Object Code* mean?

Object code is produced when an interpreter or a compiler translates source code into recognizable and executable machine code.

Object code is a set of instruction codes that is understood by a computer at the lowest hardware level. Object code is usually produced by a compiler that reads some higher level computer language source instructions and translates them into equivalent machine language instructions.

[FREE WEBINAR | How to Continuously Monitor and Analyze MySQL and MariaDB with IDERA's SQL Diagnostic Manager \(https://www.techopedia.com/reg/how-to-continuously-monitor-and-analyze-mysql-and-mariadb-with-ideras-sql-diagnostic-manager/33743?utm_source=techopedia&utm_medium=textlinks\)](https://www.techopedia.com/reg/how-to-continuously-monitor-and-analyze-mysql-and-mariadb-with-ideras-sql-diagnostic-manager/33743?utm_source=techopedia&utm_medium=textlinks)

Techopedia explains *Object Code*

Just as human beings understand native languages, computers understand machine language, which is made up of object code. Software applications are built in multiple programming languages with a standard objective: to execute processes via a machine.

A compiler translates source code into object code, which is stored in object files. Object files contain object code that includes instructions to be executed by the computer. It should be noted that object files may require some intermediate processing by the operating system (OS) before the instructions contained in them are actually executed by the hardware.

Object file examples include common object file format (COFF), COM files and ".exe" files.

Tech moves fast! Stay ahead of the curve with Techopedia!

Join nearly 200,000 subscribers who receive actionable tech insights from Techopedia.

x

Sponsored Content

Recommended by

20 NBA Players Who Quietly Retired And Are Now Working 9 To 5
Bleacher Breaker

[http://www.bleacherbreaker.com/nba-players-are-working-9-5-jobs-since-retiring/?utm_campaign=rlk-d-us-n-es-0-190206-bb-ob-a4-a1&utm_term=1_nfv.jpg&utm_source=ob&utm_medium=\\$section_id\\$-ob&utm_content=00f](http://www.bleacherbreaker.com/nba-players-are-working-9-5-jobs-since-retiring/?utm_campaign=rlk-d-us-n-es-0-190206-bb-ob-a4-a1&utm_term=1_nfv.jpg&utm_source=ob&utm_medium=$section_id$-ob&utm_content=00f)

Little Known Trick To Avoid Gutter Cleaning For Life And Increase Home Value
leafilterguards.com

https://www.leafilterguards.com/aff_g2?offer_id=2024&aff_id=2364&url_id=6098&source=UB&aff_id=6098

5 Self Tanners That Will Blow Your Mind
Tan Physics

http://tanphysics.com/info/tan-physics-5-self-tanners-that-will-blow-your-mind/?utm_source=USA&utm_medium=UB&utm_campaign=UB&utm_term=JAS211&utm_content=MAY18-GRP1-H248-JAS211&utm_term=2620037

This Is What Vikings Were Actually Like. The Photos Are Strikingly Scary
Past Factory

[http://www.pastfactory.com/2018/08/14/the-real-truth-behind-the-viking-culture/?utm_campaign=vkg-j-us-c-0-0-181116-pa-ob-a4-a1&utm_term=2_vkg.jpg&utm_source=ob&utm_medium=\\$section_id\\$-ob&utm_content=00f](http://www.pastfactory.com/2018/08/14/the-real-truth-behind-the-viking-culture/?utm_campaign=vkg-j-us-c-0-0-181116-pa-ob-a4-a1&utm_term=2_vkg.jpg&utm_source=ob&utm_medium=$section_id$-ob&utm_content=00f)

The Must-Play City Building Game of the Year
Forge Of Empires

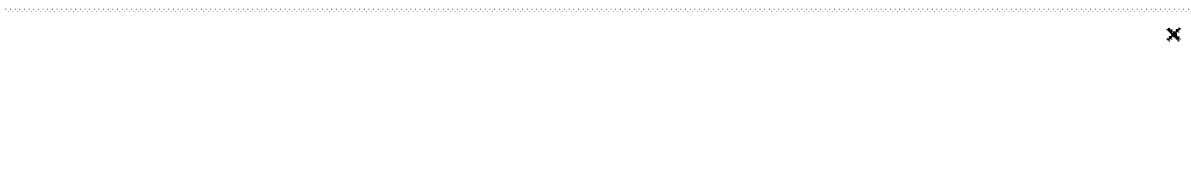
<https://om.forgeofempires.com/foe/us?ref=out-us-2019&pid=5>

Man Buys Russian Tank on Ebay. Then He Checks Inside And Realizes He's A Millionaire
Shoot Old School

http://breaking.com/tank-treasure/?utm_campaign=TankTreasur

Featured Q&A

More of your questions answered by our Experts (/experts)



Enter a term...

Advanced
Search

MAIN BROWSE TERMS DID YOU KNOW? REFERENCE ALL CATEGORIES STUDY GUIDES BLOG SLIDESHOWS SPONSORED

Unlimited feature flags are here



Try now

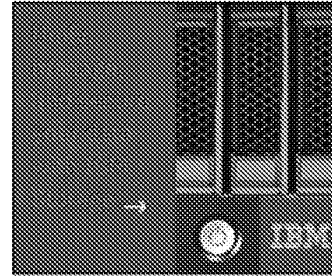
Main > TERM > R >

routine

A section of a program that performs a particular task. Programs consist of modules, each of which contains one or more routines. The term *routine* is synonymous with procedure, function, and subroutine.

Related Terms

- user defined function
- XML Schema Definition - XSD
- DDL
- software-defined services
- SDD
- Software-Defined Everything - SDE
- SOS - software-defined storage
- SDS - high definition Multimedia interface
- parameter
- high definition port



WEBOPEDIA NEWS

Stay up to date on the latest developments in Internet terminology with a free newsletter from Webopedia... Join to subscribe now

Email address

Subscribe Now

IT Solutions Builder

TOP IT RESOURCES TO MOVE YOUR BUSINESS FORWARD

Which topic are you interested in?

Mobile Security Networks IoT Cloud Data Storage

Applications Development IT Management Other

LATEST ARTICLES

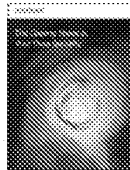
Facts about IT & Coding Boot Camps
The following coding and IT boot camp facts and statistics provide an introduction to the changing trends in education and training programs. [Read More >](#)

Top Cloud Computing Facts
The following facts and statistics capture the changing landscape of cloud computing and how service providers and customers are keeping up with... [Read More >](#)

Sponsored Article
[How to pay off your house ASAP \(It's so simple\)!](#) by [LendingTree](#)

Americans could save thousands by taking advantage of this government program [Read More >](#)

Related White Papers and Webcasts



Sponsored content
Top Three Gaping Holes in Your Data Security

By [globetecape](#)
February 8, 2019

Your IT team understands the importance of securing the network against data breaches from external threats. Regardless of your efforts, data breaches usually begin from inside your network, often due to lax or unenforced security policies. "Did you know?" Insider threats are now the most prevalent cause of data incidents, representing nearly 70% of all data leaks. Are you doing enough? [Continue reading ...](#)

PREVIOUS
[router firmware](#)
NEXT
[tolling](#)

Your IT team understands the importance of securing the network against data breaches from external threats. Regardless of your efforts, data breaches usually begin from inside your network, often due to lax or unenforced security policies ... [Continue reading ...](#)
Sponsored by [globetecape](#)

Related White Papers and Webcasts

It's Time to Rethink CRM


Today's CRM systems have the ability to deliver more than just lead generation. If leveraged correctly, they can be major drivers in loyalty and relationship management. But many companies have yet to unlock their true potential. Learn how you can use CRM to facilitate better communications and grow your business... [Continue Reading ...](#)




[Download](#)

Ready Your Enterprise for the API Revolution


APIs are changing more than just software architectures. From planning through implementation and beyond, an API-driven business model brings a host of new challenges and opportunities to organizations. Download this eBook to learn how to get the most benefit from these agents of change... [Continue Reading...](#)



[Download](#)



Try feature flags at no cost
Get started and create your first flag in minutes



- STUDY GUIDES**
- [Java Basics, Part 1](#)
Java is a high-level programming language. This guide describes the basics of Java, providing an overview of syntax, variables, data types and... [Read More >](#)
 - [Java Basics, Part 2](#)
This second Study Guide describes the basics of Java, providing an overview of operators, modifiers and control structures. [Read More >](#)
 - [Network Fundamentals Study Guide](#)
Networking fundamentals teaches the building blocks of modern network design. Learn different types of networks, concepts, architecture and... [Read More >](#)



Apigee API Management [>](#)

Named a leader in the 2018 Gartner Magic Quadrant. [LEARN MORE](#)
Download the report today.

Browse Technology Definitions:

- A
- B
- C
- D
- E
- F
- G
- H
- I
- J
- K
- L
- M
- N
- O
- P
- Q
- R
- S
- T
- U
- V
- W
- X
- Y
- Z
- 0
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9

Enter a term...

Definitions:
[Recent Terms](#)
[Previous Terms](#)
[Recent Terms](#)

- All Categories
 Articles
 Blog
 ERM You Know
 Cloud Reference
 Study Guides
 Siderbars



An eWEEK Property

[Terms of Service](#) | [License & Reprints](#) | [About Us](#) | [Privacy Policy](#) | [Contact Us](#) | [Advertise](#) | [Sitemap](#)
Copyright 2019 QuantStreet Inc., All Rights Reserved.

Advertiser Disclosure: Some of the products that appear on this site are from companies from which QuantStreet receives compensation. This compensation may impact how and where products appear on this site including, for example, the order in which they appear. QuantStreet does not include all companies or all types of products available in the marketplace.

WIKIPEDIA

Microcode

Microcode is a computer hardware technique that imposes an interpreter between the CPU hardware and the programmer-visible instruction set architecture of the computer.^[6] As such, the microcode is a layer of hardware-level instructions that implement higher-level machine code instructions or internal state machine sequencing in many digital processing elements. Microcode is used in general-purpose central processing units, although in current desktop CPUs it is only a fallback path for cases that the faster hardwired control unit cannot handle.^[4]

Microcode typically resides in special high-speed memory and translates machine instructions, state machine data or other input into sequences of detailed circuit-level operations. It separates the machine instructions from the underlying electronics so that instructions can be designed and altered more freely. It also facilitates the building of complex multi-step instructions, while reducing the complexity of computer circuits. Writing microcode is often called **microprogramming** and the microcode in a particular processor implementation is sometimes called a **microprogram**.

More extensive microcoding allows small and simple microarchitectures to emulate more powerful architectures with wider word length, more execution units and so on, which is a relatively simple way to achieve software compatibility between different products in a processor family.

Some hardware vendors, especially IBM, use the term *microcode* as a synonym for *firmware*. In that way, all code within a device is termed *microcode* regardless of it being microcode or machine code; for example, hard disk drives are said to have their *microcode* updated, though they typically contain both microcode and firmware.^[5]

Contents

- Overview
- Justification
 - Benefits
- History
- Examples
- Implementation
 - Horizontal microcode
 - Vertical microcode
- Writable control store
- Comparison to VLIW and RISC
- Micro Ops
- See also
- Notes
- References
- Further reading
- External links

Overview

The lowest layer in a computer's software stack is traditionally raw binary machine code instructions for the processor. Microcode sits one level below this. To avoid confusion, each microprogram-related element is differentiated by the *micro* prefix: microinstruction, microassembler, microprogrammer, microarchitecture, etc.

Engineers normally write the microcode during the design phase of a processor, storing it in a read-only memory (ROM) or programmable logic array (PLA)^[4] structure, or in a combination of both.^[5] However, machines also exist that have some or all microcode stored in SRAM or flash memory. This is traditionally denoted as *writable control store* in the context of computers, which can be either read-only or read-write memory. In the latter case, the CPU initialization process loads microcode into the control store from another storage medium, with the possibility of altering the microcode to correct bugs in the instruction set, or to implement new machine instructions.

Complex digital processors may also employ more than one (possibly microcode-based) control unit in order to delegate sub-tasks that must be performed essentially asynchronously in parallel. A high-level programmer, or even an assembly programmer, does not normally see or change microcode. Unlike machine code, which often retains some backward compatibility among different processors in a family, microcode only runs on the exact electronic circuitry for which it is designed, as it constitutes an inherent part of the particular processor design itself.

Microprograms consist of series of microinstructions, which control the CPU at a very fundamental level of hardware circuitry. For example, a single

typical *horizontal* microinstruction might specify the following operations:

- Connect register 1 to the A side of the ALU
- Connect register 7 to the B side of the ALU
- Set the ALU to perform two's-complement addition
- Set the ALU's carry input to zero
- Store the result value in register 8
- Update the condition codes from the ALU status flags (*negative, zero, overflow, and carry*)
- Microjump to microPC nnn for the next microinstruction

To simultaneously control all processor's features in one cycle, the microinstruction is often wider than 50 bits; e.g., 128 bits on a 360/85 with an emulator feature. Microprograms are carefully designed and optimized for the fastest possible execution, as a slow microprogram would result in a slow machine instruction and degraded performance for related application programs that use such instructions.

Justification

Microcode was originally developed as a simpler method of developing the control logic for a computer. Initially, CPU instruction sets were hardwired. Each step needed to fetch, decode, and execute the machine instructions (including any operand address calculations, reads, and writes) was controlled directly by combinational logic and rather minimal sequential state machine circuitry. While very efficient, the need for powerful instruction sets with multi-step addressing and complex operations (see below) made such hard-wired processors difficult to design and debug; highly encoded and varied-length instructions can contribute to this as well, especially when very irregular encodings are used.

Microcode simplified the job by allowing much of the processor's behaviour and programming model to be defined via microprogram routines rather than by dedicated circuitry. Even late in the design process, microcode could easily be changed, whereas hard-wired CPU designs were very cumbersome to change. Thus, this greatly facilitated CPU design.

From the 1940s to the late 1970s, a large portion of programming was done in assembly language; higher-level instructions mean greater programmer productivity, so an important advantage of microcode was the relative ease by which powerful machine instructions can be defined. The ultimate extension of this are "Directly Executable High Level Language" designs, in which each statement of a high-level language such as PL/I is entirely and directly executed by microcode, without compilation. The IBM Future Systems project and Data General Fountainhead Processor are examples of this. During the 1970s, CPU speeds grew more quickly than memory speeds and numerous techniques such as memory block transfer, memory pre-fetch and multi-level caches were used to alleviate this. High-level machine instructions, made possible by microcode, helped further, as fewer more complex machine instructions require less memory bandwidth. For example, an operation on a character string can be done as a single machine instruction, thus avoiding multiple instruction fetches.

Architectures with instruction sets implemented by complex microprograms included the IBM System/360 and Digital Equipment Corporation VAX. The approach of increasingly complex microcode-implemented instruction sets was later called CISC. An alternate approach, used in many microprocessors, is to use PLAs or ROMs (instead of combinational logic) mainly for instruction decoding, and let a simple state machine (without much, or any, microcode) do most of the sequencing. The MOS Technology 6502 is an example of a microprocessor using a PLA for instruction decode and sequencing. The PLA is visible in photomicrographs of the chip,^[6] and its operation can be seen in the transistor-level simulation.

Microprogramming is still used in modern CPU designs. In some cases, after the microcode is debugged in simulation, logic functions are substituted for the control store. Logic functions are often faster and less expensive than the equivalent microprogram memory.

Benefits

A processor's microprograms operate on a more primitive, totally different, and much more hardware-oriented architecture than the assembly instructions visible to normal programmers. In coordination with the hardware, the microcode implements the programmer-visible architecture. The underlying hardware need not have a fixed relationship to the visible architecture. This makes it easier to implement a given instruction set architecture on a wide variety of underlying hardware micro-architectures.

The IBM System/360 has a 32-bit architecture with 16 general-purpose registers, but most of the System/360 implementations actually use hardware that implemented a much simpler underlying microarchitecture: for example, the System/360 Model 30 has 8-bit data paths to the arithmetic logic unit (ALU) and main memory and implemented the general-purpose registers in a special unit of higher-speed core memory, and the System/360 Model 40 has 8-bit data paths to the ALU and 16-bit data paths to main memory and also implemented the general-purpose registers in a special unit of higher-speed core memory. The Model 50 has full 32-bit data paths and implements the general-purpose registers in a special unit of higher-speed core memory.^[6] The Model 65 through the Model 195 have larger data paths and implement the general-purpose registers in faster transistor circuits. In this way, microprogramming enabled IBM to design many System/360 models with substantially different hardware and spanning a wide range of cost and performance, while making them all architecturally compatible. This dramatically reduces the number of unique system software programs that must be written for each model.

A similar approach was used by Digital Equipment Corporation (DEC) in their VAX family of computers. As a result, different VAX processors use

different microarchitectures, yet the programmer-visible architecture does not change.

Microprogramming also reduces the cost of field changes to correct defects (bugs) in the processor; a bug can often be fixed by replacing a portion of the microprogram rather than by changes being made to hardware logic and wiring.

History

In 1947, the design of the MIT Whirlwind introduced the concept of a control store as a way to simplify computer design and move beyond *ad hoc* methods. The control store is a (*hide matrix*): a two-dimensional lattice, where one dimension accepts "control time pulses" from the CPU's internal clock, and the other connects to control signals on gates and other circuits. A "pulse distributor" takes the pulses generated by the CPU clock and breaks them up into eight separate time pulses, each of which activates a different row of the lattice. When the row is activated, it activates the control signals connected to it.^[8]

Described another way, the signals transmitted by the control store are being played much like a player piano roll. That is, they are controlled by a sequence of very wide words constructed of bits, and they are "played" sequentially. In a control store, however, the "song" is short and repeated continuously.

In 1951, Maurice Wilkes enhanced this concept by adding *conditional execution*, a concept akin to a conditional in computer software. His initial implementation consisted of a pair of matrices: the first one generated signals in the manner of the Whirlwind control store, while the second matrix selected which row of signals (the microprogram instruction word, so to speak) to invoke on the next cycle. Conditionals were implemented by providing a way that a single line in the control store could choose from alternatives in the second matrix. This made the control signals conditional on the detected internal signal. Wilkes coined the term **microprogramming** to describe this feature and distinguish it from a simple control store.

Examples

- In common with many other complex mechanical devices, Charles Babbage's analytical engine uses banks of cams to control each operation. That is, it has a read-only control store. As such, it deserves to be recognised as the first microprogrammed computer to be designed, although it had not been implemented in hardware until 2002.^[9]
- The EMIDEC 1100^[10] reputedly uses a hard-wired control store consisting of wires threaded through ferrite cores, known as "the laces".
- Most models of the IBM System/360 series are microprogrammed:
 - The Model 25 is unique among System/360 models in using the top 16 K bytes of core storage to hold the control storage for the microprogram. The 2025 uses a 16-bit microarchitecture with seven control words (or microinstructions). At power up, or full system reset, the microcode is loaded from the card reader. The IBM 1410 emulation for this model is loaded this way.
 - The Model 30, the slowest model in the line, uses an 8-bit microarchitecture with only a few hardware registers; everything that the programmer saw is emulated by the microprogram. The microcode for this model is also held on special punched cards, which are stored inside the machine in a dedicated reader per card, called "CROS" units (Capacitor Read-Only Storage). A second CROS reader is installed for machines ordered with 1620 emulation.
 - The Model 40 uses 56-bit control words. The 2040 box implements both the System/360 main processor and the multiplex channel (the I/O processor). This model uses "TROS" dedicated readers similar to "CROS" units, but with an inductive pickup (Transformer Read-only Store).
 - The Model 50 has two internal datapaths which operated in parallel: a 32-bit datapath used for arithmetic operations, and an 8-bit data path used in some logical operations. The control store uses 90-bit microinstructions.
 - The Model 85 has separate instruction fetch (I-unit) and execution (E-unit) to provide high performance. The I-unit is hardware controlled. The E-unit is microprogrammed; the control words are 108 bits wide on a basic 360/85 and wider if an emulator feature is installed.
- The NCR 315 is microprogrammed with hand-wired ferrite cores (a RCM) pulsed by a sequencer with conditional execution. Wires routed through the cores are enabled for various data and logic elements in the processor.
- The Digital Equipment Corporation PDP-11 processors, with the exception of the PDP-11/20, are microprogrammed.^[11]
- Most Data General Eclipse minicomputers are microprogrammed. The task of writing microcode for the Eclipse MV/8000 is detailed in the Pulitzer Prize-winning book titled The Soul of a New Machine.
- Many systems from Burroughs are microprogrammed:
 - The B700 "microprocessor" execute application-level opcodes using sequences of 16-bit microinstructions stored in main memory; each of these is either a register-load operation or mapped to a single 56-bit "nanocode" instruction stored in read-only memory. This allows comparatively simple hardware to act either as a mainframe peripheral controller or to be packaged as a standalone computer.
 - The B1700 is implemented with radically different hardware including bit-addressable main memory but has a similar multi-layer organisation. The operating system preloads the interpreter for whatever language is required. These interpreters present different virtual machines for COBOL, Fortran, etc.
- Microdata produced computers in which the microcode is accessible to the user; this allows the creation of custom assembler level instructions. Microdata's Reality operating system design makes extensive use of this capability.
- The Xerox Alto workstation used a microcoded design but, unlike many computers, the microcode engine is not hidden from the programmer in a layered design. Applications take advantage of this to accelerate performance.
- The Nintendo 64's Reality Coprocessor (RCP), which serves as the console's graphics processing unit and audio processor, utilizes microcode; it is possible to implement new effects or tweak the processor to achieve the desired output. Some notable examples of custom RCP microcode include the high-resolution graphics, particle engines, and unlimited draw distances found in Factor 5's Indiana Jones and the Infernal Machine, Star Wars: Rogue Squadron, and Star Wars: Battle for Naboo;^{[12][13]} and the full motion video playback found in Angel Studios' Resident Evil 2.^[14]

- The VU0 and VU1 vector units in the Sony PlayStation 2 are microprogrammable; in fact, VU1 is only accessible via microcode for the first several generations of the SDK.
- The MicroCore Labs MCL86 (<http://www.microcorelabs.com/mcl86.html>), MCL51 (<http://www.microcorelabs.com/mcl51.html>) and MCL65 (<http://www.microcorelabs.com/mcl65.html>) are examples of highly encoded "vertical" microsequencer implementations of the Intel 8086/8088, 8051 and MOS 6502.

Implementation

Each microinstruction in a microprogram provides the bits that control the functional elements that internally compose a CPU. The advantage over a hard-wired CPU is that internal CPU control becomes a specialized form of a computer program. Microcode thus transforms a complex electronic design challenge (the control of a CPU) into a less complex programming challenge. To take advantage of this, a CPU is divided into several parts:

- A microsequencer picks the next word of the control store. A sequencer is mostly a counter, but usually also has some way to jump to a different part of the control store depending on some data, usually data from the instruction register and always some part of the control store. The simplest sequencer is just a register loaded from a few bits of the control store.
- A register set is a fast memory containing the data of the central processing unit. It may include the program counter, stack pointer, and other numbers that are not easily accessible to the application programmer. Often the register set is a triple-parted register file; that is, two registers can be read, and a third written at the same time.
- An arithmetic and logic unit performs calculations, usually addition, logical negation, a right shift, and logical AND. It often performs other functions, as well.

There may also be a memory address register and a memory data register, used to access the main computer storage. Together, these elements form an "execution unit". Most modern CPUs have several execution units. Even simple computers usually have one unit to read and write memory, and another to execute user code. These elements could often be brought together as a single chip. This chip comes in a fixed width that would form a "slice" through the execution unit. These are known as "bit slice" chips. The AMD Am2900 family is one of the best known examples of bit slice elements. The parts of the execution units and the execution units themselves are interconnected by a bundle of wires called a bus.

Programmers develop microprograms, using basic software tools. A microassembler allows a programmer to define the table of bits symbolically. Because of its close relationship to the underlying architecture, "microcode has several properties that make it difficult to generate using a compiler."⁽¹⁾ A simulator program is intended to execute the bits in the same way as the electronics, and allows much more freedom to debug the microprogram. After the microprogram is finalized, and extensively tested, it is sometimes used as the input to a computer program that constructs logic to produce the same data. This program is similar to those used to optimize a programmable logic array. Even without fully optimal logic, heuristically optimized logic can vastly reduce the number of transistors from the number required for a ROM control store. This reduces the cost of producing, and the electricity consumed by, a CPU.

Microcode can be characterized as *horizontal* or *vertical*, referring primarily to whether each microinstruction controls CPU elements with little or no decoding (horizontal microcode)⁽²⁾ or requires extensive decoding by combinatorial logic before doing so (vertical microcode). Consequently, each horizontal microinstruction is wider (contains more bits) and occupies more storage space than a vertical microinstruction.

Horizontal microcode

"Horizontal microcode has several discrete micro-operations that are combined in a single microinstruction for simultaneous operation."⁽³⁾ Horizontal microcode is typically contained in a fairly wide control store; it is not uncommon for each word to be 108 bits or more. On each tick of a sequencer clock a microcode word is read, decoded, and used to control the functional elements that make up the CPU.

In a typical implementation a horizontal microprogram word comprises fairly tightly defined groups of bits. For example, one simple arrangement might be:

Register source A	Register source B	Destination register	Arithmetic and logic unit operation	Type of jump	Jump address
-------------------	-------------------	----------------------	-------------------------------------	--------------	--------------

For this type of micromachine to implement a JUMP instruction with the address following the opcode, the microcode might require two clock ticks. The engineer designing it would write microassembler source code looking something like this:

```

! Any line starting with a dollar-sign is a comment
! This is just a label, the ordinary way assemblers symbolically represent a
! memory address.
InstructionJUMP:
! To prepare for the next instruction, the instruction-sensitive microcode has already
! moved the program counter to the memory address register. This instruction fetches
! the target address of the jump instruction from the memory word following the
! jump opcode, by copying from the memory data register to the memory address register.
! This gives the memory system two clock ticks to fetch the next
! instruction to the memory data register for use by the instruction decoder.
! The sequencer instruction "next" means just add 1 to the control word address.
NEXT, NONE, MAX, OVER, MAX, NONE
! This places the address of the next instruction into the PC.
! This gives the memory system a clock tick to finish the fetch started on the
! previous microinstruction.
! The sequencer instruction is to jump to the start of the instruction decode.
NAR, 1, PC, ADD, JMP, InstructionJUMP
! The instruction decode is not shown, because it is usually a mess, very particular
! to the exact processor being emulated. Even this example is simplified.

```



```

# Many CPUs have several ways to calculate the address, rather than just fetching
# it from the word following the opcode. Therefore, rather than just one
# jump instruction, these CPUs have a family of related jump instructions.

```

For each tick it is common to find that only some portions of the CPU are used, with the remaining groups of bits in the microinstruction being no-ops. With careful design of hardware and microcode, this property can be exploited to parallelise operations that use different areas of the CPU; for example, in the case above, the ALU is not required during the first tick, so it could potentially be used to complete an earlier arithmetic instruction.

Vertical microcode

In vertical microcode, each microinstruction is significantly encoded – that is, the bit fields generally pass through intermediate combinatory logic that, in turn, generates the actual control and sequencing signals for internal CPU elements (ALU, registers, etc.). This is in contrast with horizontal microcode, in which the bit fields themselves either directly produce the control and sequencing signals or are only minimally encoded. Consequently, vertical microcode requires smaller instruction lengths and less storage, but requires more time to decode, resulting in a slower CPU clock.^[a]

Some vertical microcode is just the assembly language of a simple conventional computer that is emulating a more complex computer. Some processors, such as DEC Alpha processors and the CMOS microprocessors on later IBM System/390 mainframes and z/Architecture mainframes, have PALcode (the term used on Alpha processors) or millicode (the term used on IBM mainframe microprocessors). This is a form of machine code, with access to special registers and other hardware resources not available to regular machine code, used to implement some instructions and other functions, such as page table walks on Alpha processors.^{[a][b]}

Another form of vertical microcode has two fields:

```

Field select | Field value

```

The field select selects which part of the CPU will be controlled by this word of the control store. The field value actually controls that part of the CPU. With this type of microcode, a designer explicitly chooses to make a slower CPU to save money by reducing the unused bits in the control store; however, the reduced complexity may increase the CPU's clock frequency, which lessens the effect of an increased number of cycles per instruction.

As transistors became cheaper, horizontal microcode came to dominate: the design of CPUs using microcode, with vertical microcode being used less often.

When both vertical and horizontal microcode are used, the horizontal microcode may be referred to as nanocode or picocode.^[a]

Writable control store

A few computers were built using "writable microcode". In this design, rather than storing the microcode in ROM or hard-wired logic, the microcode is stored in a RAM called a writable control store or WCS. Such a computer is sometimes called a writable instruction set computer or WISC.^[a]

Many experimental prototype computers use writable control stores; there are also commercial machines that use writable microcode, such as the Burroughs Small Systems, early Xerox workstations, the DEC VAX 8800 ("Nautilus") family, the Symbolics L- and G-machines, a number of IBM System/360 and System/370 implementations, some DEC PDP-10 machines,^[a] and the Data General Eclipse MV/8000.^[a]

Many more machines offer user-programmable writable control stores as an option, including the HP 2100, DEC PDP-11/60 and Varian Data Machines V-70 series minicomputers. The IBM System/370 includes a facility called Initial-Microprogram Load (IML or IMPL)^[a] that can be invoked from the console, as part of power-on reset (POR) or from another processor in a tightly coupled multiprocessor complex.

Some commercial machines, for example IBM 360/85,^{[a][b]} have both a read-only storage and a writable control store for microcode.

WCS offers several advantages including the ease of patching the microprogram and, for certain hardware generations, faster access than ROMs can provide. User-programmable WCS allows the user to optimize the machine for specific purposes.

Starting with the Pentium Pro in 1995, several x86 CPUs have writable Intel Microcode.^{[a][b]} This, for example, has allowed bugs in the Intel Core 2 and Intel Xeon microcodes to be fixed by patching their microprograms, rather than requiring the entire chips to be replaced. A second prominent example is the set of microcode patches that Intel offered for some of their processor architectures of up to 10 years in age, in a bid to counter the security vulnerabilities discovered in their designs – Spectre and Meltdown – which went public at the start of 2018.^{[a][b]} A microcode update can be installed by Linux,^[a] FreeBSD,^[a] Microsoft Windows,^[a] or the motherboard BIOS.^[a]

Comparison to VLIW and RISC

The design trend toward heavily microcoded processors with complex instructions began in the early 1960s and continued until roughly the mid-1980s. At that point the RISC design philosophy started becoming more prominent.

A CPU that uses microcode generally takes several clock cycles to execute a single instruction, one clock cycle for each step in the microprogram for that instruction. Some CISC processors include instructions that can take a very long time to execute. Such variations interfere with both interrupt latency and, what is far more important in modern systems, pipelining.

When designing a new processor, a hardwired control RISC has the following advantages over microcoded CISC:

- Programming has largely moved away from assembly level, so it's no longer worthwhile to provide complex instructions for productivity reasons.
- Simpler instruction sets allow direct execution by hardware, avoiding the performance penalty of microcoded execution.
- Analysis shows complex instructions are rarely used, hence the machine resources devoted to them are largely wasted.
- The machine resources devoted to rarely used complex instructions are better used for expediting performance of simpler, commonly used instructions.
- Complex microcoded instructions may require many clock cycles that vary, and are difficult to pipeline for increased performance.

There are counterpoints as well:

- The complex instructions in heavily microcoded implementations may not take much extra machine resources, except for microcode space. For instance, the same ALU is often used to calculate an effective address as well as computing the result from the actual operands (e.g., the original Z80, 8086, and others).
- The simpler non-RISC instructions (i.e., involving direct memory operands) are frequently used by modern compilers. Even immediate to stack (i.e., memory result) arithmetic operations are commonly employed. Although such memory operations, often with varying length encodings, are more difficult to pipeline, it is still fully feasible to do so - clearly exemplified by the i486, AMD K5, Cyrix 6x86, Motorola 68040, etc
- Non-RISC instructions inherently perform more work per instruction (on average), and are also normally highly encoded, so they enable smaller overall size of the same program, and thus better use of limited cache memories.

Many RISC and VLIW processors are designed to execute every instruction (as long as it is in the cache) in a single cycle. This is very similar to the way CPUs with microcode execute one microinstruction per cycle. VLIW processors have instructions that behave similarly to very wide horizontal microcode, although typically without such fine-grained control over the hardware as provided by microcode. RISC instructions are sometimes similar to the narrow vertical microcode.

Microcoding has been popular in application-specific processors such as network processors, microcontrollers, digital signal processors, channel controllers, disk controllers, network interface controllers, graphics processing units, and in other hardware.

Micro Ops

Modern CISC implementations, such as the x86 family, decode instructions into dynamically buffered micro-operations ("μops") with an instruction encoding similar to RISC or traditional microcode. A hardwired instruction decode unit directly emits μops for common x86 instructions, but falls back to a more traditional microcode ROM for more complex or rarely used instructions.^[a]

For example, an x86 might look up μops from microcode to handle complex multistep operations such as loop or string instructions, floating point unit transcendental functions or unusual values such as denormal numbers, and special purpose instructions such as CPUID.

See also

- [Address generation unit \(AGU\)](#)
- [CPU design](#)
- [Finite-state machine \(FSM\)](#)
- [Firmware](#)
- [Floating-point unit \(FPU\)](#)
- [Instruction pipeline](#)
- [MikroSim](#)
- [Microcode](#)
- [Superscalar](#)

Notes

a. IBM historically microcoded processors had multiple micro-orders and register select fields that required decoding.

References

1. Kent, Allen; Williams, James G. (April 5, 1993). *Encyclopedia of Computer Science and Technology, Volume 2B - Supplement 13* (<https://books.google.com/books?id=EJWV8J8COEYC>). New York: Marcel Dekker, Inc. ISBN 0-6247-2261-7. Retrieved Jan 17, 2016.
2. Fog, Agner (2017-05-02). "The microarchitecture of Intel, AMD and VIA CPUs" (<http://www.agner.org/optimize/microarchitecture.pdf>) (PDF). Technical University of Denmark.
3. "IBM pSeries Servers - Microcode Update for Ultrastar 73LZX (US73) 18/36 GB" (<http://download.boulder.ibm.com/ibmdl/pub/software/server/firmware/73lzx.html>). ibm.com. Retrieved January 22, 2016.
4. Manning, B.M.; Mitby, J.S.; Nicholson, J.O. (November 1979). "Microprogrammed Processor Having PLA Control Store" (<http://www.computerhistory.org/collections/accession/102960026>). IBM Technical Disclosure Bulletin. 22 (6).
5. Olan described a ROM/PLA control store in the context of usage in a CPL; "J-11: DEC's fourth and last PDP-11 microprocessor design ... features ... ROM/PLA control store" (<http://simh.trailing-edge.com/semi/j11.html>).

6. "6502 Images" (<http://www.visual6502.org/images/6502/>). Retrieved January 22, 2015.
7. *IBM System/360 Model 50 Functional Characteristics* (http://bitsavers.org/pdf/ibm/360/funcChar/A22-6898-1_360-50_funcChar_1967.pdf) (PDF). IBM, 1967. p. 7. Retrieved September 20, 2011.
8. Everett, R.R. & Swain, F.E. (1947). "Whirlwind I Computer Block Diagrams" (<https://web.archive.org/web/20120617112919/http://www.cryptosmith.com/wp-content/uploads/2009/05/whirlwindr-127.pdf>) (PDF). Report R-127. MIT Servomechanisms Laboratory. Archived from the original (<http://www.cryptosmith.com/wp-content/uploads/2009/05/whirlwindr-127.pdf>) (PDF) on June 17, 2012. Retrieved June 21, 2006.
9. "The Babbage Engine" (<http://www.computerhistory.org/babbage/>). Retrieved 11 December 2015.
10. "EMIDEC 1100 computer" (<http://www.emidec.org.uk/>). Emidec.org.uk. Retrieved April 26, 2010.
11. Edward A. Snow; Daniel P. Siewiorek (1982). "Implementation and Performance Evaluation of the PDP-11 Family" (http://gordonbell.azurewebsites.net/computer_structures_principles_and_examples/csp0667.htm) in Daniel P. Siewiorek; C. Gordon Bell; Allen Newell. *Computer Structures: Principles and Examples* (http://archive.computerhistory.org/resources/text/bell_gordon/bell.computer_structures_principles_and_examples.1982.102630397.pdf) (PDF). New York, NY: McGraw-Hill Book Company. p. 671. ISBN 0-07-057302-6.
12. "Interview: Battling the N64 (Naboo)" (<http://ign64.ign.com/articles/987/087646p1.html>). IGN64. November 10, 2000. Retrieved March 27, 2008.
13. "Indiana Jones and the Infernal Machine" (<http://www.ign.com/articles/2000/12/13/indiana-jones-and-the-infernal-machine-2>). IGN. December 12, 2000. Retrieved September 24, 2013.
14. Meynink, Todd (July 26, 2000). "Postmortem: Angel Studios' Resident Evil 2 (N64 Version)" (http://www.gamasutra.com/view/feature/3146/postmortem_angel_studios_php). *Gamasutra*. United Business Media LLC. Retrieved October 18, 2010.
15. Neat Harman; Andy Gimblett (2009-10-12). "CS-323: High Performance Microprocessors – Chapter 1. Microprogramming" (<http://euler.mat.uson.mx/~havillam/ca/CS323/0708.cs-323003.html>). *mat.uson.mx*. Retrieved 2015-08-08.
16. "PALcode for Alpha Microprocessors System Design Guide" (http://download.majix.org/dec/palcode_dsgn_gde.pdf) (PDF). Digital Equipment Corporation. May 1988. Retrieved November 7, 2013.
17. Robert Vaupel. *High Availability and Scalability of Mainframe Environments using System z and z/OS as example* (<http://diglib.unba.uni-kalsruhe.de/volltexte/documents/2591965>). ISBN 978-3-7315-0022-3.
18. Rogers, Bob (Sep–Oct 2012). "The What and Why of zEnterprise Millicode" (http://www.ibm-systemsmag.com/mainframe/administrator/performance/millicode_rogers/). *IBM Systems Magazine*.
19. Spruit, Wilhelm (Dec 2012). *The Design of a Microprocessor* (<https://books.google.com/books?id=0YmqCAAQBAJ>). Springer Science & Business Media. p. 31. ISBN 978-3-642-74916-2. Retrieved Jan 18, 2015.
20. "Writable instruction set, stack oriented computers: The WISC Concept" (http://www.ece.cmu.edu/~koopman/forth/rochester_87.pdf) article by Philip Koopman Jr. 1987
21. Eric Smith (3 September 2002). "Re: What was the size of Microcode in various machines" (<http://pdp10.nocrew.org/cpu/i10-ucode.txt>). Newsgroup: alt.folklore.computers (news.alt.folklore.computers); Usenet: qhn0qyveyu.fsf@ruckus.brouhaha.com (<news:qhn0qyveyu.fsf@ruckus.brouhaha.com>)
22. Mark Smotherman. "CP6C 330 / The Soul of a New Machine" (<http://www.cs.clemson.edu/~mark/330/eagle.html>). "4096 x 75-bit SRAM, writable control store; 74-bit microinstruction with 1 parity bit (16 fields)
23. *IBM System/370 Principles of Operation* (http://www.bitsavers.org/pdf/ibm/370/princOps/GA22-7000-4_370_Principles_of_Operation_Sep75.pdf) (PDF). Fourth Edition. IBM. September 1974. pp. 98, 245. GA22-7000-4.
24. *IBM System/360 Model 85 Functional Characteristics* (http://www.bitsavers.org/pdf/ibm/360/funcChar/A22-6916-1_360-85_funcChar_Jun68.pdf) (PDF). SECOND EDITION. IBM. June 1968. A22-6916-1.
25. *IBM System/360 Special Feature Description 709/7090/7094 Compatibility Feature for IBM System/360 Model 85*. First Edition. IBM. March 1968. SA27-2733-0.
26. Söller, Andreas; Paul, Matthias (1998-05-12). "Prozessorgeflüster" (<https://www.heise.de/ct/artikel/Prozessorgefluester-264546.html>). *ct – magazin für computertechnik*. Trends & News (in German). Heise Verlag. Archived (<https://web.archive.org/web/20170828172141/https://www.heise.de/ct/artikel/Prozessorgefluester-264546.html>) from the original on 2017-08-28. Retrieved 2017-06-28.
27. "Intel(R) 64 and IA-32 Architectures Software Developer's Manual", Volume 3A: System Programming Guide, Part 1 (<http://www.intel.com/Assets/PDF/manual/253668.pdf>), chapter 6.11: "Microcode update facilities", December 2008.
28. Intel Patches All Recent CPUs, Promises Hardware Fixes For Upcoming 8th Gen Chips (<http://www.tomshardware.com/news/intel-meltdown-spectre-patch-silicon-36672.html>) by Paul Alcorn on March 15, 2018
29. "Download Linux® Processor Microcode Data File" (<https://downloadcenter.intel.com/download/27591/Linux-Processor-Microcode-Data-File>)
30. "Intel Microcode Update Utility for Linux" (<https://web.archive.org/web/20120226174302/http://urbanmyth.org/microcode/>). Archived from the original (<http://urbanmyth.org/microcode/>) on 2012-02-26.
31. "ports/sysutils/devcpu" (<http://www.freebsd.org/cgi/ovsweb.cgi?ports/sysutils/devcpu>). FreeBSD.org. 2008-09-23. Retrieved 2010-04-26.
32. "A microcode reliability update is available that improves the reliability of systems that use Intel processors" (<http://support.microsoft.com/kb/936357>)
33. "Server Products - BIOS Update required when Missing Microcode message is seen during POST" (<https://web.archive.org/web/20140901063251/http://www.intel.com/support/motherboards/server/sb/cs-021619.htm>). Intel. January 24, 2013. Archived from the original (<http://www.intel.com/support/motherboards/server/sb/cs-021619.htm>) on September 1, 2014.

Further reading

- Smith, Richard E. (1988). "A Historical Overview of Computer Architecture" (<http://doi.ieeecomputersociety.org/10.1109/MAHC.1988.10039>). *Annals of the History of Computing*. **10** (4): 277–303. doi:10.1109/MAHC.1988.10039 (<https://doi.org/10.1109%2FMAHC.1988.10039>). Retrieved June 21, 2006.
- Smotherman, Mark (2005). "A Brief History of Microprogramming" (<http://www.cs.clemson.edu/~mark/uprog.html>). Retrieved July 30, 2006.
- Wilkes, M.V. (1986). "The Genesis of Microprogramming" (<http://doi.ieeecomputersociety.org/10.1109/MAHC.1986.10035>). *Annals of the History of Computing*. **8** (2): 116–126. doi:10.1109/MAHC.1986.10035 (<https://doi.org/10.1109%2FMAHC.1986.10035>). Retrieved August 7, 2006.
- Wilkes, M.V., and Stringer, J. B. (April 1953). "Microprogramming and the Design of the Control Circuits in an Electronic Digital Computer" (http://research.microsoft.com/~gbell/Computer_Structures_Principles_and_Examples/csp0174.htm). *Proceedings of the Cambridge Philosophical Society*. **49** (pt. 2): 230–238. doi:10.1017/S0305004100028322 (<https://doi.org/10.1017%2FS0305004100028322>). Retrieved August 23, 2006.
- Husson, S.S. (1970). *Microprogramming Principles and Practices*. Prentice-Hall. ISBN 0-13-581464-5.
- Tucker, S.G. (1967). "Microprogram control for SYSTEM/360" (<http://domino.research.ibm.com/tchjr/journalindex.nsf/a3807c5b4823c53f85256561008324bc/758c1e6a8a3e5d0265258bfa00665a2?7OpenDocument>). *IBM Systems Journal*. **6** (4): 222–241.

External links

- Writable Instruction Set Computer (<http://c2.com/cgi/wiki?WritableInstructionSetComputer>)
- Capacitor Read-only Store (<http://www.research.ibm.com/journal/rd/102/lbrmd1002F.pdf>)
- Transformer Read-only Store (http://www-03.ibm.com/ibm/history/exhibits/attic3/attic3_018.html)
- A Brief History of Microprogramming (<http://people.cs.clemson.edu/~mark/uprog.html>)
- Intel processor microcode security update (<https://lists.debian.org/debian-user/2013/09/msg00126.html>) (fixes the issues when running 32-bit virtual machines in PAE mode)
- Notes on Intel Microcode Updates (https://web.archive.org/web/20150907195925/http://inertiar.com/microcode/hawkes_intel_microcode.pdf), March 2013, by Ben Hawkes, archived from the original on September 7, 2015
- Hole seen in Intel's bug-busting feature (<https://web.archive.org/web/20030309102752/http://www.eetimes.com/news/97/963news/hole.html>), *EE Times*, 2002, by Alexander Wolfe, archived from the original on March 8, 2003
- Opteron Exposed: Reverse Engineering AMD K8 Microcode Updates (<http://www.securiteam.com/securityreviews/SFP0M1PDF0.html>), July 26, 2004

Retrieved from "https://en.wikipedia.org/w/index.php?title=Microcode&oldid=875350118"

This page was last edited on 28 December 2018, at 21:18 (UTC).

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#). Wikipedia® is a registered trademark of the [Wikimedia Foundation, Inc.](#), a non-profit organization.

Reexamination Control Number: 90014138
Confirmation No: 7638
RE: USP 9104842

37 CFR 1.131 Declaration of Scott Moskowitz, Attachment 16

I. Introduction

1. I am a named inventor of the following United States patents, as indicated by a search of the USPTO issued patents database using search query “(IN/scott AND IN/moskowitz).”

PAT. NO. Title

- 1 10,110,379 Full-Text System and methods for permitting open access to data objects and for securing data within the data objects
- 2 9,934,408 Full-Text Secure personal content server
- 3 9,893,888 Full-Text Utilizing data reduction in steganographic and cryptographic systems
- 4 9,843,445 Full-Text System and methods for permitting open access to data objects and for securing data within the data objects
- 5 9,830,600 Full-Text Systems, methods and devices for trusted transactions
- 6 9,710,669 Full-Text Secure personal content server
- 7 9,639,717 Full-Text Methods, systems and devices for packet watermarking and efficient provisioning of bandwidth
- 8 9,270,859 Full-Text Utilizing data reduction in steganographic and cryptographic systems
- 9 9,258,116 Full-Text System and methods for permitting open access to data objects and for securing data within the data objects
- 10 9,231,980 Full-Text Secure personal content server
- 11 9,191,206 Full-Text Multiple transform utilization and application for secure digital watermarking
- 12 9,191,205 Full-Text Multiple transform utilization and application for secure digital watermarking
- 13 9,171,136 Full-Text Data protection method and device
- 14 9,104,842 Full-Text Data protection method and device
- 15 9,070,151 Full-Text Systems, methods and devices for trusted transactions
- 16 9,021,602 Full-Text Data protection method and device
- 17 8,930,719 Full-Text Data protection method and device
- 18 8,798,268 Full-Text System and methods for permitting open access to data objects and for securing data within the data objects
- 19 8,789,201 Full-Text Secure personal content server
- 20 8,781,121 Full-Text Utilizing data reduction in steganographic and cryptographic systems
- 21 8,774,216 Full-Text Exchange mechanisms for digital information packages with bandwidth securitization, multichannel digital watermarks, and key management
- 22 8,767,962 Full-Text System and methods for permitting open access to data objects and for securing data within the data objects
- 23 8,739,295 Full-Text Secure personal content server
- 24 8,712,728 Full-Text Method and device for monitoring and analyzing signals

25 8,706,570 Full-Text Methods, systems and devices for packet watermarking and efficient provisioning of bandwidth
26 8,612,765 Full-Text Security based on subliminal and supraliminal channels for data objects
27 8,549,305 Full-Text Steganographic method and device
28 8,542,831 Full-Text Multiple transform utilization and application for secure digital watermarking
29 8,538,011 Full-Text Systems, methods and devices for trusted transactions
30 8,526,611 Full-Text Utilizing data reduction in steganographic and cryptographic systems
31 8,473,746 Full-Text Methods, systems and devices for packet watermarking and efficient provisioning of bandwidth
32 RE44,307 Full-Text Methods, systems and devices for packet watermarking and efficient provisioning of bandwidth
33 8,467,525 Full-Text Steganographic method and device
34 RE44,222 Full-Text Methods, systems and devices for packet watermarking and efficient provisioning of bandwidth
35 8,307,213 Full-Text Method and system for digital watermarking
36 8,281,140 Full-Text Optimization methods for the insertion, protection, and detection of digital watermarks in digital data
37 8,271,795 Full-Text Security based on subliminal and supraliminal channels for data objects
38 8,265,278 Full-Text System and methods for permitting open access to data objects and for securing data within the data objects
39 8,265,276 Full-Text Method for combining transfer functions and predetermined key creation
40 8,238,553 Full-Text Steganographic method and device
41 8,225,099 Full-Text Linear predictive coding implementation of digital watermarks
42 8,224,705 Full-Text Methods, systems and devices for packet watermarking and efficient provisioning of bandwidth
43 8,214,175 Full-Text Method and device for monitoring and analyzing signals
44 8,175,330 Full-Text Optimization methods for the insertion, protection, and detection of digital watermarks in digitized data
45 8,171,561 Full-Text Secure personal content server
46 8,161,286 Full-Text Method and system for digital watermarking
47 8,160,249 Full-Text Utilizing data reduction in steganographic and cryptographic system
48 8,121,343 Full-Text Optimization methods for the insertion, protection, and detection of digital watermarks in digitized data
49 8,104,079 Full-Text Methods, systems and devices for packet watermarking and efficient provisioning of bandwidth
50 8,046,841 Full-Text Steganographic method and device
51 7,991,188 Full-Text Optimization methods for the insertion, protection, and detection of digital watermarks in digital data
52 7,987,371 Full-Text Optimization methods for the insertion, protection, and detection of digital watermarks in digital data
53 7,953,981 Full-Text Optimization methods for the insertion, protection, and detection of digital watermarks in digital data
54 7,949,494 Full-Text Method and device for monitoring and analyzing signals

55 7,930,545 Full-Text Optimization methods for the insertion, protection, and detection of digital watermarks in digital data

56 7,913,087 Full-Text Optimization methods for the insertion, protection, and detection of digital watermarks in digital data

57 7,877,609 Full-Text Optimization methods for the insertion, protection, and detection of digital watermarks in digital data

58 7,870,393 Full-Text Steganographic method and device

59 7,844,074 Full-Text Optimization methods for the insertion, protection, and detection of digital watermarks in digitized data

60 7,830,915 Full-Text Methods and systems for managing and exchanging digital information packages with bandwidth securitization instruments

61 7,822,197 Full-Text Optimization methods for the insertion, protection, and detection of digital watermarks in digital data

62 7,813,506 Full-Text System and methods for permitting open access to data objects and for securing data within the data objects

63 7,779,261 Full-Text Method and system for digital watermarking

64 7,770,017 Full-Text Method and system for digital watermarking

65 7,761,712 Full-Text Steganographic method and device

66 7,738,659 Full-Text Multiple transform utilization and application for secure digital watermarking

67 7,730,317 Full-Text Linear predictive coding implementation of digital watermarks

68 7,664,958 Full-Text Optimization methods for the insertion, protection and detection of digital watermarks in digital data

69 7,664,264 Full-Text Utilizing data reduction in steganographic and cryptographic systems

70 7,664,263 Full-Text Method for combining transfer functions with predetermined key creation

71 7,660,700 Full-Text Method and device for monitoring and analyzing signals

72 7,647,503 Full-Text Optimization methods for the insertion, projection, and detection of digital watermarks in digital data

73 7,647,502 Full-Text Optimization methods for the insertion, protection, and detection of digital watermarks in digital data

74 7,568,100 Full-Text Steganographic method and device

75 7,532,725 Full-Text Systems and methods for permitting open access to data objects and for securing data within the data objects

76 7,530,102 Full-Text Methods, systems and devices for packet watermarking and efficient provisioning of bandwidth

77 (Not mine)

78 (Not mine)

79 7,475,246 Full-Text Secure personal content server

80 7,457,962 Full-Text Optimization methods for the insertion, protection, and detection of digital watermarks in digitized data

81 7,409,073 Full-Text Optimization methods for the insertion, protection, and detection of digital watermarks in digitized data

82 7,362,775 Full-Text Exchange mechanisms for digital information packages with bandwidth securitization, multichannel digital watermarks, and key management

83 7,346,472 Full-Text Method and device for monitoring and analyzing signals
84 7,343,492 Full-Text Method and system for digital watermarking
85 (Not mine)
86 7,287,275 Full-Text Methods, systems and devices for packet watermarking and efficient provisioning of bandwidth
87 7,177,429 Full-Text System and methods for permitting open access to data objects and for securing data within the data objects
88 7,159,116 Full-Text Systems, methods and devices for trusted transactions
89 7,152,162 Full-Text Z-transform implementation of digital watermarks
90 7,127,615 Full-Text Security based on subliminal and supraliminal channels for data objects
91 7,123,718 Full-Text Utilizing data reduction in steganographic and cryptographic systems
92 7,107,451 Full-Text Optimization methods for the insertion, protection, and detection of digital watermarks in digital data
93 7,095,874 Full-Text Optimization methods for the insertion, protection, and detection of digital watermarks in digitized data
94 7,035,409 Full-Text Multiple transform utilization and applications for secure digital watermarking
95 7,007,166 Full-Text Method and system for digital watermarking
96 6,870,477 Full-Text Method and apparatus for wireless mobile seating platform
97 6,853,726 Full-Text Z-transform implementation of digital watermarks
98 6,598,162 Full-Text Method for combining transfer functions with predetermined key creation 99 (NOT A SCOTT MOSKOWITZ PATENT)
100 6,522,767 Full-Text Optimization methods for the insertion, protection, and detection of digital watermarks in digitized data
101 (NOT A SCOTT MOSKOWITZ PATENT)
102 6,205,249 Full-Text Multiple transform utilization and applications for secure digital watermarking
103 6,078,664 Full-Text Z-transform implementation of digital watermarks
104 5,905,800 Full-Text Method and system for digital watermarking
105 5,889,868 Full-Text Optimization methods for the insertion, protection, and detection of digital watermarks in digitized data
106 5,822,432 Full-Text Method for human-assisted random key generation and application for digital watermark system
107 5,745,569 Full-Text Method for stega-cipher protection of computer code
108 5,687,236 Full-Text Steganographic method and device
109 5,613,004 Full-Text Steganographic method and device
110 5,539,735 Full-Text Digital information commodities exchange
111 5,428,606 Full-Text Digital information commodities exchange

2. Herein below I refer to attachments. I have reviewed each of the attachment I mention below, in preparation for signing this declaration.

II. Inventive Activity, Including Conception, up until the Time of Filing Application Number 08/587,943 1/17/1996

3. I began working on inventions in fields of digital technology in the early 1990s. In the early 1990s, I was looking for software programmers to code some of my inventions. I was living in Japan at this time.

4. One of my college buddies suggested I speak with Marc Cooperman because he believed that Marc had some coding ability. Marc was residing in the United States. So Marc and I had to generally correspond long distance instead of face to face.

5. I initially entered into a non-disclosure agreement with Marc, as shown in Attachment 23. This was in 1993. Later, I entered into a financial agreement with Marc involving co-ownership of the Dice Company for commercializing my inventions. Marc was involving in implementation in code and consequently was named as an inventor on some applications. However, Marc and I had insurmountable business differences. By the end of 1997, Marc and I were in dispute about finances and the direction and control of the Dice Company, as indicated by Attachment 24. Attachment 24 is the first page of a letter Marc sent me dated 11-18-1997. Eventually, we entered into lawsuits, a result of which generally speaking was dissolution of the Dice Company.

6. During 1995, I correspondence with Marc by email. On 11/11/1995, I sent an email to Marc Cooperman containing some of my inventive ideas. On 11/15/1995, I received an email from Marc restating some of my ideas, and containing an excerpt of my 11/11/1995 email to Marc. A copy of this email is Attachment 25. This email refers to my "note of 11-11-95" relating to "ascii/software steganographic protection." In this email, Marc refers to my ideas, stating:

Your idea seems to be

- 1) hide essential pieces of the app with an Argent-like scheme
- 2) make the "key/map" to access these resources randomized/individual on a per copy basis
- 3) maybe have the corrected key/map vary from run-to-run or iteration-to-iteration, as you seem to imply when talking about font metrics

7. On 12/22/1995, I completed a draft of a patent application, a copy of which is Attachment 26. See the bottom of page 3, stating "12-22-95 Scott Moskowitz". The last page of Attachment 26 refers to adding complexity to a hackers job by having the code reorganized between each "break." See the first two lines. The first full paragraph on the last page refers to the "special code resource" which knows where the "memory scheduler is in memory." That is special code resource calls the schedule and randomly moves the scheduler. The second full paragraph provides the alternative of the scheduler being capable of moving itself in memory (copy and then modify the program counter and stack frame). As stated in the last paragraph on the last page, these structures make it hard to analyze and capture memory containing application executable code. The first two pages are too blurry to read, but I believe they were disclosure of the concepts relating to encoding essential resources into data resources shown in my 1/3/1996 draft disclosure, which is next discussed.

8. On 1/3/1996, I completed a draft disclosure of the same patent application as on 12/22/1995. This 1/3/1996 application is Attachment 27. See "01-03-96 Scott Moskowitz" on the last page. This application notes the goal of providing security to executable code. See page 2. But not by stopping copying, but instead by ensuring the license information is preserved in copies of licensed software. And goes on to discuss what is stated in the 12/22/1995 draft, such as randomizing locations in memory of resources (page 2); hiding code resources in digitized sample resources (page 2); data resources (page 3); encoding essential resources into data resources (pages 4-5); and randomly reorganizing program memory structure during runtime (pages 5-6). The last full paragraph on page 2 reads:

An improvement over the art is disclosed in the present invention, in that software itself is a set of commands, compiled by the software engineer, which can be configured in such a manner as to tie underlying functionality to the license or authorization of the copy in the possession by the user. Without such verification, the functions sought by the user in the form of software cease to properly work. Attempts to tamper or "patch" substitute code resources can be made highly difficult by randomizing the location of said resources in memory on an intermittent basis to resist most attacks at disabling the system.

This goal of tying underlying functionality to the license or authorization of the copy in the possession by the user relates directly to claims 11-14 in USP 9104842. Page 4 of this draft discusses the method of compiling by encoding code resources in data resources using a key, including my discussion of code resources. This page refers to the licensed user and licensed copy for which a key is needed. This page states that the "key... corresponds, is equal to, or is a function of the license code," just like USP 9104842. Substantially the same disclosure appears in USP 9104842. The text spanning pages 4 and 5 contains the same "1)", "2)", and "3)" clauses describing operation of the installed software application that appear in USP 9104842. The description of uninhibited copying so long as the application contains the license code in page 5 is like the corresponding description in USP 9104842.

I see in claim 1 on page 6 that I claimed encoding a digital sample stream. I see in claim 2 that I claimed the key required to access the encoded information was a function of licensing information. I see in claim 3 that I claimed various transmission or memory media. I see, significantly, in claim 5, that I claimed decoding the digital watermark to extract, load, and execute the object code. I see in claim 6 that I claimed the decode resource contained in the code.

So this draft application substantially discloses claims 11-14 in USP 9104842.

And the date embedded on the last page of this draft is "01-03-96", followed by my name. This means to me that I wrote this draft and finished it on 1/3/1996.

III I Understood, by 1/03/1996, That the Claims Worked for Their Intended Purpose, and was fully disclosed in my Attachment 27 Draft

9. Not later than the 1/03/1996 date on my Attachment 27 draft, I recognized that the

subject matter defined by claims 11-14 of USP 9104842 worked for their intended purpose. These claims read:

11. A method for licensed software use, the method comprising: loading a software product on a computer, said computer comprising a processor, memory, an input, and an output, so that said computer is programmed to execute said software product;

said software product outputting a prompt for input of license information;

and

said software product using license information entered via said input in response to said prompt in a routine designed to decode a first license code encoded in said software product.

12. A method for encoding software code using a computer having a processor and memory, comprising: storing a software code in said memory; wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system; and encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key encoded software code; and wherein, when installed on a computer system, said first license key encoded software code will provide said specified underlying functionality only after receipt of said first license key.

13. A method for encoding software code using a computer having a processor and memory, comprising: storing a software code in said memory; wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system; and modifying, by said computer, using a first license key and an encoding algorithm, said software code, to form a modified software code; and wherein said modifying comprises encoding said first code resource to form an encoded first code resource; wherein said modified software code comprises said encoded first code resource, and a decode resource for decoding said encoded first code resource; wherein said decode resource is configured to decode said encoded first code resource upon receipt of said first license key.

14. A method for encoding software code using a computer having a processor and memory, comprising: storing a software code in said memory; wherein said software code defines software code interrelationships between code resources that result in a specified underlying functionality when installed on a computer system; and encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key encoded software code in which at least one of said software code interrelationships are encoded.

10. For example, USP 9104842 claim 11 requires “said software product using license information entered via said input in response to said prompt in a routine designed to decode a first license code encoded in said software product. Encode and decode routines existed, as indicated for example by my reference on page 4 to my prior Stegacipher patent. My

attachment 27 draft page 4 discusses using a key based decoder to decode the encoded code resource. So there was no uncertainty that decoding encoded software worked. For example, USP 9104842 claim 12 requires also that “first license key encoded software code will provide said specified underlying functionality only after receipt of said first license key.” For the same reason states for claim 11, there was no doubt that claim 12 worked. Claim 14 differs by requiring decoding “upon receipt” of the key. But that merely requires conventional coding. And claim 14 requires “software code interrelationships.” But software code interrelationships are a necessary consequence of encoding software having relationships. My Attachment 27, on page 3, first paragraph, describes these interrelations as the location in memory where sub objects defining functions, can be found.

11. I am told that Attachment 1 is a copy of the official file history of the USPTO, for 08/587,943, as filed. (I vaguely recall some third party providing this in one of the IPR’s filed against one of my patents.) On my brief comparison, I note that disclosure in Attachment 1, other than the claims and abstract, are substantially identical to the disclosure of my 1/3/1996 draft disclosure, Attachment 27.

IV. My 08/587,943 application issued into USP 5745469

12. On 1/17/1996, an application containing this disclosure was filed in the USPTO. The USPTO assigned application number 08/587,943 to this disclosure. Application number 08/587,943 issued as USP 5745569, on 4/28/1998, as shown by Attachment 2 (which is a copy of USP 5745569).

13. Moreover, the disclosures appear in my 08/587,943 application filed a couple weeks later, on 1/17/1996.

My 08/587,943 application issued into USP 5745469. Claims 1-3 in this patent read:

1. A method for copy protection of computer software, the computer software including executable code and a non-executable digital sample, said method comprising the steps of:
 - identifying a portion of the executable code to be encoded;
 - generating an encoded code resource from the identified portion of the executable code; and
 - embedding the encoded code resource in the non-executable digital sample.
2. The method of claim 1, further comprising the step of requiring a predetermined key to decode the encoded code resource prior to execution of the executable code.
3. The method of claim 2, further comprising deriving the predetermined key from licensing information associated with the computer software.

14. These claims of my USP 5745469 show that I pursued to issuance, claims to “embedding” a “code resource” in a “non-executable” digital sample, and requiring a key necessary to access those code resources, and in which the key had to be derived from licensing information. Those claim elements are very similar to the elements of the rejected claims of USP

9104842.

III.C My Application Number 08/587,943 1/17/1996 My 08/587,943 Discloses Claims 11-14 of USP 9104842

15. My 08/587,943 application discloses the subject matter of claims 11-14 of USP 9104842. I show this in the following claim chart. This chart recites claims 11-14 of USP 9104842 in the left column. This chart's right hand column recites the corresponding original text of my 08/587,943 application and cites to its location in Attachment 1.

Claims 11-14 of USP 9104842.	Attachment 1, the 08/587,943 application. Page citation are to the numbers appearing at the bottom of the pages in Attachment 1, and not to the page of the pdf document.
CLAIM 11	
11. A method for licensed software use, the method comprising:	Page 1:8 refers to the field of the invention as including "computer software." Page 1:25 refers to "Other methods for protection of computer software." Page 3:29-30 refers to a "key is linked to a license code by means of a mathematical function" Page 5:25-26 refers to the "goal of the present invention, to provide a level of security for executable code." Page 11:24:33 discloses operation of "the application." Thus, 08/587,943 discloses a method for licensed software use.
loading a software product on a computer,	Page 11:24-25 states "The application can then operate as follows: 1) when it is run for the first time...." Running an application requires the application be loaded. Hence, this passage discloses loading the software application.

<p>said computer comprising a processor, memory, an input, and an output, so that said computer is programmed to execute said software product;</p>	<p>Computers necessarily include a processor, memory, an input, and an output. Accordingly, the many references to computers and software and running and loading software in the application respond to this limitation. See for example, “memory” at page 3:34; 4:1; 6:9; 7:5. See for example, for an input, page 11:25-26 (programmed computer “asks the user for personalization information”). See for example for processor, input, and output, page 12:36-37 (“embedded system”); and 13:15 (“set-top box”).</p>
<p>said software product outputting a prompt for input of license information;</p>	<p>Page 11:24-26 states “The application can then operate as follows: 1) when it is run for the first time, after installation, it asks the user for personalization information...” Asking the user is a prompt.</p>
<p>and said software product using license information entered via said input in response to said prompt in a routine designed to decode a first license code encoded in said software product.</p>	<p>Page 11:31-33 states “3) Once it has -the license code, it can then generate the proper decoding key to access the essential code resources.” This indicates the software program uses the license code input in response to the prompt to generate the decoding key. And this passage also states the software program uses the decoding key to actually decode the code resources of the software program. Page 11:37 to page 12:2 states “This method, then, is to choose the key so that it ... is equal to ... a license code...” And page 11:11-13 “Alternatively; the key ... can be stored as a data resource and encrypted” This discloses encoding the license code in the software product.</p>
<p>Claim 12</p>	<p>Attachment 1, the 08/587,943 application</p>

<p>12. A method for encoding software code using a computer having a processor and memory, comprising:</p>	<p>Page 1:8 refers to the field of the invention as including “computer software.” Page 10:14-16 states “The utility will choose one or several essential coder resources, and encode them into one or several data resources....” This discloses encoding software code using a computer. The same paragraph explains at page 10:8-9 that “When code and data resources are compiled and assembled...” This passage discloses the utility is being used on a processor having memory.</p>
<p>storing a software code in said memory;</p>	<p>Page 7:16 refers to the “arrangement .in memory” as not important, indicating the software is stored in memory. Page 7:19 refers to “The memory address. of the first instruction” which indicates that instructions, which are the software code, are stored in memory.</p>
<p>wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system; and</p>	<p>Page 7:26-28 states that ‘These sub-objects can be packaged into what are referred to in certain systems as ‘code resources,’” which discloses that the software code comprises code resources. Page 6:19-20 equate “indivisible portions of object code” with “with the programmers function or procedure implementations in higher level languages” and that when optimized and compiled form the page 7:28 “code resources.”</p>

<p>encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key encoded software code;</p>	<p>Page 10: 13-15 states “The utility will choose one or several essential code resources, and encode them into one or several data resources.” which discloses encoding of the software code. Page 10:28-29 states ‘For the encoding of the essential code resources, a ‘key’ is needed”, which discloses the encoding algorithm using the key. Page 10:29-30 states that “Such a key is similar to those described in [U.S. Patent 5,613,003] the "Steganographic Method Device” patent.” This patent discloses an encoding algorithm. Page 4:10-15 incorporate by reference U.S. Patent 5,613,003.</p>
<p>and wherein, when installed on a computer system, said first license key encoded software code will provide said specified underlying functionality only after receipt of said first license key.</p>	<p>Page 11:6-8 states that “The key is necessary to access the underlying code, i.e., what the user understands to be the application program.” This discloses the application only runs after it receives the license key.</p>
<p>CLAIM 13</p>	<p>Attachment 1, the 08/587,943 application</p>
<p>13. A method for encoding software code using a computer having a processor and memory, comprising:</p>	<p>Claim 12 has the same recitation. See claim 12 for disclosure in Attachment 1, the 08/587,943 application.</p>
<p>storing a software code in said memory;</p>	<p>Claim 12 has the same recitation. See claim 12 for disclosure in Attachment 1, the 08/587,943 application.</p>
<p>wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system; and</p>	<p>Claim 12 has the same recitation. See claim 12 for disclosure in Attachment 1, the 08/587,943 application.</p>
<p>modifying, by said computer, using a first license key and an encoding algorithm, said software code, to form a modified software code; and</p>	<p>Encoding is modifying. Claim 12 recites encoding. See claim 12's encoding recitation for disclosure in Attachment 1, the 08/587,943 application, of this modifying recitation.</p>

wherein said modifying comprises encoding said first code resource to form an encoded first code resource;	Claim 12 recites “to form a first license key encoded software code” This is the same thing as claim 13’s “ encoded first code resource.” See claim 12’s modifying to form the first license key encoded software code for disclosure in Attachment 1, the 08/587,943 application, of this “form an encoded first code resource;” of claim 13.
wherein said modified software code comprises said encoded first code resource, and a decode resource for decoding said encoded first code resource;	Page 11:17-19 states “Note further that the application contains a code resource which performs the function of decoding an encoded code resource from a data resource.” This discloses the claimed decode resource.
wherein said decode resource is configured to decode said encoded first code resource upon receipt of said first license key.	Page 11:17-19 states in part that the decode resource “performs the function of decoding an encoded code resource.” Page 11:24-33 states in relevant part that the application the license code after which “it can then generate the proper decoding key to access the essential code resources.” This discloses the decode key decoding upon receipt of the first license key.
CLAIM 14	Attachment 1, the 08/587,943 application
14. A method for encoding software code using a computer having a processor and memory, comprising:	Claim 12 has the same recitation. See claim 12 for disclosure in Attachment 1, the 08/587,943 application.
storing a software code in said memory;	Claim 12 has the same recitation. See claim 12 for disclosure in Attachment 1, the 08/587,943 application.

<p>wherein said software code defines software code interrelationships between code resources that result in a specified underlying functionality when installed on a computer system; and</p>	<p>Claim 12 has the same recitation, except for the “code interrelationships.” See claim 12 for disclosure in Attachment 1, the 08/587,943 application. Regarding “code interrelationships,” page 7:26-28 states that “These sub-objects can be packaged into what are referred to in certain systems as ‘code resources,’” which discloses that the software code comprises code resources. Page 6:19-20 equate “indivisible portions of object code” with “with the programmers function or procedure implementations in higher level languages” and that when optimized and compiled form the page 7:28 “code resources.” Page 7:19-21 explains that “The memory address of the first instruction in one of these objects is called the “entry point” of the function or procedure.” And page 7:17-18 explains that “exact order or arrangement .in memory is not important” of the sub-object is not important, “so long as any sub-object which uses another sub-object knows where in memory it can be found.” These sub-objects are code resources and this disclosure of locating the entry points of sub-objects in memory is disclosure of code interrelationships between code resources.</p>
<p>encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key encoded software code</p>	<p>Claim 12 has the same recitation. See claim 12 for disclosure in Attachment 1, the 08/587,943 application.</p>
<p>in which at least one of said software code interrelationships are encoded.</p>	<p>The recitation in this claim “wherein said software code defines software code interrelationships between code resources that result in a specified underlying functionality when installed on a computer system;” identifies and explains the code resource interrelationships. Page 10:13-16 discloses encoding “several essential code resources” to form the executable application. That discloses encoding the code interrelationships.</p>

16. Thus, my application 08/587,943 discloses all of the limitations Claims 11-14 of USP 9104842.

IV. Jurat

17. I have been warned that willful false statements and the like are punishable by fine or imprisonment, or both (18 U.S.C. 1001) and may jeopardize the validity of the application or any patent issuing thereon. All statements I make in the declaration I either know to be true or on information and belief I believe them to be true.

Signed:


SCOTT MOSKOWITZ

Y:\Clients\SCOT Scott A Moskowitz and Wistaria Trading, Inc\90014138, USP9104842, SCOT0014-4\Drafts\Attachment16_131MoskowitzDeclaration.wpd

Reexamination Control Number: 90014138
Confirmation No: 7638
RE: USP 9104842

37 CFR 1.132 Declaration of Scott Moskowitz, Attachment 17

I. Introduction

1. I am a named inventor of the following United States patents, as indicated by a search of the USPTO issued patents database using search query “(IN/scott AND IN/moskowitz).”

<u>PAT. NO.</u>	<u>Title</u>
1 10,110,379	Full-Text System and methods for permitting open access to data objects and for securing data within the data objects
2 9,934,408	Full-Text Secure personal content server
3 9,893,888	Full-Text Utilizing data reduction in steganographic and cryptographic systems
4 9,843,445	Full-Text System and methods for permitting open access to data objects and for securing data within the data objects
5 9,830,600	Full-Text Systems, methods and devices for trusted transactions
6 9,710,669	Full-Text Secure personal content server
7 9,639,717	Full-Text Methods, systems and devices for packet watermarking and efficient provisioning of bandwidth
8 9,270,859	Full-Text Utilizing data reduction in steganographic and cryptographic systems
9 9,258,116	Full-Text System and methods for permitting open access to data objects and for securing data within the data objects
10 9,231,980	Full-Text Secure personal content server
11 9,191,206	Full-Text Multiple transform utilization and application for secure digital watermarking
12 9,191,205	Full-Text Multiple transform utilization and application for secure digital watermarking
13 9,171,136	Full-Text Data protection method and device
14 9,104,842	Full-Text Data protection method and device
15 9,070,151	Full-Text Systems, methods and devices for trusted transactions
16 9,021,602	Full-Text Data protection method and device
17 8,930,719	Full-Text Data protection method and device
18 8,798,268	Full-Text System and methods for permitting open access to data objects and for securing data within the data objects
19 8,789,201	Full-Text Secure personal content server
20 8,781,121	Full-Text Utilizing data reduction in steganographic and cryptographic systems
21 8,774,216	Full-Text Exchange mechanisms for digital information packages with bandwidth securitization, multichannel digital watermarks, and key management
22 8,767,962	Full-Text System and methods for permitting open access to data objects and for securing data within the data objects
23 8,739,295	Full-Text Secure personal content server

24 8,712,728 Full-Text Method and device for monitoring and analyzing signals
25 8,706,570 Full-Text Methods, systems and devices for packet watermarking and efficient provisioning of bandwidth
26 8,612,765 Full-Text Security based on subliminal and supraliminal channels for data objects
27 8,549,305 Full-Text Steganographic method and device
28 8,542,831 Full-Text Multiple transform utilization and application for secure digital watermarking
29 8,538,011 Full-Text Systems, methods and devices for trusted transactions
30 8,526,611 Full-Text Utilizing data reduction in steganographic and cryptographic systems
31 8,473,746 Full-Text Methods, systems and devices for packet watermarking and efficient provisioning of bandwidth
32 RE44,307 Full-Text Methods, systems and devices for packet watermarking and efficient provisioning of bandwidth
33 8,467,525 Full-Text Steganographic method and device
34 RE44,222 Full-Text Methods, systems and devices for packet watermarking and efficient provisioning of bandwidth
35 8,307,213 Full-Text Method and system for digital watermarking
36 8,281,140 Full-Text Optimization methods for the insertion, protection, and detection of digital watermarks in digital data
37 8,271,795 Full-Text Security based on subliminal and supraliminal channels for data objects
38 8,265,278 Full-Text System and methods for permitting open access to data objects and for securing data within the data objects
39 8,265,276 Full-Text Method for combining transfer functions and predetermined key creation
40 8,238,553 Full-Text Steganographic method and device
41 8,225,099 Full-Text Linear predictive coding implementation of digital watermarks
42 8,224,705 Full-Text Methods, systems and devices for packet watermarking and efficient provisioning of bandwidth
43 8,214,175 Full-Text Method and device for monitoring and analyzing signals
44 8,175,330 Full-Text Optimization methods for the insertion, protection, and detection of digital watermarks in digitized data
45 8,171,561 Full-Text Secure personal content server
46 8,161,286 Full-Text Method and system for digital watermarking
47 8,160,249 Full-Text Utilizing data reduction in steganographic and cryptographic system
48 8,121,343 Full-Text Optimization methods for the insertion, protection, and detection of digital watermarks in digitized data
49 8,104,079 Full-Text Methods, systems and devices for packet watermarking and efficient provisioning of bandwidth
50 8,046,841 Full-Text Steganographic method and device
51 7,991,188 Full-Text Optimization methods for the insertion, protection, and detection of digital watermarks in digital data
52 7,987,371 Full-Text Optimization methods for the insertion, protection, and detection of digital watermarks in digital data
53 7,953,981 Full-Text Optimization methods for the insertion, protection, and detection of digital watermarks in digital data

54 7,949,494 Full-Text Method and device for monitoring and analyzing signals
55 7,930,545 Full-Text Optimization methods for the insertion, protection, and detection of digital watermarks in digital data
56 7,913,087 Full-Text Optimization methods for the insertion, protection, and detection of digital watermarks in digital data
57 7,877,609 Full-Text Optimization methods for the insertion, protection, and detection of digital watermarks in digital data
58 7,870,393 Full-Text Steganographic method and device
59 7,844,074 Full-Text Optimization methods for the insertion, protection, and detection of digital watermarks in digitized data
60 7,830,915 Full-Text Methods and systems for managing and exchanging digital information packages with bandwidth securitization instruments
61 7,822,197 Full-Text Optimization methods for the insertion, protection, and detection of digital watermarks in digital data
62 7,813,506 Full-Text System and methods for permitting open access to data objects and for securing data within the data objects
63 7,779,261 Full-Text Method and system for digital watermarking
64 7,770,017 Full-Text Method and system for digital watermarking
65 7,761,712 Full-Text Steganographic method and device
66 7,738,659 Full-Text Multiple transform utilization and application for secure digital watermarking
67 7,730,317 Full-Text Linear predictive coding implementation of digital watermarks
68 7,664,958 Full-Text Optimization methods for the insertion, protection and detection of digital watermarks in digital data
69 7,664,264 Full-Text Utilizing data reduction in steganographic and cryptographic systems
70 7,664,263 Full-Text Method for combining transfer functions with predetermined key creation
71 7,660,700 Full-Text Method and device for monitoring and analyzing signals
72 7,647,503 Full-Text Optimization methods for the insertion, projection, and detection of digital watermarks in digital data
73 7,647,502 Full-Text Optimization methods for the insertion, protection, and detection of digital watermarks in digital data
74 7,568,100 Full-Text Steganographic method and device
75 7,532,725 Full-Text Systems and methods for permitting open access to data objects and for securing data within the data objects
76 7,530,102 Full-Text Methods, systems and devices for packet watermarking and efficient provisioning of bandwidth
77 (Not mine)
78 (Not mine)
79 7,475,246 Full-Text Secure personal content server
80 7,457,962 Full-Text Optimization methods for the insertion, protection, and detection of digital watermarks in digitized data
81 7,409,073 Full-Text Optimization methods for the insertion, protection, and detection of digital watermarks in digitized data
82 7,362,775 Full-Text Exchange mechanisms for digital information packages with bandwidth

securitization, multichannel digital watermarks, and key management
83 7,346,472 Full-Text Method and device for monitoring and analyzing signals
84 7,343,492 Full-Text Method and system for digital watermarking
85 (Not mine)
86 7,287,275 Full-Text Methods, systems and devices for packet watermarking and efficient provisioning of bandwidth
87 7,177,429 Full-Text System and methods for permitting open access to data objects and for securing data within the data objects
88 7,159,116 Full-Text Systems, methods and devices for trusted transactions
89 7,152,162 Full-Text Z-transform implementation of digital watermarks
90 7,127,615 Full-Text Security based on subliminal and supraliminal channels for data objects
91 7,123,718 Full-Text Utilizing data reduction in steganographic and cryptographic systems
92 7,107,451 Full-Text Optimization methods for the insertion, protection, and detection of digital watermarks in digital data
93 7,095,874 Full-Text Optimization methods for the insertion, protection, and detection of digital watermarks in digitized data
94 7,035,409 Full-Text Multiple transform utilization and applications for secure digital watermarking
95 7,007,166 Full-Text Method and system for digital watermarking
96 (Not mine)
97 6,853,726 Full-Text Z-transform implementation of digital watermarks
98 6,598,162 Full-Text Method for combining transfer functions with predetermined key creation 99 (NOT A SCOTT MOSKOWITZ PATENT)
100 6,522,767 Full-Text Optimization methods for the insertion, protection, and detection of digital watermarks in digitized data
101 (NOT A SCOTT MOSKOWITZ PATENT)
102 6,205,249 Full-Text Multiple transform utilization and applications for secure digital watermarking
103 6,078,664 Full-Text Z-transform implementation of digital watermarks
104 5,905,800 Full-Text Method and system for digital watermarking
105 5,889,868 Full-Text Optimization methods for the insertion, protection, and detection of digital watermarks in digitized data
106 5,822,432 Full-Text Method for human-assisted random key generation and application for digital watermark system
107 5,745,569 Full-Text Method for stega-cipher protection of computer code
108 5,687,236 Full-Text Steganographic method and device
109 5,613,004 Full-Text Steganographic method and device
110 5,539,735 Full-Text Digital information commodities exchange
111 5,428,606 Full-Text Digital information commodities exchange

2. Herein below I refer to attachments and exhibits and the Non Final Office Action (NFOA) issued in this reexamination. I have reviewed each of the attachment, exhibits, and the NFOA, I mention below, in preparation for signing this declaration. I have also reviewed and approved of a draft response to the NFOA. I understand that my declaration

supports the factual statements and technical conclusions I make below regarding Beetcher.

3. The rest of my declaration tracks the draft response to the NFOA

VI. The Beetcher Rejections Should be Withdrawn Because They are Improperly Vague

4. The NFOA's rejections based upon Beetcher do not explain the rejection. All they do is state "here is a bunch of stuff in Beetcher" in subsections titled with quotes from the rejected claims. This fails to identify a correspondence of disclosure in Beetcher to claim limitations.

5. The NFOA in this regard parallels the biased litigation work product in the request for reexamination. There are in fact block quotes restated in the NFOA that are taken, without attribution, from the request for reexamination, and at least one such block quote is an inaccurate rendition of text in the request for reexamination.

6. I have had to speculate what might be the basis for rejection. If the examiner maintains that the claims are anticipated by Beetcher, then the examiner should impose a new **non-final** rejection that explains the basis for the rejections with clarity, expressly identifying what correspondence exists between claim limitations, and disclosure in Beetcher.

VII.1 The Beetcher Rejections, Whatever Their Basis, are Wrong, and Should be Withdrawn

7. Its not clear from the NFOA why the claims are rejected based upon Beetcher. However, no correspondence of claimed limitations to Beetcher's disclosures results in anticipation of claim 11. Claim 11 is shown below, followed by description of Beetcher's unencrypted entitlement key 200, encrypted entitlement key 111, and entitlement verification instruction 301, and Beetcher's disclosure how they are used.

VII.2 Rejected Claim 11

8. Claim 11 reads:

11. A method for licensed software use, the method comprising:

loading a software product **on a computer**, said computer comprising a processor, memory, an input, and an output, so that said computer is programmed to execute said software product;

said software product outputting a prompt for input of license information;

and

said software product using license information **entered via said input in response to** said prompt ***in a routine designed to decode a first license code encoded in said software product.***

VII.2.1 Description of Beetcher's Unencrypted entitlement key 200

9. What does Beetcher Fig. 2 show? Beetcher col. 4:52-53 states that "FIG. 2 shows the unencrypted contents of an entitlement key according to the preferred embodiment of this invention." Beetcher's Fig. 2 shows entitlement key 200 including 128 bits in five enumerated

fields, 201 to 205. Fields 201 to 205 are described in the paragraph at Beetcher col. 6:20-40, which state that there are: 4 bits defining a charge group field 201; 8 bits defining software version field 202; 8 bits defining key type field 203; 28 bits defining machine serial number field 204; and 80 bits defining product entitlement flags 205.

10. Beetcher col. 6:36-40 specifies that entitlement flags 205 each correspond to a product number, and each of those flags is set to "1" if the product number is entitled to run (aka licensed for use on that computer) and otherwise set to "0":

...Product entitlement flag 205 is an 80-bit field containing 80 separate product flags, each corresponding to a product number. The bit is set to '1' if the corresponding product number is entitled; otherwise it is set to "0".

11. Beetcher also discloses that the marketing system 124 generates the entitlement key 200 specific to a customer's order, including the serial number 105, 410 of the customer's computer, and generates encrypted entitlement key 111 from entitlement key 200, such that the serial number 105 is required to decrypt encrypted entitlement key 200. Beetcher's paragraph at col. 9:21-47 describes how Beetcher's entitlement key is generated and then encrypted. Beetcher discloses that the marketing system 124, Fig. 1, receives a customer order, and a key generator/encryptor 122 on marketing system 124 retrieves information about the customer from its database 123, and generates the unencrypted entitlement key 200. Beetcher col.8:22-36.

12. Beetcher discloses unencrypted entitlement key 200 includes the serial number of the customer computer 101 stored in the customer computer. Abstract ("The key includes the serial number of the computer for which the software is licensed"); col. 4:6-7 ("The key includes the serial number of the machine for which the software is licensed"); which customer computer 101 uses to decrypt encrypted entitlement key 111, see col. 4:9-13 ("The decryption mechanism fetches the machine serial number and uses it as a key to decrypt the entitlement key.")

VII.2.2 Description of How Beetcher's Marketing System 124 and Customer Computer 101 Use Beetcher's UnEncrypted Entitlement Key 200 and Encrypted Entitlement Key 111

13. Beetcher col. 8:39-40 then explains that "Key generator/encryptor 122 then encrypts the key," forming encrypted entitlement key 111. Beetcher col. 8:42-43 then explains that "The resultant encrypted entitlement key 111 is then transmitted to the customer at step 805."

14. Beetcher's paragraph at col. 9: 49 to 10:19 explains that Beetcher's customer/user's computer (Fig. 1, element 101) decrypts the entered and encrypted entitlement key 111, and stores entitlement data in memory. Beetcher col. 9:51-52 discloses that the customer enters encrypted entitlement key 111 into computer system 101. Beetcher col. 9:55-60 discloses that unlock routine 430 uses the serial number of computer 101 to "decode" [sic; decrypt] encrypted entitlement key 111. Beetcher col. 9:56 to 10:39 then describes the process of storing new unencrypted entitlement key data in product key table 450. Beetcher col. 9:67 to 10:2 discloses this process includes determining if the decoded [sic; decrypted] entitlement flag for a product is "1". If so, Beetcher col. 10:3-8 discloses that unlock routine 430 updates entry 501 of product key table 450 with the "version number, and charge group value" contained in the entitlement key; and sets entitlement bit field 506 of entry 501 of product key table 450 (Fig. 5) to "1."

15. Beetcher col. 10: then discloses that, if the entitlement flag for a product is "0", unlock

routing 430 also sets the version number in product lock table 460 to "0" (indicating no entitlement).

16. Finally, Beetcher col. 10:18-19 disclose that, once table 450 is rebuilt, its contents are "saved in storage at step 909." Presumably, this reference to storage refers to the "plurality of storage devices 106, 107, 108." attached to the customer computer system 101 shown in Fig. 1 and identified at col. 5:21-22. Presumably, this statement refers to the transmission of table 450 from RAM 101 of the customer computer 101 to one of these storage devices, so that table 450 will survive reinitialization of the customer's computer. This presumption is in view of Beetcher's discussion at col. 8:23-27 that product key table 450 is contained in RAM 104, but duplicated in a non-volatile storage device so it can be recovered in case the computer system is powered down. It is well known in the art that data in RAM is lost when the RAM is powered down, and that RAM is powered down upon re-initialization of a computer system.

17. In summary, Beetcher discloses that Beetcher's customer computer system 101 receives encrypted entitlement key 111, decrypts the key using the customer computer 101's hard coded machine identification, and saves entitlement data in memory in records in tables 450 and 460.

VII.2.3 Beetcher's Entitlement Key is Not Part of Beetcher's Software Module

18. Beetcher contains no disclosure indicating that Beetcher's entitlement key is part of Beetcher's installed software module. Beetcher notes that the encrypted key is sent separately from the software modules, from the marketing system 124 to the customer computer 101. Abstract and col. 4:4-5 ("separately distributed encrypted entitlement key"). Beetcher Fig. 3 and col. 6:41-43 discloses that the software modules are "compiled object code". Beetcher's entitlement key 111, 200 is not object code.

VII.2.4 Beetcher's Entitlement Key is Not Used to Decode Any Instruction In Beetcher's Software Module

19. Beetcher contains no disclosure indicating that Beetcher's entitlement key, or any data stored in customer computer 101 derived from Beetcher's entitlement key, is used to decode an instruction in Beetcher's object code. Beetcher does not disclose that its instruction 301 is encrypted or encoded using Beetcher's entitlement key. Therefore, there would be no basis to decrypt or decode instruction 301 using Beetcher's entitlement key.

VII.2.5 Description of Beetcher's entitlement verification instruction 301

20. Beetcher col. 6:47-48 notes that all the instructions 301 in a software module are identical. Beetcher discloses that instruction 301 is a 48 bit instruction consisting of four fields: Operation; Unused; Version; and Product Number. Beetcher discloses that Fig. 3 shows contents of a typical executable software module (Beetcher col. 4:54-55, Brief description of Fig. 3). Beetcher Fig. 3 and col. 6:41-46 disclose that Fig. 3 shows software module 300 "comprises a plurality of object code instructions capable of executing on computer system 101." Fig. 3, lower portion, also shows the contents of instruction 301. Beetcher's Fig. 3, lower portion, shows that each instruction 301 consists of four fields 302 to 305, totaling 48 bits. These are the Operation code field 302 (16 bits); the Version field 303 (8 bits); the Unused field 304 (8 bits); and the Product Number field 305 (8 bits). Beetcher col. 6:50-55 describes these four fields as follows:

...Field 305 is unused. Operation code 302 is the verb portion of the object code instruction, identifying the operation to be performed. Version 303 identifies

the version level of the software module. Product number 304 identifies the product number associated with the software module.

VII.2.6 Description of How Customer Computer 101 Uses Beetcher's entitlement verification instruction 301

21. Beetcher explains how customer computer system 101 executes instruction 301 in connection with Fig. 10. Beetcher's Brief description of Fig. 10 states: "FIG. 10 is a block diagram of the steps required to verify entitlement during execution of a software module according to the preferred embodiment." Beetcher Fig.10 shows that Beetcher's computer program determines if a fetched instruction, is an instance of instruction 301. See steps 1001, "Fetch next instruction" and step 1004, "Trigger?". Beetcher col. 10:51-54 explains this process, stating:

The process for executing a software module according to the preferred embodiment is shown in FIG. 10. System 101 executes the module by fetching (step 1001) and executing (step 1002) object code instructions until done (step 1003). If any instruction is an entitlement verification triggering instruction 301 (step 1004) check lock function 422 is invoked.

22. Thus, Beetcher discloses that instruction 301 is executed, and instruction 301 causes customer computer 101 to invoke check lock function 422.

VII.2.7 Beetcher Discloses that Check lock function 422 compares data in entry 601 in product lock table 460 to the product number contained in the Product Number field 304 in instruction 301

23. Beetcher discloses that check lock function 422 compares data in entry 601 in product lock table 460 to the product number contained in the Product Number field 304 in instruction 301. See Beetcher col. 10:54-65, which states:

...Check lock function 422 accesses the product lock table entry 601 corresponding to the product number contained in the triggering instruction at step 1005. If the **version number in product lock table 460** is equal to or greater than the **version number 303 contained in triggering instruction 301**, the software is entitled to execute (step 1006). In this case, check lock function 422 takes no further action, and the system proceeds to execute the next object code instruction in the software module. If the software is not entitled, check lock function generates an exception condition, causing control to pass to exception handler 432, which will terminate program execution (step 1007).

24. While not essential to this analysis, for completeness, note that Beetcher indicates that it is Operation code 302 of instruction 301 that instructs customer computer 101 to invoke check lock function 422. This is because Beetcher col. 6:51-52 states that "Operation code 302 is the verb portion of the object code instruction, identifying the operation to be performed." The operation performed by step 1004, is "Trigger?", which is shown to be a decisional step, that is, a

result of a comparison to something. That requires comparison of a value in the compiled object code to a predetermined value. Operation code 302 is the only set of bits of instruction 301 whose function is not specified for some other purpose, therefore the only portion of instruction 301 that can have such a predefined value. Moreover, "operation to be performed" is consistent identifying some predefined value against which a comparison to some other value is performed.

25. Beetcher Fig. 10 shows that Beetcher's computer program determines whether the instruction 301 indicates entitlement, by comparison of the version information in instruction 301, with the corresponding version number obtained from the entitlement key, in step 1006, "Entitlement Version < Product Version?". Beetcher states this expressly, at col. 10:56-65, which reads:

...If the **version number in product lock table 460** is equal to or greater than the **version number 303 contained in triggering instruction 301**, the software is entitled to execute (**step 1006**). ... If the software is not entitled, check lock function generates an exception condition, causing control to pass to exception handler 432, which will terminate program execution (**step 1007**).

26. As explained above, Beetcher discloses that product lock table 460 stores data obtained from the encrypted entitlement key 111.

27. Finally, Fig. 10 shows that step 1003 "More instruction?" executes when the software is determined in step 1006 to be entitled to run, and that this step merely loops back to instruction 1001, which fetches the next instruction in the object code, or terminates if there are no next instruction. And Fig. 10 shows that step 1007 "...Condition to Abort" terminates that program, if the software is determined in step 1006 to not be entitled to run.

VII.2.8 Beetcher's Customer Computer 101 Does Not Decode Beetcher's entitlement verification instruction 301

28. As shown above, instruction 301 is executed, just like any other instruction. Beetcher does not disclose decoding instruction 301, and Beetcher does not disclose a decoding routine to do so.

29. Beetcher also does not disclose using licensing information input in response to the prompt, that is, information contained in encrypted entitlement key 111, in a decoding routing to decode instruction 301. Therefore, instruction 301 does not correspond to claim 11's "first license code encoded in said software product." Therefore, Beetcher's entry of encrypted entitlement key 111 and decryption of that key do not correspond to "using license information entered via said input in response to said prompt in a routine designed to decode a first license code encoded in said software product."

VII.2.9 Beetcher's Customer Computer 101 Does Not Have a Routine to Decode Beetcher's entitlement verification instruction 301

30. As noted above, Beetcher discloses executing instructions 301. Just like any other instruction, execution of instruction 301 does not involve decoding. Beetcher contains no disclosure of a routine designed to decode instruction 301.

VII.2.10 Beetcher's Customer Computer 101 Does Not Use Information from the Encrypted Entitlement Key 111 to Decode Beetcher's entitlement verification instruction 301

31. As noted above, Beetcher discloses customer computer 101 decrypting and storing data from encrypted entitlement key 111, and executing instructions 301. The only relation between the entitlement data in key 111 and the instruction 301 that Beetcher discloses is the comparison of bits of instruction 301 to bits derived from encrypted entitlement key 111. So Beetcher does not disclose using the encrypted entitlement key or information contained therein to decode instruction 301.

VII.3 Claim 11: "loading a software program ... said software product outputting a prompt for input of license information; and "

32. Beetcher describes a system in which the customer computer 101 (on which the Beetcher's software module is installed) prompts the user for Beetcher's encrypted entitlement key 111, and must receive that key in order to run.

VII.3.1 Beetcher Discloses Customer Computer 101 Prompts for Input of Encrypted Entitlement Key 111

33. Beetcher discloses the computer prompting for encrypted entitlement key 111. Beetcher discloses that the user inputs the encrypted form of that key, encrypted entitlement key 111. For example, Beetcher's Summary of the Invention section begins at col. 4:1-9, which state:

Software is distributed according to the present invention without entitlement to run. **A separately distributed encrypted entitlement key** enables execution of the software. The key includes the serial number **of the machine** for which the software is licensed, together with a plurality of entitlement bits indicating which software modules are entitled to run on the machine.

34. And Beetcher explains in the next sentences at col. 4:9-14 that this encrypted entitlement key is decrypted *after being input* to the computer:

A secure decryption mechanism is contained **on the machine**. The decryption mechanism fetches the machine serial number and uses it as a key **to decrypt the entitlement key**. The entitlement information is then stored in a product lock table in memory.

This passage clarifies that what the computer receives is the encrypted key, by explaining that the encrypted key is decrypted after receipt in the computer.

35. Further, Beetcher clarifies that the user of the computer receives only the encrypted entitlement key, not the unencrypted entitlement key. That is, Beetcher col. 5:59-61 explains that "Encrypted entitlement key 111 is sent from the software distributor to the customer by mail, telephone, or other appropriate means." And Beetcher clarifies that the computer has a

mechanism for decrypting the key. That is, Beetcher col. 6:66-67 then explains that "Computer system 101 contains means for receiving and decoding [sic; decrypting] encrypted entitlement key 111." And Beetcher's summary of the 4 steps of Beetcher's invention at col. 8:53-67, specifies that the computer receives and decrypts the entitlement key. That is, Beetcher states at col. 8:60-62 that "In the third part, computer system 101 *receives, decodes* [sic; decrypts] *and stores entitlement key 111*, and sets product lock table 460."

36. Beetcher describes no mechanism for handling receipt of unencrypted entitlement key 200. Beetcher never suggests that the user input unencrypted entitlement key 200.

37. Furthermore, NFOA page 12:12-14 states "*Beetcher details that the customer enters entitlement key 111, i.e., license information, in response to the prompt.*" Thus, the NFOA asserts that encrypted entitlement key 111 is the licensing information.

38. Thus, Beetcher describes a system in which the customer computer 101 (on which the Beetcher's software module is installed) prompts the user for Beetcher's encrypted entitlement key 111, and must receive that key in order to run.

VII.3.2 Claim 11 defines a process in which the licensing information input in response to the prompt is not part of the loaded software product

39. Claim 11 requires "license information" be "input in response to said prompt" by the software product loaded on the computer. That follows from claim 11's limitation "**loading a software product on a computer.**" It follows necessarily because the sequence of claimed steps, loading, prompting, and then using, are predicated on the occurrence of the earlier steps. The software product cannot prompt, unless it has been loaded. The software product cannot use information input in response to a prompt, unless the prompt has already occurred which means the software has already been loaded. Therefore, claim 11 defines a process in which the licensing information input in response to the prompt is not part of the loaded software product.

VII.3.3 Beetcher's Encrypted Entitlement Key 111 Is Input In Response to a Prompt, Does Not Correspond to Claim 11's Installed Software Product

40. Like claim 11's prompt, Beetcher's encrypted entitlement key 111 is input in response to Beetcher's prompt.

41. Beetcher defines his installed software module as object code. See Beetcher's Brief Description of Fig. 3, col. 4:54-55 ("FIG. 3 shows the contents of a typical executable software module") and description at col. 6:41-45:

In the preferred embodiment, software modules are distributed as **compiled object code**. A typical software module 300 is shown in FIG. 3. The software module **comprises a plurality of object code instructions capable of executing on computer system 101.**

42. Object code is a set of instruction codes that is understood by a computer, as defined in Attachment 13, which is the definition of "Object Code" from the Technopeida website. Beetcher's encrypted entitlement key 111 is not object code. Beetcher's encrypted entitlement key 111 is not a set of instructions that is understood by a computer.

43. Moreover, Beetcher col. 8:60-65 describes his entitlement key as distinct from his

software modules, at col. 8:60-65:

...In the **third part**, computer system 101 receives, decodes and stores entitlement key 111, and sets product lock table 460. In the **fourth part**, the software module executing on system 101 causes system 101 to verify entitlement upon encountering an entitlement verification instruction.

Thus, Beetcher's installed software module does not include encrypted entitlement key 111. Therefore, Beetcher's encrypted entitlement key 111 (input in response to Beetcher's prompt) and its unencrypted version of the key and information therefrom that is stored in memory do not correspond to claim 11's loaded software product.

VII.4 Claim construction, Claim 11: "encode ... in a routine designed to decode"

44. The construction of "encode ... in a routine designed to decode" in claim 11 is relevant to the rejections. The subject patent makes it perfectly clear (1) that "encode" and "decode" have the meanings associated with those terms in the digital data arts, and (2) that "routine" refers to the part of the installed software code. Software function only in digital processing. In digital processing, encode and decode mean changes in digital representation of information. Consequently, claim 11's encode and decode refer to changes in digital representations of information. This is clear from the specification, the claims, and the general definitions of encode and decode.

Specification

45. The Abstract of the subject patent begins:

An apparatus and method for **encoding and decoding** additional information **into a digital information** in an integral manner. More particularly, the invention relates to a method and device for data protection. The "Field of the Invention" section, first sentence, states that:

The invention relates to the protection of *digital* information.

46. The Background of the Invention, first sentence, states:

Increasingly, commercially valuable information is being created and stored in "*digital*" form.

47. This clearly specifies the invention is directed to encoding and decoding of information represented in *digital* form. And the specification repeatedly refers to decoding using a key, which requires a digital decoding algorithm, and that the software may include the encode and decode functions. For example, the specification states:

To decode the information, a predetermined key is used **before playing the digital information** at steps 140 and 150.

A key-based decoder can act as a "**plug-in**" **digital player** of broadcast

signal streams without foreknowledge of the encoded media stream. Moreover, the data format orientation is used to partially scramble data in transit to prevent unauthorized descrambled access by **decoders that lack authorized keys**. A distributed key can be used to unscramble the scrambled content because a decoder would understand how to process the key.

The same keys can be used to later validate the embedded digital signature, or even fully decode the digital watermark if desired

48. Moreover, the specification refers to "code resource" of an "application" which means software. The specification states:

Note further that the **application contains a code resource which performs the function of decoding an encoded code resource from a data resource**. *** 3) Once it has the license code, it can then generate the proper decoding key to access the essential code resources.

Software necessarily functions on digital computers and therefore is limited to digital processing.

Claims

49. Claim 11 is specifically limited to software, and therefore necessarily has the meaning for encode and decode specific to digital representations. That meaning is one of representation of information digitally.

50. Moreover, claim 11 recites "first license code encoded in said software product." In this regard, the specification col. 13:36-48 states "This method, then, is to choose **the key** so that it corresponds, **is equal to**, or is a function of, **a license code** .. Alternatively, the key, possibly random, can be stored as a data resource and **encrypted** with a derivative of the license code." Thus, a corresponding disclosure to what claim 11 recites appearing in the specification refers to encoding as changing the digital representation of data.

51. Further, claim 11 requires software having "a *routine* designed to decode" encoded information. The specification explains that the claimed routine is part of the software "application [that] contains a code resource which performs the function of decoding." That is, claim 11 defines a **software routine designed to decode the license code encoded in the software product**. Software acts on digital data. As such, the encoding and decoding defined by claim 11 are limited to digital encoding, which means changing the digital representation of information.

Example of encoding/decoding based upon the specification

52. Moreover, the specification mentions both ASCII and binary encoding. Therefore, to make the concept of the claimed "encode... routine designed to decode," more concrete, and the meaning of digital representation clear, consider the following example of a digital encoding representing the number ten, in base 10 and base 2, using the 7 bit ASCII specification.

53. Assuming the ASCII 7 bit specification, "10" in base 10 would be the ASCII code for numeral "1", which are "011 0001", followed by the ASCII code for numeral "0", which are "011 0000." So the number ten in base 10 represented in 7 bit ASCII code would appear in digital memory to be the following two sets of 7 digits: "011 0001", "011 0000".

54. But representing the number "10" in base 2 results in the digital bits "1010". So when the number 10 is encoded in base 2, and represented in the 7 bit ASCII specification, there are a sequence of four characters in sequence. So representing "10" in binary, in 7 bit ASCII would result on the following four sets of 7 digits: "011 0001", "011 0000", "011 0001", "011 0000".

55. Converting from binary to decimal, and from binary and decimal to 7 bit ASCII are examples of what encoding and decoding mean in the digital arts. That is how the same value (that is information) is represented differently. Encoding, in this patent, means changing the digital representation of information.

Definition in the Art of encoding and decoding

56. Moreover, encoding and decoding have clear meanings in the digital computer arts. A search on "what does encoding mean in the computer" using the Google search engine, returns the result stating:

In computers, encoding is the process of putting a sequence of characters (letters, numbers, punctuation, and certain symbols) into a specialized format ...Decoding is the opposite process -- the conversion of an encoded format back into the original sequence of characters. Nov 14, 2005

A copy of the screen image showing the Google search query and quote definitions is Attachment 11.

57. The URL link cited by the Google search is:

<https://searchnetworking.techtargt.com/definition/encoding-and-decoding>.

58. A copy of the web page provided by

<https://searchnetworking.techtargt.com/definition/encoding-and-decoding> Attachment 12.

59. Attachment 12 provides definitions of encoding and decoding. Attachment 12, and is quoted below (emphasis added):

Definition encoding and decoding
Posted by: Margaret Rouse
WhatIs.com

In computers, encoding is the process of putting a sequence of characters (letters, numbers, punctuation, and certain symbols) **into a specialized format** for efficient transmission or storage. **Decoding is the opposite process** -- the conversion of an encoded format **back into the original sequence of characters**. Encoding and decoding are used in data communications, networking, and storage. The term is especially applicable to radio (wireless) communications systems.

The **code used by most computers for text files is known as ASCII** (American Standard Code for Information Interchange, pronounced ASK-ee). ASCII can depict uppercase and lowercase alphabetic characters, numerals, punctuation marks, and common symbols. Other commonly-used codes include

Unicode, BinHex, Uuencode, and MIME. In data communications, Manchester encoding is a special form of encoding in which the binary digits (bits) represent the transitions between high and low logic states. In radio communications, numerous encoding and decoding methods exist, some of which are used only by specialized groups of people (amateur radio operators, for example). The oldest code of all, originally employed in the landline telegraph during the 19th century, is the Morse code.

The terms encoding and decoding are often used in reference to the processes of analog-to-digital conversion and digital-to-analog conversion. In this sense, these terms can apply to any form of data, including text, images, audio, video, multimedia, computer programs, or signals in sensors, telemetry, and control systems. Encoding should not be confused with encryption, a process in which data is deliberately altered so as to conceal its content. **Encryption can be done without changing the particular code** that the content is in, and encoding can be done without deliberately concealing the content.

60. This passage, in the last paragraph, explains that encoding relates to both digital and analog. While generally true, the subject patent is specifically directed to digital information, and the claims are limited to software programs, which only functions on digital information. The specification and claims therefore excludes claim constructions reading on analog encoding and decoding.

Definition of “routine”

61. Claim 11 recites “routine”. To avoid doubt, in the software arts, a “routine” defines a section of a computer program that performs a particular task, as stated in Attachment 14, which is the definition of “routine” from the Webopedia website.

62. The definitions of decode are important because Beetcher does not disclose that its software modules 300 loaded on customer computer 101 contains a routine designed to decode instructions when 301; to change the digital representation or instruction 301.

VII.5 Response to the NFOA’s Assertions Regarding Beetcher

63. The NFOA concludes that Beetcher discloses claim 11's "said software product using license information entered via said input in response to said prompt *in a routine designed to decode* a first license code *encoded* in said software product." NFOA page 12:7 to 17:1. This conclusion is incorrect. The NFOA lacks application of asserted facts to this claim language and contains irrelevant and incorrect statements of fact regarding Beetcher, discussed below.

VII.5.1 Contrary to the NFOA, Beetcher *does not* disclose using the entitlement key (111, 200; encrypted, unencrypted) in a routine designed to decode anything in Beetcher’s software, as required by claim 11

64. NFOA page 12:15-17, states “After entering that key, Beetcher teaches that the customer's computer uses a decode key to initiate unlock routine 430 **to decode the license code encoded in the software product.**”

65. This assertion of fact is incorrect. Beetcher does not disclose that unlock routine 430 decodes a license code encoded in the software product. Beetcher discloses that **unlock routine**

430 acts only on the encrypted entitlement key, not Beetcher's software module 300, as shown in the following passages:

66. Beetcher col. 7:39-42:

...Unlock routine 430 uses the unique machine key to decodes entitlement key 111, and **stores encrypted entitlement key 111** in encoded product key table 450.

Beetcher col. 9:55-60:

The entitlement key is passed to unlock routine 430, which handles the decoding process. Unlock routine 430 causes get machine key function 420 to retrieve the machine serial number and generate the machine key at 902. Unlock routine 430 then uses the machine key to decode the entitlement key 111 at step 903.

Beetcher col. 9:67 to 10:5:

...Unlock routine 430 scans each product entitlement flag 205 in the decoded key (step 904). If the product entitlement flag is set to '1' (indicating entitlement) at step 905, the corresponding entry in product key table 450 is replaced with the new entitlement key [sic; key's], version number, and charge group value at step 905.

67. These passages show that Beetcher's unlock routine 430 performs the decryption of encrypted entitlement key 111 and stores certain information (version numbers and charge groups) read from the decrypted entitlement key, in computer memory (key table 450 and product lock table 460). As explained above, encrypted entitlement key 111 and its decrypted version are not part of Beetcher's software module. Therefore, Beetcher's entitlement key does not correspond to the installed software product defined by claim 11. Therefore, Beetcher's "decoding" [sic; decrypting] of the encrypted entitlement key by routine 430 is not decoding of a license code encoded in Beetcher's installed software product. As explained herein above, the fact that Beetcher's computer decrypts and then stores in memory in tables 450, 460, certain information contained in the entitlement key, does not make that information "encoded" in Beetcher's software module.

68. In any case, Beetcher discloses the decryption and storing of data contained in the entitlement key as separate from Beetcher's software module. See col. 8:60-67:

In the third part, computer system 101 receives, decodes and stores entitlement key 111, and sets product lock table 460. **In the fourth part, the software module executing on system 101** causes system 101 to verify entitlement upon encountering an entitlement verification instruction. The first two parts are performed under the control of the software distributor. The last two

are performed on the customer's system 101.

Thus, the decrypting and storing of information contained in encrypted entitlement key 111 could not correspond to the decoding of a "license code encoded *in said software product*," as required by claim 11. Thus, storing information contained in Beetcher's encrypted entitlement key 111 in memory of customer computer 101 does result in that information becoming "encoded **in said software product**," as required by claim 11.

69. Further, Beetcher does not disclose unlock routine 430 acting on instruction 301. Accordingly, Beetcher cannot disclose unlock routine 430 decoding instruction 301.

70. In summary, Beetcher does not disclose that unlock routine 430 decodes a license code encoded in the software product.

VII.5.2 Contrary to the NFOA, Beetcher also does not disclose using the entitlement key in a routine designed to decode entitlement verification triggering instruction 301; does not disclose a routine designed to decode instruction 301; and does not disclose decoding instruction 301, all of which are required by claim 11

71. The NFOA page 12:7 to page 16 bottom concludes that Beetcher discloses claim 11's "software product using license information entered via said input in response to said prompt in a routine designed to decode a first license code encoded in said software product."

72. Specifically, **NFOA, page 12:18-20** assert that:

...Beetcher's Figures 4 and 9a, which are provided below, show the software using the key (i.e., license information) entered by the customer to decode a first license code encoded in the software product.

This statement is incorrect.

73. Beetcher states that "FIG. 4 shows the hardware and software structures required on the customer's computer system to support the software protection mechanism according to the preferred embodiment." Beetcher Fig. 4 does not show software. Nor does it show anything entered by the customer decoding anything in Beetcher's software module 300.

74. Beetcher states that "FIG. 9 is a block diagram of the steps required to decode an entitlement key and maintain a record of entitlement status on the customer's computer system, according to the preferred embodiment." Fig. 9a shows the process of receipt by customer computer 101 of encrypted entitlement key 111 (step 901) through the storage of information contained in the key in tables 450, 460 (step 904 for table 450 and step 907 for table 460). Decoding of the entitlement key and storing in memory the information contained in that key is not a disclosure of decoding a license code **in the software product**. The entitlement code, as explained in Beetcher and discussed above, is not part of Beetcher's software module 300.

VII.5.2.1 Contrary to the NFOA, Beetcher Does Not Disclose Using the Key's Version and Product Number to Decode a License Code

75. The NFOA, page 14:12-13 states "Beetcher's software product uses the key's version and product number fields to decode a license code."

76. This statement lacks citation to Beetcher. Therefore, it should be given no weight. Moreover, this statement is incorrect. Beetcher does not disclose its software product decoding a

license code. As explained above, Beetcher discloses customer computer 101 decrypting the encrypted entitlement key received by that computer, and then comparing information contained in the entitlement key to version number contained in instruction 301. Nowhere does Beetcher disclose decoding of an instruction in software module 300.

III.5.2.2 Entitlement verification triggering instructions

77. The NFOA, page 15:1-5, states:

...When compiling and translating the software code, Beetcher explains that the code includes entitlement verification triggering instructions encoded into the software. (Id. at 6:41 -58. 11 :4-59; see also id. at 4:14-23, 8:5-22, 8:56-9:20) Beetcher's triggering instructions are encoded into the software when the software code is compiled and translated, as shown in Figure 3 provided below:

78. This statement in the NFOA does not state that Beetcher discloses decoding entitlement verification triggering instructions. This assertion in the NFOA does not support rejection of claim 11.

VII.5.2.3 Triggering Instruction 301

79. The NFOA **page 15:4-5** states that "Beetcher's triggering instructions are encoded into the software when the software code is compiled and translated, as shown in Figure 3." This statement is not relevant because claim 11 requires *decoding* of a license code encoded in the software product, not encoding. Claim 11 defines using the input licensing information "to **decode** a first license code encoded in said software product." Executing instruction 301, which invokes product lock table 422, is not decoding.

VII.5.2.4 Key table 460

80. Next, the NFOA, page 15:6 to 16:1, states:

Beetcher explains that its software code verifies the customer is entitled to use the software when the code encounters a triggering instruction. When it encounters one of these instructions, Beetcher's code accesses the license key information stored in the key table 460. (Id. at 10:48-11:39; See also id, at Abstract, 8:14-22, 8:53-9:20, Fig. 10).

81. This statement in the NFOA does not state that Beetcher discloses decoding entitlement verification triggering instructions. Key table 460 contains information received from the encrypted entitlement key 111, as discussed above. This key and the information contained in key table 460 are not part of Beetcher's software module 300. Beetcher's software module accessing this information is not decoding license information contained **in the software product**, as required by claim 11. Thus, the assertions at NFOA, page 15:6 to 16:1 do not support rejection of claim 11.

VII.5.2.5 Check Lock Function 422

82. The NFOA **page 16:3-5 concludes** that "As such, a POSITA would have understood that

Beetcher uses its license information in a routine, such as check lock function 422, designed to decode a first license code encoded in a software product via the triggering instructions."

83. In response, the NFOA's conclusion that Beetcher discloses that check lock function 422 is designed to decode a first license code encoded in a software product is supported no reasoning. And it is wrong. Beetcher does not disclose that check lock function 422 is designed to decode a first license code encoded in a software product.

84. Instead, Beetcher discloses that check lock function 422 compares data in entry 601 in product lock table 460 to the product number contained in the Product Number field 304 in instruction 301. See Beetcher col. 10:54-65, quoted above in the description of check lock function 422. Performing a comparison is not decoding.

85. Beetcher discloses that check lock function 422 functions by *reading* or *accessing* values from lock table 422. See Beetcher:

...Check lock function 422 accesses product lock table 460 and *reads* one of the entries to verify entitlement. [Beetcher, col. 7:29-31; bold added for emphasis.]

...The executable code contains entitlement verification triggering instructions 301 (only one shown), which are *executed* by horizontal microcode check lock function 422. [Beetcher, col. 8:19-22; bold added for emphasis.]

Check lock function 422 *accesses* the product lock table entry 601 corresponding to the product number contained in the triggering instruction at step 1005. [Beetcher, col. 8:54-56; bold added for emphasis.]

86. The NFOA, page 16:5-11, continues, presenting the following inaccurate and uncited block quote, apparently lifted from the reexamination request, but inaccurately reproduced in the NFOA.

If any instruction is an entitlement verification triggering instruction 301 (step 1004) check lock function 422 is invoked. Check lock function 422 accesses the product lock table entry 601 corresponding to the product number contained in the triggering instruction at step 1005. If the version number in product lock table 460 is equal to or greater than the version number 303 contained in triggering instruction 301, the software is entitled to execute (step 1006). (Id. at 10:52-62, Fig. 10; Silva Declaration at ¶ 46-51).

87. This block quote contains no citation. Moreover, nothing in this block quote identifies

the source of "Id." appearing therein.

88. After searching the reexam request, the undersigned found this quote to be an inaccurate copy of a passage in the Reexam request, page 39. Reexam request page 39:6-12 states the following:

If any instruction is an entitlement verification triggering instruction 301 (step 1004) check lock function 422 is invoked. Check lock function 422 accesses the product lock table entry 601 corresponding to the product number contained in the triggering instruction at step 1005. If the version number in product lock table 460 is equal to or greater than the version number 303 contained in triggering instruction 301, the software is entitled to execute (step 1006).

At the end of this text from the reexam request is a cite to footnote 144. Footnote 144 appears in the bottom margin of page 39 of the reexam request. Footnote 144 states "¹⁴⁴ Id. at 10:49-60; see also id. at 10:48-49, 10:60-11:3; Silva Declaration at ¶¶78-82." The "Id." in the reexamination request, refers to footnote 144, which refers to Beetcher. So this passage cites, for support, Beetcher 10:49-60; 10:48-49; and 10:60-11:3. These passages in Beetcher do not support the rejection.

89. Beetcher 10:48-49 reads "The process for executing a software module according to the preferred embodiment is shown in FIG. 10." All Fig. 10 shows is that the "trigger" in step 1004 is compared to the product lock table entry, in step 1006. Nothing here discloses decoding a trigger. Therefore, Fig. 10 does not disclose claim 11's "**routine designed to decode**" such a trigger "**encoded in said software product.**"

90. Beetcher 10:49-60 reads:

System 101 executes the module by fetching (step 1001) and executing (step 1002) object code instructions until done (step 1003). If any instruction is an entitlement verification triggering instruction 301 (step 1004) check lock function 422 is invoked. Check lock function 422 *accesses* the product lock table entry 601 corresponding to the product number contained in the triggering instruction at step 1005. If the version number in product lock table 460 is *equal to or greater than* the version number 303 contained in triggering instruction 301, the software is entitled to execute (step 1006).

91. All Beetcher 10:49-60 discloses is accessing entitlement verification triggering instruction 301 and comparing that to product lock table data. Nothing here discloses decoding an entitlement verification triggering instruction 301. Therefore, Beetcher 10:49-60 does not disclose claim 11's "**routine designed to decode**" such a triggering instruction "**encoded in said software product.**"

92. Beetcher 10:60-11:3 reads:

In this case, check lock function 422 takes no further action, and the system proceeds to execute the next object code instruction in the software module. If the software is not entitled, check lock function generates an exception

condition, causing control to pass to exception handler 432, which will terminate program execution (step 1007). The system does not save the results of an entitlement check which shows that the software is entitled. Therefore, when a triggering instruction is again encountered in the software module, the system again verifies entitlement as described above.

93. All Beetcher 10:60-11:3 discloses is the results of the comparison shown in Fig. 10, step 1006. The results of the comparison do not related to operations performed on the data used in the comparison. Therefore, Beetcher 10:60-11:3 does not disclose claim 11's "**routine designed to decode**" anything "**encoded in said software product**."

94. In summary, contrary to the NFOA, Beetcher **does not** disclose that check lock function 422 is designed to decode a first license code encoded in a software product via the triggering instructions.

VII.5.2.6 Execution of Instruction 301

95. Next, the NFOA, page 16:12-26, states that:

Moreover, Beetcher teaches that the triggering instructions will be encoded into the code resources controlling software functionality: [quote lines 17-22, *lacking citation*, omitted] And Beetcher details that "the triggering instruction is also a direct instruction to perform some other useful work I [sic] Execution of the triggering instruction causes system 101 to perform some other operation simultaneous with the entitlement verification." (Id. at 6:58-65 (Beetcher specifies that these functions are those "which do not require that an operand for the action be specified in the instruction."); Silva Declaration at ¶ 52-53).

96. These statements fail to assert that Beetcher discloses a "**routine designed to decode**" such a triggering instruction "**encoded in said software product**." The fact that Beetcher uses portions of triggering instructions for purposes of effecting the functionality of Beetcher's software does not indicate encoding or decoding; it merely indicates execution of the instruction.

VII.7 Contrary to the NFOA, Patent Owner's Description of Beetcher in the Patent Owner's Statement, Was Accurate

97. The NFOA contains a section responding to the Patent Owner Statement. See NFOA pages 89-108. This section contains the following passage at NFOA 93:6-15:

Patent Owner specifically argues that "encrypting and decrypting the key is not encoding or decoding and [sic; the] software module ... " and that "Beetcher does not disclose encoding or decoding of software using a key, or decoding license code encoded in software."

In response, the Examiner respectfully submits that the described process starts by placing entitlement verification triggers in the object code. The process involves creation of a program template that is input into a translator along with

product numbers and version numbers. From this the encoded code is generated, where this code is encoded with hidden functionality not available until it is decoded. (see column 9, lines 1-20).

98. The Examiner's statements of "responses" implies that the Examiner disagrees with the Patent Owner's statements of fact. The Examiner's statement, like the statement of the rejection, leaves the reader at sea as to the meaning of these statements. These statements are improper because they imply that the Patent Owner's assertion of fact are incorrect, without expressly stating so.

99. The implications based upon the foregoing "response" of the Examiner are incorrect. Patent Owner's statement in the Patent Owner Statement were and remain correct. Specifically:

100. It is a fact that "encrypting and decrypting the key is not encoding or decoding and [sic; the] software module ... "

101. It is a fact that "Beetcher does not disclose encoding or decoding of software using a key."

102. And it is a fact that "Beetcher does not disclose decoding a license code encoded in software."

103. The Examiner's response cites Beetcher column 9:1-20. That passage does not contradict the Patent Owner's statements of fact. That passage reads:

When the software distributor compiles a software module, it must place the entitlement verification triggers in object code. A typical such process, which takes place on development system 125, is shown in FIG. 7. Source code is generated by a programmer in the normal fashion, without inclusion of entitlement verification triggers. The source code is input into compiler 126 at step 701 to produce a program template at 702. The program template comprises machine instructions at virtual machine level 404 (i.e. above machine interface 405). The program template serves as input to translator 127 at step 704, along with its product number and version number identification. Translator 127 automatically generates a substantial number of entitlement verification triggers, inserts them in random locations in the object code, and resolves references after the triggers have been inserted. The resultant executable object code form of the software module which is output at 705 contains the embedded triggers. This executable form of the module comprises object code instructions at executable code level 403.

104. Beetcher column 9:1-20 does not disclose or suggest that the entitlement key (either in its unencrypted form, unencrypted entitlement key 200, see col. 9:34-36, col. 9:42-48) is involved in the process of encoding the software module.

105. In fact, Beetcher explains that the compiling of the software code discussed in the paragraph at col. 9:1-20 occurs in a distinct part from the generation of the entitlement key 200 discussed in the paragraph at col. 9:21-48. Specifically, Beetcher provides an overview of the four steps involved in his process in the paragraph at col. 8:53-67. There, in pertinent part, Beetcher states:

The operation of a software module on computer system 101 in accordance with the preferred embodiment of the present invention will now be described. There are four parts to this operation. **In the first part**, a plurality of entitlement verification triggering instructions are placed in the executable object code form of the software module. **In the second part, an encrypted entitlement key 111 authorizing access to the software module is generated.** In the third part, computer system 101 receives, decodes and stores entitlement key 111, and sets product lock table 460. In the fourth part, the software module executing on system 101 causes system 101 to verify entitlement upon encountering an entitlement verification instruction.

In this passage, Beetcher indicates no relation between placing the entitlement verification triggering instructions 301 in the object code and generating and entitlement key 111 authorizing access to the software module. Moreover, the description of the compiling in the "first part" suggests that the generation of the encryption key in the "second part" occurs after compiling. In which case, there is no encryption key in existence at the time of compiling.

106. Beetcher's disclosure clearly indicates that instruction 301 is not a function of entitlement key 200. That means instruction 301 is not encrypted using entitlement key 200 and therefore cannot be decrypted or encoded using the entitlement key.

107. In fact, Beetcher teaches a method that improves upon the prior art by providing *exactly the same software module to all users*, but different encrypted versions of entitlement key 200 to each user. Beetcher teaches that, by explaining that its invention provides the benefit of a simplified method of distribution of the same software module to all of its customers; not software product that are individually encoded or encrypted based upon each customer's unique encryption key. See, e.g., col. 2:49-52 in the Background identification of the prior art ("Another restricted entitlement method is to encode *user or machine specific information* in the software itself.") and col. 3:51-53 in the Summary of the Invention section indicating one of Beetcher's goals is to avoid that prior art ("Another object of this invention is to *simplify the distribution system* of a distributor of software protected against unauthorized use"); followed by Beetcher's express disclosure that each of its software modules distributed to all of its users are identical at col. 4:34-39 ("Because the software itself does not contain any entitlement, **no restrictions on the distribution are necessary.** In the preferred embodiment, the distributor of software may record multiple software modules on a single generic medium, and **distribute the same** recorded set of **modules to all its customers.**")

108. Thus, Beetcher clearly discloses that its software module are neither encoded nor encrypted using Beetcher's encryption key 200.

VII.8 Claim 12

109. Claim 12 reads:

12. A method for encoding software code using a computer having a processor and memory, comprising:
storing a software code in said memory;

wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system; and encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key encoded software code; and wherein, when installed on a computer system, said first license key encoded software code will provide said specified underlying functionality only after receipt of said first license key.

VII.8. Claim 12 Claim Construction, “encoding using at least a first license key and an encoding algorithm”

110. Claim 12 recites “encoding, by said computer using at least a first license key and an encoding algorithm, said software code, to form a first license key encoded software code;”

111. Claim 12 defines encoding “software code” by using both the first license key and the encoding algorithm to encode the software code, stating “encoding using at least a first license key and an encoding algorithm.” The specification col. 13: 8-49 explains this process, stating:

One method of the present invention is now discussed. When code and data resources are compiled and assembled into a precursor of an executable program the next step is to use a utility application for final assembly of the executable application. The programmer marks several essential code resources in a list displayed by the utility. The utility will choose one or several essential code resources, and **encode them** into one or several data resources **For the encoding of the essential code resources, a "key" is needed.** ... The purpose of this scheme is to make a particular licensed copy of an application **distinguishable from any other.** ... The assembly utility can be supplied with a key generated from a license code generated for the license in question. Alternatively, the key, possibly random, can be stored as a data resource and encrypted with a derivative of the license code. **Given the key, it encodes one or several essential resources into one or several data resources.**

112. This passage clarifies that the key chosen for the encoding process is what makes the resulting code “distinguishable from any other” copy. That is, the key is an input to the encoding algorithm. This is the conventional meaning of a key in the encoding arts. It is that which is used in the encoding algorithm to encode some other data.

113. To avoid doubt, this passage does not describe encoding software code that includes a key in the software code. Instead, it clearly describes conventional key encoding of something else, and the something else in this case are essential code resources of the software code.

114. Consistent with that description in the specification, the broadest reasonable construction of claim 12's “**encoding using at least a first license key** and an encoding algorithm” refers to an encoding process in which the encoding algorithm uses a key to encode something else. And claim 12 defines that the encoding is performed on “said software code.” Claim 12 therefore defines using a key based encoding algorithm and its key to encode software code.

115. Using a key based encoding algorithm and a particular key for encoding is referred to

herein below as key based encoding.

VII.9 Beetcher does not disclose “encoding using at least a first license key and an encoding algorithm.”

116. Beetcher does not disclose key based encoding to form its software module. Instead, Beetcher discloses the following process of forming its distributed software module.

When the software distributor compiles a software module, it must place the entitlement verification triggers in object code. A typical such process, which takes place on development system 125, is shown in FIG. 7. Source code is generated by a programmer in the normal fashion, without inclusion of entitlement verification triggers. The source code is input into compiler 126 at step 701 to produce a program template at 702. The program template comprises machine instructions at virtual machine level 404 (i.e. above machine interface 405). The program template serves as input to translator 127 at step 704, along with its product number and version number identification. Translator 127 automatically generates a substantial number of entitlement verification triggers, inserts them in random locations in the object code, and **resolves references after the triggers have been inserted**. The resultant executable object code form of the software module which is output at 705 contains the embedded triggers. This executable form of the module comprises object code instructions at executable code level 403.

117. So Beetcher teaches conventional compiling of its high level code, followed by inserting instructions 301 at various locations therein, and then followed by resolving the references (addresses) in the compiled code caused by the insertion of instances of instructions 301. Beetcher does not teach key based encoding to form its software module. Accordingly, Beetcher does not anticipate claim 12.

118. The NFOA, page 25 argues that Beetcher discloses and “encoding algorithm to encode the first license key” and corresponds the key to instruction 301. For the foregoing reasons, instruction 301 does not correspond to the key required by claim 12 for encoding the software program.

VII.10 Claim 12 Claim construction, “only after receipt of said first license key.”

119. Claim 12 recites “wherein, when installed on a computer system, said first license key encoded software code will provide said specified underlying functionality only after receipt of said first license key.” “Only after receipt of” clearly indicates that the first license key must be received by the software after the software is installed on the computer.

VII.11 Beetcher Does not Disclose Claim 12s “only after receipt of said first license key.”

120. Beetcher’s software module 300 does not receive Beetcher’s instruction 301 after being installed on Beetcher’s customer computer 101. Therefore, Beetcher’s instruction 301 does not correspond to claim 12’s receipt of a first license key.

121. The NFOA, page 26 states that “Beetcher discloses element 12.4. Specifically, Beetcher

explains that its first license key encoded software code provides the specified underlying functionality only after receipt of the first license key. ... Beetcher teaches **encoding the triggering instructions into the software code** that is decoded **via the first license key.**”

122. This statement is logically flawed because it is inconsistent with the NFOA’s correspondence of license key to instruction 301 in the prior section of the NFOA. The claimed license key cannot correspond to both instruction 301 and the encrypted entitlement key 111. Moreover, what Beetcher discloses as being received after the software is installed is only the encrypted entitlement key. The NFOA conveniently fails to mention that the encrypted entitlement key is what the NFOA previously equated to the licensing information in discussing claim 11.

123. Claim 12 requires that “said first license key encoded software code will provide said specified underlying functionality only after receipt of said first license key.” So the first license key can only correspond to encrypted entitlement key 111. In that case, Beetcher fails to disclose claim 12’s “encoding, by said computer using at least a first license key” because Beetcher does not disclose encoding using encrypted entitlement key 111.

VII.12 Claim 12 Claim Construction “first license key”

124. The specification col. 13:27 to 14:3 states “For the encoding of the essential code resources, a **‘key’ is needed.** ... The purpose of this scheme is **to make a particular licensed copy of an application distinguishable from any other.** It is not necessary to distinguish every instance of an application, merely every instance of a license. A **licensed user may then wish to install multiple copies** of an application, legally or with authorization. ... Note that the application can be copied in an uninhibited manner, but must contain the license **code issued to the licensed owner,** to access its essential code resources.” This passage clarifies that the license code is specific to the licensed owner. And the specification col. 13:36-37 then explains “This method, then, is to choose the key so that it corresponds, is equal to, or is a function of, a license code...”

125. So the specification explains that the key is also specific to the licensed owner. Claim 12 therefore defines the “first license key” to be specific to the licensed owner.

VII.13 Beetcher Does Not Disclose Instruction 301 is a “first license key,” as Defined by Claim 12

126. Beetcher discloses that instruction 301 stores software version information, not a license key. And Beetcher discloses that the same software module 300 is distributed to all of its customers containing this generic instruction 301. Accordingly, corresponding instruction 301 to a license key that must be received by the installed software program is incorrect for this additional reason.

127. The NFOA, for example at page 23 identifies instruction 301 as corresponding to the claimed “licensing key.” For the reason just noted, that correspondence is incorrect.

VII.14 Claim 13

128. Claim 13 reads as follows:

13. A method for encoding software code using a computer having a processor and

memory, comprising:
storing a software code in said memory;
wherein said software code comprises a first code resource and provides a specified underlying functionality when installed on a computer system; and
modifying, by said computer, using a first license key and an encoding algorithm, said software code, to form a modified software code; and
wherein said modifying comprises encoding said first code resource to form an encoded first code resource;
wherein said modified software code comprises said encoded first code resource, and a decode resource for decoding said encoded first code resource;
wherein said decode resource is configured to decode said encoded first code resource upon receipt of said first license key.

VII.15 Contrary to the NFOA, Beetcher does not disclose Claim 13's "wherein said modified software code comprises said encoded first code resource, and a decode resource for decoding said encoded first code resource"

129. The NFOA page 35:15 to page 16:3 alleges the claimed decode resource of the modified software code, for decoding said encoded first code resource, corresponds to the elements inside the dashed perimeter of the copy of Beetcher Fig. 4 in paragraph 96 of the Silva declaration. Those elements consist of; unlock decode key **430**; exception handler **432**; check lock **422**.

130. The NFOA is wrong, and the Silva declaration's conclusion is wrong. Moreover, the conclusion does not follow from the Silva declaration's statements paragraph 96's statements of fact. Paragraph 96 of the Silva declaration states:

... Beetcher discloses that executing a trigger 301 invokes check lock function 422, which results in accessing "unlock (decode key)" function 430 upon confirmation that the customer possesses the software's license key.

Based upon these statements, Mr. Silva concludes in paragraph 96 that "Beetcher explains that its modified software code includes a decode resource for decoding the encoded first code resource."

131. Mr. Silva does not state any fact showing decoding of Beetcher's installed software module 300. And Mr. Silva and the NFOA are wrong.

132. First, Mr. Silva and the NFOA are wrong in concluding that check lock function 422 is part of installed software module 300. Beetcher, Fig. 4, clearly shows that check lock function 422 is microcode and therefore not part of installed software module 300. Moreover, Beetcher expressly states that this microcode is hardware, not software, and therefore could not be part of an installed software module. See Beetcher col. 7:16-31 which states:

Horizontal microcode 402 contains microcode entries interpreting the executable instruction set. It is physically stored in control store 103, which in the preferred embodiment is a **read-only memory (ROM)** which is not capable of alteration by the customer. **Entries in horizontal microcode support get machine key 420, set lock 421, and check lock 422 functions.** Get machine key

function 420 fetches a unique identifier, which in the preferred embodiment is based on the system serial number, from some permanent hardware location. Set lock function 421 accesses product lock table 460 and alters an entry in the table. The set lock function is the only microcode function capable of altering product lock table 460. Check lock function 422 accesses product lock table 460 and reads one of the entries to verify entitlement.

133. “Microcode” refers to “microcode is a layer of hardware-level instructions that implement higher-level machine code instructions,” as explained in the definition of Microcode in Wikipedia, Attachment 15. Beetcher does not disclose including in its software module 300 hardware-level instructions that implement higher-level machine code instructions. Microcode is not part of Beetcher’s installed software module 300.

134. Second, Mr. Silva and the NFOA are wrong in concluding that exception handler 432 is part of the installed software module 300, and they are wrong in concluding that exception handler 432 is part of a decode resource for decoding said encoded first code resource of the installed software.

135. Beetcher explains that function of exception handler 432 as follows.

136. First, Beetcher Fig. 4 shows only one arrow pointing into exception handler 432, which identifies the one source of input to exception handler 432 to be microcode check lock 422. Beetcher col. 7:43-46 states “Exception handler 432 responds to exception conditions raised by functions in horizontal microcode 402.” ” Fig. 7 and this passage show that exception handler 432 receives input only from microcode check lock function 422.

137. Third, Beetcher col. 10:20-41 states that ‘Upon first execution of a previously unentitled software product, an exception is generated by the system when a triggering instruction is encountered. **Exception handling routine 432 then calls unlock routine 430** to attempt to unlock the product at step 920. Unlock routine 430 then fetches the encrypted entitlement key from the appropriate entry in encoded product key table 450 at step 921, obtains the 30 machine key at step 922, and **decodes the entitlement key** at step 923. ... **The triggering instruction is then retried** and program execution continues at step 928. If entitlement is not indicated at step 924, program execution aborts at step 925.

138. This passage specifies that exception handling routine 432 calls unlock routine 430. It does not specify that exception handling routine 432 decodes the code of software module 300. Moreover, decoding [sic; decrypting] encrypted entitlement key 111 is not decoding software module 300. This is because encrypted entitlement key 111 is not part of software module 300. Furthermore, retrying triggering instruction 301 (“**The triggering instruction is then retried** and program execution continues...”) is not decoding the triggering instruction. Retrying merely means executing the instruction again to see if the updated entry 601 will result in entitlement of the program to run.

139. Finally, Mr. Silva and the NFOA are wrong in concluding that “unlock (decode key)” 430 is part of a decode resource for decoding the modified software code.

140. Beetcher described unlock routine 430 at col. 7:38-42, stating “Unlock routine 430 uses the unique machine key to decodes [sic] entitlement key 111, and stores encrypted entitlement key 111 in encoded product key table 450.” As explained herein above in the discussion of claim 11, unlock routing 430 stores license information contained in the encrypted entitlement

key in memory, in tables 450, 460. Unlock routine 430 does not act on or modify instruction 301. Accordingly, unlock routine 430 is not a “decode resource for decoding” Beetcher’s instruction 301. Accordingly, Mr. Silva and the NFOA are wrong.

VIII.16 Claim 14

141. Claim 14 reads:

14. A method for encoding software code using a computer having a processor and memory, comprising:
storing a software code in said memory;
wherein said software code defines software code interrelationships between code resources that result in a specified underlying functionality when installed on a computer system;
and
encoding, by said computer using at least a first license key and an encoding algorithm, said software code,
to form a first license key encoded software code in which at least one of said software code interrelationships are encoded.

VIII.16 Contrary to the NFOA, Beetcher Does Not Anticipate Claim 14

142. Claim 14 recites “encoding, by said computer using at least a first license key and an encoding algorithm, said software code.”

143. For the same reasons stated for claim 12, this recitation defines key based encoding; using the key as in input to the encoding function, to encode software code.

144. For the same reasons discussed for claim 12, Beetcher does not disclose this encoding, and therefore does not disclose the claimed “first license key encoded software code” encoding at least one “code interrelationship.”

IV. Jurat

17. I have been warned that willful false statements and the like are punishable by fine or imprisonment, or both (18 U.S.C. 1001) and may jeopardize the validity of the application or any patent issuing thereon. All statements I make in the declaration I either know to be true or on information and belief I believe them to be true.

Signed:

SCOTT MOSKOWITZ

Y:\Clients\SCOT Scott A Moskowitz and Wistaria Trading, Inc\90014138, USP9104842, SCOT0014-4\Drafts\Attachment17_132MoskowitzDeclaration.wpd

key in memory, in tables 450, 460. Unlock routine 430 does not act on or modify instruction 301. Accordingly, unlock routine 430 is not a "decode resource for decoding" Beetcher's instruction 301. Accordingly, Mr. Silva and the NFOA are wrong.

VIII.16 Claim 14

141. Claim 14 reads:

14. A method for encoding software code using a computer having a processor and memory, comprising:
storing a software code in said memory;
wherein said software code defines software code interrelationships between code resources that result in a specified underlying functionality when installed on a computer system;
and
encoding, by said computer using at least a first license key and an encoding algorithm, said software code,
to form a first license key encoded software code in which at least one of said software code interrelationships are encoded.

VIII.16 Contrary to the NFOA, Beetcher Does Not Anticipate Claim 14

142. Claim 14 recites "encoding, by said computer using at least a first license key and an encoding algorithm, said software code."

143. For the same reasons stated for claim 12, this recitation defines key based encoding; using the key as in input to the encoding function, to encode software code.

144. For the same reasons discussed for claim 12, Beetcher does not disclose this encoding, and therefore does not disclose the claimed "first license key encoded software code" encoding at least one "code interrelationship."

IV. Jurat

17. I have been warned that willful false statements and the like are punishable by fine or imprisonment, or both (18 U.S.C. 1001) and may jeopardize the validity of the application or any patent issuing thereon. All statements I make in the declaration I either know to be true or on information and belief I believe them to be true.

Signed: 
SCOTT MOSKOWITZ

Y:\Clients\SCOT Scott A Moskowitz and Wistaria Trading, Inc\90014138, USP9104842, SCOT0014-4\Drafts\Attachment17_132MoskowitzDeclaration.wpd

Agreement

Scott Moskowitz agrees to disclose certain information concerning pending patent ("Digital Information Commodities Exchange," filing #083-593, June 30, 1993) owned by Scott Moskowitz.

Marc Cooperman, upon receiving information from whatever source regarding said pending patent, agrees not to disclose or cause to be disclosed any information regarding said pending patent or affect said patent.

Nor shall Marc Cooperman permit any of his/her employees, associates, family members or others to disclose any information concerning said pending patent or cause to be disclosed pending patent in any manner.

Marc Cooperman



Signature

11/20/95
Date

Scott Moskowitz



Signature

11/12/92
Date

CERTIFICATE OF REGISTRATION



This Certificate issued under the seal of the Copyright Office in accordance with title 17, United States Code, attests that registration has been made for the work identified below. The information on this certificate has been made a part of the Copyright Office records.

HORT FORM TX
 or a Nondramatic Literary Work
 UNITED STATES COPYRIGHT OFFICE

TXu 892-516

Margbeth Peters

Effective Date of Registration
Feb 08 1999

REGISTER OF COPYRIGHTS
 United States of America

Amended by G.O. Authority of Scott Moskowitz in telephone call on June 1999.
 TYPE OR PRINT IN BLACK INK. DO NOT WRITE ABOVE THIS LINE.

Examined By: *AD*
 Correspondence:
 Application Received: **FEB 08 1999**
 Deposit Received: **ONE FEB 08 1999** Ten
 Fee Received

Title of This Work:	1	<i>Giovanni Master (audio digital watermark source code)</i>
Alternative title or title of larger work in which this work was published:		
Name and Address of Author and Owner of the Copyright:	2	<i>Scott Moskowitz 16711 Collins Avenue #2505 Miami Florida 33160 Phone (305) 956 9041 Fax () Email: scott@bluespike.com</i>
Nationality or domicile. Phone, fax, and email:		
Year of Creation:	3	<i>1998, 1999</i>
If work has been published, Date and Nation of Publication:	4	a. Date _____ (Month, day, and year all required) b. Nation _____
Type of Authorship in This Work:	5	<input checked="" type="checkbox"/> Text (includes fiction, nonfiction, poetry, computer programs, etc.) <input type="checkbox"/> Illustrations <input type="checkbox"/> Photographs <input checked="" type="checkbox"/> Compilation of terms or data
Check all that this author created:		
Signature: Registration cannot be completed without a signature.	6	I certify that the statements made by me in this application are correct to the best of my knowledge. Check one: <input type="checkbox"/> Author <input type="checkbox"/> Authorized agent <input checked="" type="checkbox"/> <i>Scott Moskowitz</i>
Name and Address of Person to Contact for Rights and Permissions: Phone, fax, and email:	7	<input checked="" type="checkbox"/> Check here if same as #2 above Phone () Fax () Email

8 Certificate will be mailed in window envelope to this address:

Name	<i>Scott Moskowitz</i>
Number Street Apt	<i>16711 Collins Avenue #2505</i>
City/State/Zip	<i>Miami Florida 33160</i>

9 Deposit Account # _____
 Name _____

17 U.S.C. § 405: Any person who knowingly makes a false representation of a material fact in the application for copyright registration provided for by section 402, or in any written statement filed in connection with the application, shall be fined not more than \$5,000.
 U.S. COPYRIGHT OFFICE WWW MARCH 1998

415-473-6811

ATTN: TIM STEVENS

HAPPY B'DAY

FR: S. Moskowitz

November 18, 1996

Dear Scott,

In the past several days I have been trying to communicate with you. Unfortunately, each time seems to end in a shouting match. So this is the only way I know of getting through to you in a clear manner, without being told to shut up long before I get to the point. I want you to do your best to read this through and consider what I am saying with a clear head, and do not jump to conclusions that I am an asshole. Please read it all. It will be long and wandering, but I want you to know everything in my head, because that is the only way to clear this up, however that happens. I know you have a meeting with Jot today, so this can wait until afterward.

I want you to know right here, at the top, before I raise any other issues, that I want this company to move forward, and I want to be part of it. I did not spend two years of my life with the intention of wasting it, and I personally don't want to let you down, because you trusted me. I am not stupid. I can see the threshold of what we are standing on. I am not selfish. I know other people are involved, but that doesn't mean I should fuck myself. I want you to know I do think about other people at the start of this, because the rest of it is simply what is in my head, and so therefore focuses on me and you.

Perhaps my choice of wordage regarding the equity financing clause and investments was a mistake. It has obviously upset you, and you must realize that this is about the last thing I want to do. Think about it. I imagine I am the most selfish person in the world. What could I possibly gain by doing that? I don't want to be in a pushing contest with you. Hindsight is 20/20. Do you think I want to fuck everything up when we are on the verge of success?

I used the term "wordage" because it is precisely that. I took no action whatsoever, and I want that to be clear. Despite what you may think, I have said only good things about you and this company to any one I have spoken to. I think I have tried on certain occasions to tell you personally how good a management job you are doing, because I had questioned it in the past. I did feel there were certain risks that they had to know about, or I am not meeting an obligation to them. There were certain people on my side of the financing who simply didn't belong there, and that was my mistake, and they pulled out because they were not close enough to me to simply take my word with no other limitations. I said what I said to Tim because he needed to know at the time, what I was thinking, and I did not want to conceal anything from him. Do you think I wanted to raise the issue with him? Of course not, but I felt it would be unholistic and unprofessional not to say it at that time.

Now, as a result of what I SAID to Tim directly and straightforwardly with the intent of reaching you (without waking you up as soon as possible, I believe you think I am playing games with you and/or trying to hold you hostage). My intent in communicating to you my feelings on the F&E financing and the equity financing clause was to make clear to you that I wanted the issue raised several weeks ago with you taken care of, one way or the other, before we move forward. I know there were delays, and I did not feel it was such a big deal, considering we had both agreed it was not necessary, which you now tell me has changed, in the interest of buying time, to simply reduce my side of the investment to make it a non issue for the moment. I am not impatent. I know you are busy, but if you will listen to my reasons below, you will see that I felt it could not remain silent longer. My hope was that we could settle what was on the table between us so it would not require me to do any such thing. We have had numerous arguments about the same issue in the past, and each time you put me off with some variation of "there is no point in talking about this now". So, based on past experience, I felt I had to make it very clear using the only means

Attachment 24 Page 1 of 1

Date: Wed Nov 15, 1995 4:57 am EST
From: Marc Cooperman
EMS: INTERNET / MCIID: 376-5414
MBX: coopman@netcom.com

TO: * Wistaria / MCIID: 554-8103
Subject: Spy vs. Font

Scott,

Regarding your note of 11-11-95
(ascii/software protection based on steganographic font metrics):

Looking at all this in the context of what you are saying:

Your idea seems to be

- 1) hide essential pieces of the app with an Argent-like scheme
- 2) make the "key/map" to access these resources randomized/individualized on a per copy basis
- 3) maybe have the correct key/map vary from run-to-run or iteration-to-iteration, as you seem to imply when talking about font metrics

---BEGIN EXCERPT---

Goal is to tie as much of the functionality of the software into the writing of the "written" code as possible. Afterall, the writing relates in some manner to the actual execution of concepts embodied in the code.

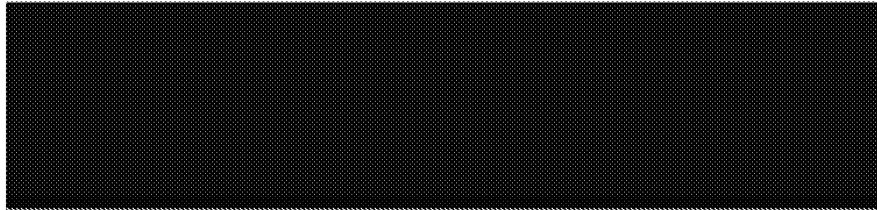
Should include both macro and micro approaches. The flaw is the copying of machine level code (the 0s and 1s that comprise the actual code). I think that tying actual processes into the randomized font can get around this. That is, for the missing puzzle pieces of the code a randomization process occurs when installed that identifies the machine and fills in the appropriate pieces to allow for missing functional pieces to all the whole to work. This could be encrypted also-- but I think, in my monkey brain, that both treating the body of code as an approximated picture, meaning each delivered copy is slightly different because of the randomized delivery of different fonts for each letter, and the functionality being tied to different pieces of the picture, as it were, is also random. So it is not just picture differences but the actual first time the code is "delivered" to the hard drive, its font comes out dissimilarly each time. The user really does not have to concern himself with IDs!!! at worst case.... The software manufacturer, however, can rest assured that copies will not work.

---END EXCERPT

Marc Cooperman

"There's a very fine line between clever... and stupid."
- famous fictional rock musician

Attachment 25 Page 1 of 1



The code and data resources which comprise the application are compiled and assembled into a collection which is a pre-variant of an executable application.

At this point, a utility application is used for final assembly of the executable application. The utility will choose one or several essential code resources, and encode these into one or several data resources using the assembler process. The end result will be that these essential code resources are not stored in their own partition, but rather stored as encoded information in data resources.

For the encoding, a key is needed. The purpose of this scheme is to make a licensed copy of an application distinguishable from any other. It is not necessary to distinguish every instance of an application, merely every instance of a license. A licensed user may wish to install multiple copies of an application, legally.

One method then, is to choose the key so that it corresponds, is equal to, or is a function of a license code.

The assembly utility can be supplied with a key generated from a license code generated for the license in question. Given the key, it encodes one or several essential resources into one or several data resources. Exactly which code resources are encoded into which data resources may be determined in various ways.

Note that the application contains a code resource which performs the function of decoding an encoded code resource from a data resource.

The application may also contain a data resource which specifies in which data resource a particular code resource is encoded.

This data resource is created and edited at assembly time by the assembly utility.

The application can then operate as follows. When it is run for the first time, after installation, it asks the user for personalization information, which includes the license code. It stores this information in a personalization data resource.

Since it has the license code, it can then generate the proper decoding key to access the protected code resources.

Note that the application can be copied to an installation medium, but MUST contain the license code found on the licensed system. In order to access its essential code resources.

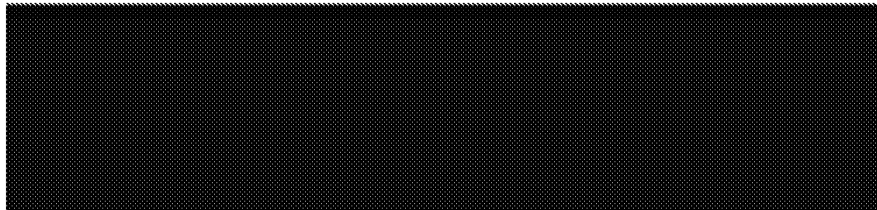
The object of the invention is thus accomplished.

Irregularly navigable program memory structure to prevent attempts at memory capture or object code analysis.

The object of this invention is to make it extremely difficult to perform memory capture based analysis of an executable computer program.

Since the code resources of a program are loaded into memory, they typically remain at a fixed position within the computer operating system. Thus it is necessary to rearrange certain portions of memory during "runtime". Typically, this is done in low memory systems, to maximize optimal memory utilization. If a computer program contains countermeasures against willful copying, a skilled technician can often take a snapshot of the code in memory, analyze it, determine which instructions comprise the countermeasures, and duplicate them in the saved application file, by means of a "patch".

Under this scheme, the application contains a special code resource which knows about all the other code resources in memory. During execution time, the special code resource, called a "memory scheduler", can be called periodically, or at random intervals, at which time it intentionally shuffles the other code resources randomly in memory, so that someone trying to analyze fragments of memory at random intervals



Security scheme for executable computer programs

Background & Intro

An executable computer program is variously referred to as an application, from the point of a user, or executable object code from the point of the engineer, a collection of modules, atoms (or indivisible chunks) of object code typically comprise a complete executable or application. These indivisible portions of object code correspond with the programmers' functions or procedures implementations in higher level languages, such as C, or Pascal. In creating an application, a programmer writes "code" in a higher level language, which is then compiled down into "machine language", or, the executable object code, which can actually be run by the computer. Each function, or procedure, written in the programming language, represents a self contained portion of the larger program, which implement, typically, a very small part of its functionality. The order in which the programmer types the code for the various functions or procedures, and the distribution of and arrangement of these implementations in various files which hold them is important. Within a function or procedure, however, the order of individual language constructs, which correspond to particular machine instructions is important, and so functions or procedures are considered indivisible for purposes of this discussion. That is, once a function or procedure is compiled, the order of the machine instructions which comprise the executable object code of the function is important and their order in the computer memory is of vital importance. Note that many compilers perform "optimizations" within functions or procedures, which determine, on a limited scale, if there is a better arrangement for executable instructions which is more efficient than that constructed by the programmer, but does not change the result of the function or procedure. Once these optimizations are performed, however, making random changes to the order of instructions is very likely to "break" the function. When a program is compiled, then, it consists of a collection of these self-objects, whose exact order or arrangement in memory is not important, so long as any sub-object which uses another sub-object knows where in memory it can be found.

The memory address of the first instruction in one of these sub-objects is called the "entry point" of the function or procedure. The rest of the instructions comprising that sub-object immediately follow from the entry point.

These sub-objects can be packaged into what are referred to in certain systems as "code resources", which may be stored separately from the application, or shared with other applications, although not necessarily.

Within an application there are also data objects, which consist of user data to be operated on by the executable code. These data objects are not executable. That is, they do not consist of executable instructions. The data objects can be referred to in certain systems as "resources".

Including necessary performance in digitized sample resources using single-chip process

The basic idea in this scheme is that there are a certain sub-set of executable code resources which comprise an application which are essential to the proper function of the application. In general, any code resource can be considered "essential" in that if the program proceeds to a point where it must "call" the code resource, and the code resource is not present in memory, or cannot be loaded, then the program fails. However, we use a definition of "essential" which is more narrow. A program may be written to work around unavailable resources. Particularly programs which incorporate an optional "plug-in architecture", where several code resources may be made optionally available at run-time. We are also concerned with concentrated efforts by technically skilled people who can analyze executable object code and "patch" it to ignore or bypass certain code resources. For our purposes, "essential" means that the function which distinguishes that application from any other application depends upon the presence and use of the code resource in question. The best candidates for this type of code resources are BIOS optional, or plug-in types.

Given that there are one or more of these essential resources, we then need the presence of certain data resources of a type which are accessible to the single-chip process. Data which consists of image or audio samples is particularly useful.



cannot be sure if they are looking at the same code or organization from one "break" to the next. This adds significant complexity to their job.

It is also necessary, to complete the effectiveness of the scheme to provide a second special code resource, which knows where the memory scheduler is in memory. This is a "scheduler envelope" It calls the scheduler, and, when the scheduler is finished, it randomly moves the scheduler, since the scheduler cannot move itself, a very hairy operation.

An alternative method is to increase the functionality of the scheduler so that it can move itself. In order to do this, the scheduler would have to first copy itself to a new location, and then specifically modify the program counter and stack frame, so that it could then jump into the new copy of the scheduler, but return to the correct calling frame.

The method described above accomplishes the purpose of the invention to make it hard to analyze captured memory containing application executable code.

12-22-05
Scott Moskowitz

A



METHOD FOR STEGA-CIPHER PROTECTION OF COMPUTER CODE

FIELD OF INVENTION

With the advent of computer networks and digital multimedia, protection of intellectual property has become a prime concern for creators and publishers of digitized copies of copyrightable works, such as musical recordings, movies, video games, and computer software. One method of protecting copyrights in the digital domain is to use "digital watermarks." Digital watermarks can be used to mark each individual copy of a digitized work with information identifying the title, copyright holder, and even the licensed owner of a particular copy. When marked with licensing and ownership information, responsibility is created for individual copies where before there was none. Computer application programs can be watermarked by watermarking digital content resources contained within the program, such as images or audio. Digital watermarks can be encoded with random or pseudo random keys, which act as secret maps for locating the watermarks. These keys make it impossible for a party without the key to find the watermark - in addition, the encoding method can be enhanced to force a party to cause damage to a watermarked data stream when trying to erase a random-key watermark. For more information on digital watermarks see

(state specific references, not application serial numbers).

"Steganographic Method and Device" - The DICE Company.

patent application

"Technology: Digital Commerce", Denise Caruso, New York Times.

August 7, 1995

"Copyrighting in the Information Age", Harley Ungar,

ONLINE MARKETPLACE, September 1995, Jupiter Communications

For more information on other methods for hiding information signals in content signals, see

U.S. Patent No. 5,319,735 - Preuss et al.

U.S. Patent No. 5,190,345 - Greenberg

It is desirable to use a "stega-cipher" or watermarking process to hide the necessary parts or resources of the executable object code in the digitized sample resources. It is also desirable to further modify the underlying structure of an executable computer application such that it is more resistant to attempts at patching and analysis by memory capture. Being that a computer application seeks to provide a user with certain utilities or tools, that is, users interact with a computer or similar device to accomplish various tasks and applications provide the relevant interface, a level of authentication can also be introduced into software, or "digital products," that include digital content, such as audio, video, pictures or multimedia, with digital watermarks. Security is maximized because erasing this code watermark without a key results in the destruction of one or more essential parts of the underlying application, rendering the "program" useless to the unintended user who lacks the appropriate key. Further, if the key is linked to a license code by means of a mathematical function, a mechanism for identifying the licensed owner of an application is created.

It is also desirable to randomly reorganize program memory structure intermittently during program run time, in order to prevent attempts at memory capture or object code analysis aimed at eliminating licensing or ownership information, or otherwise modifying, in an unintended manner, the functioning of the application. In this way, attempts to capture memory to determine underlying functionality or provide a "patch" to facilitate unauthorized use of the "application," or computer program, can be made difficult or impossible without destroying the functionality and thus usefulness of a copyrightable computer program.

It is thus the goal of the present invention to provide a higher level of copyright security to object code on par with methods described in digital watermarking systems for digitized media content such as pictures, audio, video and multimedia content in its multifarious forms, as described in previous disclosures, "Steganographic Method and Device" and "Human Assisted Random Key Generation and Application for Digital Watermark System." It is a further goal of the present invention to establish methods of copyright protection that can be combined with such schemes as software metering, network

Attachment 27 Page 1 of 8

distribution of code and specialized protection of software that is designed to work over a network, such as that proposed by Sun Microsystems in their HotJava browser and Java programming language, and manipulation of application code in proposed distribution of documents that can be exchanged with resources or the look and feel of the document being preserved over a network, such systems are currently being offered by companies including Adobe, with their Acrobat software. The latter goal being accomplished primarily by means of the watermarking of font, or typeface, resources included in applications or documents, which determine how a bitmap representation of the document is ultimately drawn on a presentation device.

SUMMARY OF THE INVENTION

The present invention includes an application of the technology of "digital watermarks." As described in previous disclosures, "Steganographic Method and Device" and "Human Assisted Random Key Generation and Application for Digital Watermark System," watermarks are particularly suitable to the identification, metering, distributing and authenticating digitized content such as pictures, audio, video and derivatives thereof under the description of "multimedia content." With methods described for combining both cryptographic methods, and steganography, or hiding something in plain view. Discussions of these technologies can be found in Applied Cryptography by Bruce Schneier and The Code Breakers by David Kahn. For more information on prior art public-key cryptosystems see U.S. Pat. No. 4,200,770 Diffie-Hellman, 4,218,582 Hellman, 4,405,829 RSA, 4,424,414 Hellman Pohlrig. Computer code, or machine language instructions, which are not digitized and have zero tolerance for error, must be protected by derivative or alternative methods, such as those disclosed in this invention, which focuses on watermarking with "keys" derived from license codes or other ownership identification information, and using the watermarks encoded with such keys to hide an essential sub set of the application code resources.

It is thus a goal of the present invention, to provide a level of security for executable code on similar grounds as that which can be provided for digitized samples. The prior art includes copy protection systems attempted at many stages in the development of the software industry, these may be various methods by which a software engineer can write the software in a clever manner to determine if it has been copied, and if so to deactivate itself. Also included are undocumented changes to the storage format of the content. Copy protection was generally abandoned by the software industry, since pirates were generally just as clever as the software engineers and figured out ways to modify their software and deactivate the protection. The cost of developing such protection was not justified considering the level of piracy which occurred despite the copy protection. Other methods for protection of computer software include the requirement of entering certain numbers or facts that may be included in a packaged software's manual, when prompted at start-up. These may be overcome if copies of the manual are distributed to unintended users, or by patching the code to bypass these measures. Other methods include requiring a user to contact the software vendor and disclosing "keys" for unlocking software after registration attached to some payment scheme, such as credit card authorization. Further methods include network-based searches of a user's hard drive and comparisons between what is registered to that user and what is actually installed on the user's general computing device. Other proposals, by such parties as Bell Labs, use "kerning" or actual distance in pixels, in the rendering of text documents, rather than a varied number of ASCII. However, this approach can often be defeated graphics processing analogous to sound processing, which randomizes that information. All of these methods require outside determination and verification of the validity of the software license. The present invention differs from the prior art in that it does not attempt to stop copying, but rather, to determine responsibility for a copy by ensuring that licensing information must be preserved in descendant copies from an original. Without the correct license information, the copy cannot function.

An improvement over the art is disclosed in the present invention, in that the software itself is a set of commands, compiled by software engineer, which can be configured in such a manner as to tie underlying functionality to the license or authorization of the copy in possession by the user. Without such verification, the functions sought out by the user in the form of software cease to properly work. Attempts to tamper or "patch" substitute code resources can be made highly difficult by randomizing the location of said resources in memory on an intermittent basis to resist most attacks at disabling the system.

BRIEF DESCRIPTION OF THE DRAWINGS

Attachment 27 Page 2 of 8

DETAILED DESCRIPTION

An executable computer program is variously referred to as an application, from the point of a user, or executable object code from the point of the engineer. A collection of smaller, atomic (or indivisible) chunks of object code typically comprise the complete executable object code or application which may also require the presence of certain data resources. These indivisible portions of object code correspond with the programmer's function or procedure implementations in higher level languages, such as C or Pascal. In creating an application, a programmer writes "code" in a higher level language, which is then compiled down into "machine language," or, the executable object code, which can actually be run by a computer, general purpose or otherwise. Each function, or procedure, written in the programming language, represents a self-contained portion of the larger program, and implements, typically, a very small piece of its functionality. The order in which the programmer types the code for the various functions or procedures, and the distribution of and arrangement of these implementations in various files which hold them is unimportant. Within a function or procedure, however, the order of individual language constructs, which correspond to particular machine instructions is important, and so functions or procedures are considered indivisible for purposes of this discussion. That is, once a function or procedure is compiled, the order of the machine instructions which comprise the executable object code of the function is important and their order in the computer memory is of vital importance. Note that many "compilers" perform "optimizations" within functions or procedures, which determine, on a limited scale, if there is a better arrangement for executable instructions which is more efficient than that constructed by the programmer, but does not change the result of the function or procedure. Once these optimizations are performed, however, making random changes to the order of instructions is very likely to "break" the function. When a program is compiled, then, it consists of a collection of these sub-objects, whose exact order or arrangement in memory is not important, so long as any sub-object which uses another sub-object knows where in memory it can be found.

The memory address of the first instruction in one of these sub-objects is called the "entry point" of the function or procedure. The rest of the instructions comprising that sub-object immediately follow from the entry point. Some systems may prefix information to the entry point which describes calling and return conventions for the code which follows, an example is the Apple Macintosh Operating System (MacOS). These sub-objects can be packaged into what are referred to in certain systems as "code resources," which may be stored separately from the application, or shared with other applications, although not necessarily. Within an application there are also data objects, which consist of some data to be operated on by the executable code. These data objects are not executable. That is, they do not consist of executable instructions. The data objects can be referred to in certain systems as "resources."

It is a goal, in seeking to purchase or acquire a computer program, by a user that a computer program "function" in a some desired manner. Simply, computer software is overwhelmingly purchased for its underlying functionality. In contrast, persons who copy multimedia content, such as pictures, audio and video, do so for the entertainment or commercial value of the content. The difference between the two types of products is that multimedia content is not generally interactive, but passive, and its commercial value relies more on passive not interactive or utility features, such as that required in packaged software, set-top boxes, cellular phones, VCRs, PDAs, and the like. Simply, interactive digital products which include computer code may be mostly interactive but can also contain content in add to the interactive experience of the user or make the underlying utility of the software more aesthetically pleasing. It is a common concern of both of these creators, both of interactive and passive multimedia products, that "digital products" can be easily and perfectly copied and made into unpaid or unauthorized copies. This concern is especially heightened when the underlying product is copyrighted and intended for commercial use.

The first method described in the present invention involves hiding necessary "parts" or "resources" in digitized sample resources using "digital watermarking" process, such as that described in the "Steganographic Method and Device" patent application. The basic premise for this scheme is that there are a certain sub-set of executable code resources, which comprise an application, that are "essential" to the proper function of the application. In general, any code resource can be considered "essential" in that if the program proceeds to a point where it must "call" the code resource, and the code resource is not present in memory, or cannot be loaded, then the program fails. However, the present invention uses a

definition of "essential" which is more narrow. This is because, those skilled in the art or those with programming experience, may create a derivative program, not unlike the utility provided by the original program, by writing additional or substituted code to work around unavailable resources. This is particularly true with programs that incorporate an optional "plug-in architecture," where several code resources may be made optionally available at run-time. The present invention is also concerned with concentrated efforts by technically skilled people who can analyze executable object code and "patch" it to ignore or bypass certain code resources. Thus, for the present embodiment's purposes, "essential" means that the function which distinguishes this application from any other application depends upon the presence and use of the code resource in question. The best candidates for this type of code resources are MDT optional, or plug-in types, unless special care is taken to prevent work-arounds.

Given that there are one or more of these essential resources, what is needed to realize the present invention is the presence of certain data resources of a type which are amenable to the "stega-cipher" process described in the "Steganographic Method and Device" patent application. Data which consists of image or audio samples is particularly useful. Because this data consists of digital samples, digital watermarks can be introduced into the samples. What is further meant is that certain applications include image and audio samples which are important to the look and feel of the program or are essential to the processing of the application's functionality when used by the user. These computer programs are familiar to users of computers but also less obvious to users of other devices that run applications that are equivalent in some measure of functionality to general purpose computers including, but not limited to, set-top boxes, cellular phones, "smart televisions," PDAs and the like. However, programs still comprise the underlying "operating systems" of these devices and are becoming more complex with increases in functionality.

One method of the present invention is now discussed. When code and data resources are compiled and assembled into a precursor of an executable program the next step is that a utility application is used for final assembly of the executable application. The utility will choose one or several essential code resources, and encode them into one or several data resources using the stega-cipher process. The end result will be that these essential code resources are not stored in their own partition, but rather stored as encoded information in data resources. They are not accessible at run-time without the key. Basically, the essential code resources that provide functionality in the final end-product, an executable application or computer program, are no longer easily and recognizably available for manipulation by those seeking to remove the underlying copyright or license, or its equivalent information, or those with skill to substitute alternative code resources to "force" the application program to run as an unauthorized copy. For the unloading of the essential code resources, a "key" is needed. Such a key is similar to those described in the "Steganographic Method and Device." The purpose of this scheme is to make a licensed copy of an application distinguishable from any other. It is not necessary to distinguish every instance of an application, merely every instance of a license. A licensed user may then wish to install multiple copies of an application, legally or with authorization. This method, then, is to choose the key so that it corresponds, is equal to, or is a function of, a license code, not just a text file, audio clip or identifying piece of information as desired in digital watermarking schemes extant and typically useful to stand-alone, digitally sampled content. The key is necessary to access the underlying code, what the user understands to be the application program.

The assembly utility can be supplied with a key generated from a license code generated for the licensee in question. Given the key, it encodes one or several essential resources into one or several data resources. Exactly which code resources are encoded into which data resources may be determined in a random or pseudo random manner. Note further that the application contains a code resource which performs the function of decoding an encoded code resource from a data resource. The application must also contain a data resource which specifies in which data resource a particular code resource is encoded. This data resource is created and added at assembly time by the assembly utility. The application can then operate as follows:

- 1) When it is run for the first time, after installation, it asks the user for personalization information, which includes the license code. This can include a particular computer configuration.
- 2) It stores this information in a personalization data resource.

Attachment 27 Page 4 of 8

If Once it has the license code, it can then generate the proper decoding key to access the essential code resources.

Note that the application can be copied in an uninhibited manner, but must contain the license code issued to the licensed owner, in order to access its essential code resources. The goal of the invention: copyright protection of computer code and establishment of responsibility for copies, is thus accomplished.

This invention represents a significant improvement over prior art because of the inherent difference in use of purely informational watermarks versus watermarks which contain executable object code. If the executable object code in a watermark is essential to an application which accesses the data which contains the watermark, this creates an all-or-none situation. Either the user must have the extracted watermark, or the application cannot be used, and hence the user cannot gain full access to the presentation of the information in the watermark bearing data. In order to extract a digital watermark, the user must have a key. The key, in turn, is a function of the license information for the copy of the software in question. The key is fixed prior to final assembly of the application files, and so cannot be changed at the option of the user. That, in turn, means the license information in the software copy must remain fixed, so that the correct key is available to the software. The key and the license information are, in fact, interchangeable. One is merely more readable than the other. In the prior art, "Steganographic Method and Device," the possibility of randomization/erasure attacks on digital watermarks was discussed. Simply, it is always possible to erase a digital watermark, depending on how much damage you are willing to do to the watermark-bearing content stream. The present invention has the significant advantage that you must have the watermark to be able to use the code it contains. If you erase the watermark you have lost a key piece of the functionality of the application, or even the means to access the data which bears the watermark.

A preferred embodiment would be implemented in an embedded system, with a minimal operating system and memory. No media playing "applets," or smaller sized applications as proposed in new operating environments envisioned by Sun Microsystems and the advent of Sun's Java operating system, would be permanently stored in the system, only the bare necessities to operate the device, download information, decode watermarks and execute the applets contained in them. When an applet is finished executing, it is erased from memory. Such a system would guarantee that content which did not contain readable watermarks could not be used. This is a powerful control mechanism for ensuring that content to be distributed through such a system contains valid watermarks. Thus, in such networks as the Internet or set-top box controlled cable systems, distribution and exchange of content would be made more secure from unauthorized copying to the benefit of copyright holders and other related parties. The system would be enabled to invalidate, by default, any content which has had its watermark(s) erased, since the watermark conveys, in addition to copyright information, the means to fully access, play, record or otherwise manipulate the content.

A second method for the present invention is to randomly re-organize program memory structure to prevent attempts at memory capture or object code analysis. The object of this method is to make it extremely difficult to perform memory capture-based analysis of an executable computer program. This analysis is the basis for a method of attack to defeat the system envisioned by the present invention.

Once the code resources of a program are loaded into memory, they typically remain in a fixed position, unless the computer operating system finds it necessary to rearrange certain portions of memory during "system time," when the operating system code, not application code, is running. Typically, this is done in low memory systems, to maintain optimal memory utilization. The MacOS for example, uses Handles, which are double-indirect pointers to memory locations, in order to allow the operating system to rearrange memory transparently, underneath a running program. If a computer program contains countermeasures against uncensored copying, a skilled technician can often take a snapshot of the code in memory, analyze it, determine which instructions comprise the countermeasures, and disable them in the stored application file, by means of a "patch." Other applications for designing code that moves to prevent scanning-tunnelling microscopes, and similar high sensitive hardware for analysis of electronic structure of microchips running code, have been proposed by such parties as Wave Systems. Designs of Wave Systems' microchip are intended for preventing attempts by hackers "photograph" or otherwise determine "burn in" to microchips for attempts at reverse engineering. The present invention seeks to prevent

attempts at patches that can be introduced to determine the code that comprises the application file. Unlike systems such as Wave Systems', the present invention seeks to move code around in such a manner as to complicate attempts by software engineers to reengineer a means to disable the methods for creating licensed copies on any device that lacks "trusted hardware." Moreover, the present invention concerns itself with any application software that may be used in general computing devices, not chipsets that are used in addition to an underlying computer to perform encryption. Wave Systems' approach to security of software if interpreted similarly to the present invention would dictate separate microchip sets for each piece of application software that would be tamperproof-- not consistent with the economics of software and its distribution.

Under the present invention, the application contains a special code resource which knows about all the other code resources in memory. During execution time, this special code resource, called a "memory scheduler," can be called periodically, or at random or pseudo random intervals, at which time it intentionally shuffles the other code resources randomly in memory, so that someone trying to analyze snapshots of memory at various intervals cannot be sure if they are looking at the same code or organization from one "break" to the next. This adds significant complexity to their job. The scheduler also randomly relocates itself when it is finished. In order to do this, the scheduler would have to first copy itself to a new location, and then specifically modify the program counter and stack frame, so that it could then jump into the new copy of the scheduler, but return to the correct calling frame. Finally, the scheduler would need to maintain a list of all memory addresses which contain the address of the scheduler, and change them to reflect its new location. The methods described above accomplish the purpose of the invention - to make it hard to analyze captured memory containing application executable code in order to create an identifiable computer program or application that is different from other copies and is less susceptible to unauthorized use by those attempting to disable the underlying copyright protection system. Simply, each copy has particular identifying information making that copy different from all other copies.

What is Claimed:

- 1) The method of associating executable object code with a digital sample stream by means of a digital watermark wherein the digital watermark contains the executable object code and is encoded into the digital sample stream
- 2) The method of claim 1 where the key to access the digital watermark is a function of a collection of license information pertaining to the software which is accessing the watermark
where license information consists of one or more of the following items:
 - Owning Organization name
 - Personal Owner name
 - Owner Address
 - License code
 - Software serialization number
 - Distribution parameters
 - Appropriate executable general computing device architecture
 - Pricing
 - Software Metering details
- 3) The method of claim 1 further comprised of the step of transmitting the digital sample stream, via a transmission means, from a publisher to a subscriber
where transmission means can be one of:
 - soft sector magnetic disk media
 - hard sector magnetic disk media
 - magnetic tape media
 - CD-ROM disc media
 - CD-R disc media
 - Digital Video Disc media
 - magneto-optical disc media
 - memory cartridge

telephone lines
SCSI
Ethernet or Token Ring Network
ISDN
ATM network
TCP/IP network
analog cellular network
digital cellular network
wireless network
digital satellite
cable network
fiber optic network
electric powerline network

4) The method of claim 1 where the object code to be encoded is comprised of series of executable machine instructions which perform the function of at least one of
(ADD APPLLET LANGUAGE HERE)
processing a digital sample stream for the purpose of modifying it
playing a digital sample stream

5) The method of claims 1 and 4 further comprised of the steps of
decoding said digital watermark and extracting object code
loading object code into computer memory for the purpose of execution
executing said object code in order to process said digital sample stream for the purpose of playback

6) The method of assembling an application to be protected by watermark encoding of essential resources comprised of the steps of

assembling a list of identifiers of essential code resources of an application where identifiers allow the code resource to be accessed and loaded into memory
providing license information on the licensee who is to receive an individualized copy of the application
storing license information in a personalization resource which is added to the list of application data resources
generating a digital watermark key from the license information
using the key as a pseudo-random number string to select a list of suitable digital sample data resources, the list of essential code resources, and a mapping of which essential code resources are to be watermarked into which data resources
storing the map, which is a list of paired code and data resource identifiers, as a data resource, which is added to the application
adding a digital watermark decoder code resource to the application, to provide a means for extracting essential code resource from data resources, according to the map
processing the map list and encoding essential code resources into digital sample data resources with a digital watermark encoder
removing self-contained copies of the essential code resources which have been watermarked into data resources
combining all remaining code and data resources into a single application installer

7) The method of intermittently relocating application code resources in computer memory, in order to prevent, discourage, or complicate attempts at memory capture based code analysis

8) The method of claim 7 additionally comprised of the steps of

assembling a list of identifiers of code resources of an application

Attachment 27 Page 7 of 8



where identifiers allow the code resource to be accessed and loaded into memory.

9) The method of claim 8 additionally comprised of the step of modifying application program structure to make all code resource calls indirectly, through the memory scheduler, which looks up code resources in a list and dispatches calls.

10) The method of claim 9 additionally comprised of the step of intermittently rescheduling or shuffling all code resources prior to or following the dispatch of a code resource call through the memory scheduler.

11) The method of claim 10 additionally comprised of the step of the memory scheduler copying itself to a new location in memory.

12) The method of claim 11 additionally comprised of the step of modifying the stack frame, program counter, and memory registers of the CPU to cause the scheduler to jump to the next instruction comprising the scheduler, in the copy, to erase the previous memory instance of the scheduler, changing all memory references to the scheduler to reflect its new location, and to return from the copy of the scheduler to the frame which called the previous copy of the scheduler.

ABSTRACT:

01-03-96

Scott Moskowitz

Attachment 27 Page 8 of 8



Electronic Acknowledgement Receipt

EFS ID:	35107072
Application Number:	90014138
International Application Number:	
Confirmation Number:	7638
Title of Invention:	DATA PROTECTION METHOD AND DEVICE
First Named Inventor/Applicant Name:	9104842
Customer Number:	31518
Filer:	Richard A. Neifeld
Filer Authorized By:	
Attorney Docket Number:	
Receipt Date:	11-FEB-2019
Filing Date:	16-MAY-2018
Time Stamp:	09:48:24
Application Type:	Reexam (Third Party)

Payment information:

Submitted with Payment	no
------------------------	----

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)/ Message Digest	Multi Part /.zip	Pages (if appl.)
1	Reexam Certificate of Service	CertificateOfService_90014138.pdf	53137 <small>c7651313e4c9c34e539ffa05d41e7c3c498b8023</small>	no	1

Warnings:

Information:					
2	Amendment/Req. Reconsideration-After Non-Final Reject	2019-01-28_ResponseToNFOA_90014138.pdf	211923 3ee6ee2f482a74fb6b0bef9c8cc3530286014b844	no	36
Warnings:					
Information:					
3	Affidavit-not covered under specific rule	Attachment11_GoogleSearch_Encoding_ComputerImage.pdf	735874 68f9c7115ca2a97487dfe9e45fd5dca78888f8fb	no	2
Warnings:					
Information:					
4	Affidavit-not covered under specific rule	Attachment12_Encoding_decoding_WhatsImage.pdf	686655 a7c5ed21c4ab3dbb8ce7aa51d05a40c5f73d010	no	7
Warnings:					
Information:					
5	Affidavit-not covered under specific rule	Attachment13_ObjectCodeImage.pdf	725538 283dc8026b65d0f520b7bd05d3de27cc864c5903	no	2
Warnings:					
Information:					
6	Affidavit-not covered under specific rule	Attachment14_RoutineImage.pdf	627148 39e88539cb4c7ed21950af0383a71107ea98ca40	no	3
Warnings:					
Information:					
7	Affidavit-not covered under specific rule	Attachment15_MicrocodeImage.pdf	1850175 8909caad206d9bdd57a9b8c56f3365f942db18ee	no	8
Warnings:					
Information:					
8	Affidavit-Rule 131-pre-AIA (FTI) ONLY	Attachment16_131MoskowitzDeclarationSigned.pdf	287397 2c82909cd9cf203e81f8a8b666cc29801849b9ea	no	15
Warnings:					
Information:					

9	Affidavit-traversing rejectns or objectns rule 132	Attachment17_132Moskowitz DeclarationSigned.pdf	608644 af50730d521c6a309b60c4b76ba8fa74e933a204	no	30
Warnings:					
Information:					
10	Affidavit-not covered under specific rule	Attachment23_SM_MC_1993Agreement.pdf	828037 5986d2a4178e2fba3342397a9e73958e02b24cd3f	no	2
Warnings:					
Information:					
11	Affidavit-not covered under specific rule	Attachment24_Page1_CoopermanLetter_11-18-1996Image.pdf	568450 64abb4cb8f30bed9b53d44d1bb6c606386b4ac67	no	1
Warnings:					
Information:					
12	Affidavit-not covered under specific rule	Attachment25_11-15-95EmailDisclosureImage.pdf	164230 3ea54be760aff1a746b33ffba208061a967be8430	no	1
Warnings:					
Information:					
13	Affidavit-not covered under specific rule	Attachment26_DraftApplicationDated12-22-95Image.pdf	341691 5203407373bcacf8e2d03a7b270ea29842a50ad9b	no	3
Warnings:					
Information:					
14	Affidavit-not covered under specific rule	Attachment27_DraftApplicationDated01-03-96Image.pdf	1703883 408f2cc14fb7f4360bca8f23405ead4b6f7bb2de	no	8
Warnings:					
Information:					
Total Files Size (in bytes):				9392782	

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

New Applications Under 35 U.S.C. 111

If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

National Stage of an International Application under 35 U.S.C. 371

If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

New International Application Filed with the USPTO as a Receiving Office

If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.

Reexamination Control Number: 90014138
Confirmation No: 7638
RE: US Patent 9104842

37 CFR 1.234 Certificate of Service

I served by first class mail (priority mail), the third party requestor at the reexamination requestor's correspondence address:

Attn: Joseph P. Edell
FISCH SIGLER LLP
5301 WISCONSIN AVENUE, NW
FOURTH FLOOR
WASHINGTON, DC 20015

with the following documents:

This Certificate of Service (1 page), showing service by first claims mail (express mail) of:

37 CFR 1.530 Patent Owner Response to the Non Final Office Action Dated 12/12/2028
Attachment 11, a copy of the screen image showing the Google search query and quote definitions.

Attachment 12, the definition of encoding and decoding, at <https://searchnetworking.techtarget.com/definition/encoding-and-decoding>.

Attachment 13, the definition of "Object Code" from the Technopeida website.

Attachment 14, the definition of "routine" from the Webopedia website.

Attachment 15, definition of "microcode" from Wikipedia.

Attachment 16, 37 CFR 1.131 declaration of Scott Moskowitz

Attachment 17, 37 CFR 1.132 declaration of Scott Moskowitz

Attachment 23, SM_MC_1993Agreement.pdf Agreement between Marc Cooperman and Scott Moskowitz

Attachment 24, Page1_CoopermanLetter_11-18-1996.pdf Letter from Marc Cooperman to Scott Moskowitz

Attachment 25, 11-15-95EmailDisclosure.pdf Email dated 11/15/1995 from Marc Cooperman to Scott Moskowitz

Attachment 26, DraftApplicationDated12-22-95.pdf Draft patent application dated 12/22/1995.

Attachment 27, DraftApplicationDated01-03-96.pdf Draft patent application dated 1/3/1996.

Date of Service: 2/11/2019

/RichardNeifeld/
Richard Neifeld, Reg. No. 35,299
Attorney for patent owner

STATEMENT UNDER 37 CFR 3.73(c)

Applicant/Patent Owner: WISTARIA TRADING LTD
 Application No./Patent No.: 9104842 Filed/Issue Date: August 11, 2015
 Titled: Data protection method and device
WISTARIA TRADING LTD, a Corporation
(Name of Assignee) (Type of Assignee, e.g., corporation, partnership, university, government agency, etc.)

states that, for the patent application/patent identified above, it is (choose **one** of options 1, 2, 3 or 4 below):

1. The assignee of the entire right, title, and interest.
2. An assignee of less than the entire right, title, and interest (check applicable box):
- The extent (by percentage) of its ownership interest is _____%. Additional Statement(s) by the owners holding the balance of the interest must be submitted to account for 100% of the ownership interest.
 - There are unspecified percentages of ownership. The other parties, including inventors, who together own the entire right, title and interest are:

Additional Statement(s) by the owner(s) holding the balance of the interest must be submitted to account for the entire right, title, and interest.

3. The assignee of an undivided interest in the entirety (a complete assignment from one of the joint inventors was made). The other parties, including inventors, who together own the entire right, title, and interest are:

Additional Statement(s) by the owner(s) holding the balance of the interest must be submitted to account for the entire right, title, and interest.

4. The recipient, via a court proceeding or the like (e.g., bankruptcy, probate), of an undivided interest in the entirety (a complete transfer of ownership interest was made). The certified document(s) showing the transfer is attached.

The interest identified in option 1, 2 or 3 above (not option 4) is evidenced by either (choose **one** of options A or B below):

- A. An assignment from the inventor(s) of the patent application/patent identified above. The assignment was recorded in the United States Patent and Trademark Office at Reel 036342, Frame 0663, or for which a copy thereof is attached.

- B. A chain of title from the inventor(s), of the patent application/patent identified above, to the current assignee as follows:

1. From: _____ To: _____

The document was recorded in the United States Patent and Trademark Office at Reel _____, Frame _____, or for which a copy thereof is attached.

2. From: _____ To: _____

The document was recorded in the United States Patent and Trademark Office at Reel _____, Frame _____, or for which a copy thereof is attached.

This collection of information is required by 37 CFR 3.73(b). The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.11 and 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

STATEMENT UNDER 37 CFR 3.73(c)

3. From: _____ To: _____

The document was recorded in the United States Patent and Trademark Office at
Reel _____, Frame _____, or for which a copy thereof is attached.

4. From: _____ To: _____

The document was recorded in the United States Patent and Trademark Office at
Reel _____, Frame _____, or for which a copy thereof is attached.

5. From: _____ To: _____

The document was recorded in the United States Patent and Trademark Office at
Reel _____, Frame _____, or for which a copy thereof is attached.

6. From: _____ To: _____

The document was recorded in the United States Patent and Trademark Office at
Reel _____, Frame _____, or for which a copy thereof is attached.

Additional documents in the chain of title are listed on a supplemental sheet(s).

As required by 37 CFR 3.73(c)(1)(i), the documentary evidence of the chain of title from the original owner to the assignee was, or concurrently is being, submitted for recordation pursuant to 37 CFR 3.11.

[NOTE: A separate copy (i.e., a true copy of the original assignment document(s)) must be submitted to Assignment Division in accordance with 37 CFR Part 3, to record the assignment in the records of the USPTO. See MPEP 302.08]

The undersigned (whose title is supplied below) is authorized to act on behalf of the assignee.

/RichardNeifeld/

2/11/2019

Signature

Date

RICHARD NEIFELD

35,299

Printed or Typed Name

Title or Registration Number

Reexamination Control Number: 90014138
Confirmation No: 7638
RE: US Patent 9104842

37 CFR 1.234 Certificate of Service

I served by first class mail (priority mail), the third party requestor at the reexamination requestor's correspondence address:

Attn: Joseph P. Edell
FISCH SIGLER LLP
5301 WISCONSIN AVENUE, NW
FOURTH FLOOR
WASHINGTON, DC 20015

with the following documents:

This Certificate of Service (1 page), showing service by first claims mail (express mail)
of:

37 CFR 1.324 Renewed Petition to Correct Inventorship in Issued US Patent 9104138
Statement by Marc Cooperman - Agreement (37 CFR 1.324(b)(1) statement)
PTO/AIA/96, 37 CFR 3.73(c) statement establishing that Wistaria Trading, Ltd. is the
assignee

Agreement of Assignee of Record and Original Inventor to Correction of Inventorship
(1.324(b)(1) and (b)(2) statement)

Date of Service: 2/11/2019

/RichardNeifeld/
Richard Neifeld, Reg. No. 35,299
Attorney for patent owner

Electronic Patent Application Fee Transmittal

Application Number:	90014138				
Filing Date:	16-May-2018				
Title of Invention:	DATA PROTECTION METHOD AND DEVICE				
First Named Inventor/Applicant Name:	9104842				
Filer:	Richard A. Neifeld				
Attorney Docket Number:					
Filed as Large Entity					
Filing Fees for ex parte reexam					
Description	Fee Code	Quantity	Amount	Sub-Total in USD(\$)	
Basic Filing:					
Pages:					
Claims:					
Miscellaneous-Filing:					
Petition:					
PROCESSING FEE CORRECTING INVENTORSHIP	1816	1	150	150	
Patent-Appeals-and-Interference:					
Post-Allowance-and-Post-Issuance:					

Description	Fee Code	Quantity	Amount	Sub-Total in USD(\$)
Extension-of-Time:				
Miscellaneous:				
Total in USD (\$)				150

Electronic Acknowledgement Receipt

EFS ID:	35110727
Application Number:	90014138
International Application Number:	
Confirmation Number:	7638
Title of Invention:	DATA PROTECTION METHOD AND DEVICE
First Named Inventor/Applicant Name:	9104842
Customer Number:	31518
Filer:	Richard A. Neifeld
Filer Authorized By:	
Attorney Docket Number:	
Receipt Date:	11-FEB-2019
Filing Date:	16-MAY-2018
Time Stamp:	13:02:54
Application Type:	Reexam (Third Party)

Payment information:

Submitted with Payment	yes
Payment Type	CARD
Payment was successfully received in RAM	\$ 150
RAM confirmation Number	021119INTEFSW13035800
Deposit Account	
Authorized User	

The Director of the USPTO is hereby authorized to charge indicated fees and credit any overpayment as follows:

File Listing:					
Document Number	Document Description	File Name	File Size(Bytes)/ Message Digest	Multi Part /.zip	Pages (if appl.)
1	Petition for review/processing depending on status	2019-02-11_Petition_Inventors hip_USP9104842.pdf	93613 be8fc7e09a7efacde71cf7a7c6ffd24a4e3ef405	no	5
Warnings:					
Information:					
2	Petition for review/processing depending on status	2019-02-11_CoopermanInventorhsipStatement_90104138.pdf	103565 08e1278fd308741b870ef456b039d0e747573828	no	1
Warnings:					
Information:					
3	Petition for review/processing depending on status	2019-02-11_373c_90014138.pdf	492400 61ff8c57b15ee871604e994524a4c9e4d97e0849	no	2
Warnings:					
Information:					
4	Petition for review by the Technology Center SPRE	CertificateOfService_90014138.pdf	43015 15113b2306cf660c08008b9e7e4a07ad6a4b323f	no	1
Warnings:					
Information:					
5	Petition for review/processing depending on status	2019-02-11_AgreementAssigneeOriginalInventorSigned_USP9104842.pdf	212157 9ed30ac7c6214df9463a0ec270dfd1f0323f3a59	no	1
Warnings:					
Information:					
6	Fee Worksheet (SB06)	fee-info.pdf	29749 d9c3b9ea0df677e12c77cac9d616626ca48f9c6	no	2
Warnings:					
Information:					

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

New Applications Under 35 U.S.C. 111

If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

National Stage of an International Application under 35 U.S.C. 371

If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

New International Application Filed with the USPTO as a Receiving Office

If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.

RE: US Patent 9104842

**Agreement of Assignee of Record and Original Inventor
to Correction of Inventorship**

I am the named inventor, Scott Moskowitz, of USP9104842. I agree to correcting inventorship of the patent by adding Marc Cooperman as a named inventor.

I am also authorized to act on behalf of the assignee of record, WISTARIA TRADING LTD, of USP9104842. The assignment is recorded at reel/frame: 036342/0953.

I agree on behalf of WISTARIA TRADING LTD to correcting inventorship of the patent by adding Marc Cooperman as a named inventor.


Scott Moskowitz,
Manager, WISTARIA TRADING LTD

\\Files\Y\Clients\SCOT Scott A Moskowitz and Wistaria Trading, Inc\90014138, USP9104842,
SCOT0014-4\Drafts\2019-02-11_AgreementAssigneeOriginalInventor.wpd

RE: US Patent 9104138
Reexamination: 90014138
Reexam conf code: 7638

37 CFR 1.324 Renewed Petition to Correct Inventorship in Issued US Patent 9104138

I. Relief Requested

Pursuant to 35 USC 355 and 37 CFR 1.323, MPEP 1481, and MPEP 1412.04, and MPEP 1481.02, the applicant petitions to correct the inventorship in this patent, as follows.

On the cover page, replace:

"(76) Inventor: Scott A. Moskowitz, Sunny Isles Beach, FL (US)"

with

"(76) Inventors: Scott A. Moskowitz, Sunny Isles Beach, FL (US); Marc Cooperman, Short Hills, NJ (US)"

That is, the patent owner petitions to add Mr. Cooperman as a named inventor.

II. Material Facts

1. The following attachments are submitted with this petition.

Statement by Marc Cooperman - Agreement (37 CFR 1.324(b)(1) statement)
PTO/AIA/96, 37 CFR 3.73(c) statement establishing that Wistaria Trading, Ltd. is the assignee

Agreement of Assignee of Record and Original Inventor to Correction of Inventorship (1.324(b)(1) and (b)(2) statement)

Moreover, a \$150 fee is being submitted electronically upon filing this petition.

2. Pursuant to MPEP 1412.04: (A) the only change being made in the patent is to correct the inventorship; and (B) all parties are in agreement and the inventorship issue is not contested. MPEP 1481.02 identifies "all parties" to be the originally named inventors and assignees, noting "Correction of inventorship in a patent under 37 CFR 1.324 requires petition of all the parties, *i.e., originally named inventors and assignees.*"

3. The assignee of record is "WISTARIA TRADING LTD" as shown by the abstract of title indexed at reel/frame: 036342/0953. The named inventor is Scott Moskowitz, as shown on the face of the patent. Scott Moskowitz is authorized to act on behalf of WISTARIA TRADING LTD.

4. On 2/11/2019, the patent owner filed a response to the Non Final Office Action (NFOA) which stated on pages 31 and 312, in relevant part:

XI.3 Regarding Mr. Marc Cooperman

The NFOA, at page 91:5-15 states that:

Patent Owner argues that "If Mr. Cooperman refuses to cooperate, then that fact would also indicate he is not an inventor of the subject matter defined by claims 11-14 in USP9104842."

In response, the Examiner respectfully submits that Mr. Cooperman has been cooperating by the Patent Owner's own admission, as noted on page 1 of the Patent Owner Statement: "Mr. Cooperman recently testified in the related district court litigation and the transcript of his testimony suggests that he was not an inventor of at least some of the subject matter defined by claims in the subject patent, USP9104842." It is just that the testimony of Mr. Cooperman didn't serve the purpose of attaching him to the subject Patent. Additionally someone's refusal to cooperate does not alone indicate inventorship or preclude inventorship.

With due respect, the examiner's conclusion that the fact that Mr. Cooperman recently testified in a court proceeding is an admission that Mr. Cooperman is cooperating with the patent owner is a non sequitur. Exhibit 3 clearly shows that Mr. Cooperman's testimony was taken by "defendant" Juniper Networks, Inc. Exhibit 3, page 1. And that page also shows that the deposition was pursuant to a "Notice", presumably an subpoena. Mr. Cooperman's testimony was not at the request of, and not for the benefit of patent owner, and is not an admission that Mr. Cooperman "has been cooperating" with patent owner.

It is true that the long ago settlement agreement in Attachment 7, Section 2, obligated Mr. Cooperman to sign certain assignment documents and otherwise assigned all rights relating to IP in that dispute to the patent owner. However, the undersigned has repeatedly emailed and telephoned Mr. Cooperman, over a dozen times between the summer of 2018 and the present, requesting that he determine if he believes himself to be an inventor of any claim in USP9104842, and notify the patent owner of that determination. In fact, the undersigned emailed short alternative statements to Mr. Cooperman, for his signature and return. One form stating he was an inventor and one form stating he was not an inventor. To date, Mr. Cooperman has not returned a signed form. (If Mr. Cooperman does notify the undersigned that he believes himself to be an inventor of any claim in SP9104842 during the pendency of this reexam, the undersigned will at a minimum notify the examiner of that fact.)

Mr. Cooperman long ago surrendered any rights he might have in USP9104842 to the patent owner, pursuant to the settlement agreement in

attachment 7. Therefore, whether or not Mr. Cooperman should be named as inventor is not relevant to ownership.

Given his lack of response, to date, on this issue, to say Mr. Cooperman "has been cooperating by the Patent Owner" is not correct.

5. The undersigned finalized the response to the NFOA on 2-10-2019, uploaded that response to the EFS server and saved it using the "Saved submission" function in EFS, including the certificate of service, and generated a service package (printed and prepared the service documents and obtained and printed a USPS Express mailing certificate and prepare the express mailing envelope).

6. The undersigned deposited the service package with the USPS local branch and completed the EFS filing, first thing in the morning of 2-11-2019 and then submitted the saved submission.

7. The undersigned's email client received an email from Mr. Cooperman dated "Sun 2/10/2019 10:03 PM" which stated:

Rick,
So I've reviewed the patent in question.
Given that it clearly derives from patent
5,745,569 4/28/98 Method for stega-cipher protection of computer code
by way of the two additional continuation patents of that patent referenced in the
doc (ea. with 1 claim)
and the original patent, which I was deeply involved with, includes specifically
- copy protection aspects
- personalization/licensing aspects
- given: fungibility of data and code
- the memory shifting anti-reverse engineering measures anticipated
and given the claims of the patent in question
my conclusion is I believe I contributed to the conception of the claims of the
patent in question
I will be forwarding the completed doc momentarily
Yours truly,
Marc

8. The undersigned's email client received an email from Mr. Cooperman dated "Sun 2/10/2019 10:10 PM" which stated:

Rick,
attached is the executed agreement
I sign "Marc S. Cooperman"
if there is any issue with that please let me know and I'll redo

9. A copy of what Mr. Cooperman referred to as the "executed agreement," is a statement signed by Mr. Cooperman. That statement reads:

Reexamination Control Number: 90/014,138

Reexamination of US patent 9104842

Statement by Marc Cooperman - Agreement

I have reviewed the claims of US patent 9104842.

Upon review, I believe I contributed to the conception of at least one claim of US patent 9104842.

I state that I have no disagreement with the change to patent 9104842 adding my name as a named inventor to that patent.

10. The undersigned prepared this petition and associated documents promptly upon review of Mr. Cooperman's emails, beginning during the morning of 2-11-2019.

11. A copy of that Signed "Statement by Marc Cooperman - Agreement", is attached to this petition. That document is a 37 CFR 1.324(b)(1) statement signed by Mr. Cooperman.

12. The USPTO's Assignment Recordation database, Reel/Frame "036342/0953," shows that USP 9104842 is assigned by Mr. Moskowitz to Wistaria Trading Ltd.

13. A 37 CFR 3.73(c) statement establishing that Wistaria Trading, Ltd. is the assignee and referring to Reel/Frame "036342/0953" is attached to this petition.

14. An "Agreement of Assignee of Record and Original Inventor to Correction of Inventorship," signed by Scott Moskowitz, showing that the assignee and original inventor agree to the change, is attached to this petition. This document is a 37 CFR 1.324(b)(2) statement signed by Mr. Moskowitz on behalf of the assignee, Wistaria Trading Ltd., and a 37 CFR 1.324(b)(1) statement signed by Mr. Moskowitz, as a named inventor.

15. Attachment 7 to the patent owner response contains a settlement agreement dated 2002. Attachment 7, sections 2.1 and 2.4, and Exhibit 1, therein, shows that Mr. Cooperman assigned his entire right, title, and interest in inventions **disclosed in**, inter alia, in USP5745569 and WO/9726732, to Wistaria Trading Ltd. and its successors and assigns.

16. Attachment 16 to the response to the NFOA is Mr. Moskowitz's 131 declaration. Section III.C of that declaration shows that all claims 11-14 of USP 9104842 are disclosed by application 08/587,943, as filed. This application issued as USP5745569.

17. 37 CFR 1.324(c) requires payment of the fee specified in 1.20(b). That fee is currently \$150. See fee code 1816, on <https://www.uspto.gov/learning-and-resources/fees-and-payment/uspto-fee-schedule>. That fee is being paid electronically upon filing of this petition.

III. Reasons the Petition Should be Granted

The patent owner has complied with all regulatory requirements in 37 CFR 1.324 required for grant of the petition, specifically:

Pursuant to 1.324(a), all parties and assignees apply for this correction. Mr. Moskowitz on behalf of the assignee and as a named inventor, and Mr. Cooperman agree.

Pursuant to 1.324(b)(1), Mr. Cooperman does not object to being added as a named inventor. This is shown by his statement submitted herewith.

Pursuant to 1.324(b)(1), Mr. Moskowitz either agrees or does not object to the change, as shown by Mr. Moskowitz's signed statement submitted herewith.

Pursuant to 1.324(b)(2), Wistaria Trading Ltd. either agrees or does not object to the change, as shown by Mr. Moskowitz's signed statement submitted herewith.

This petition also complies with the requirements of 37 CFR 3.73(c) to show the assignee's entitlement to take action. This is shown by the form PTO/AIA/96, which is a 3.73(c) statement also submitted herewith.

The subject patent is a pre-AIA patent. For pre-AIA patents, 3.73(c) requires the assignee establish ownership of the patent. For pre-AIA patents one alternative for complying with 3.73(c) is a 3.73(c)(ii) statement signed pursuant to 3.73(2). 3.73(c)(ii) requires: "A statement specifying where documentary evidence of a chain of title from the original owner to the assignee is recorded in the assignment records of the Office (e.g., reel and frame number)." 3.73(2) requires a "The submission establishing ownership must show that the person signing the submission is a person authorized to act on behalf of the assignee." The form PTO/AIA/96 submitted herewith is a 3.73(c) statement complying with these requirements.

Truly,
/RichardNeifeld/
RICHARD NEIFELD, REG. NO. 35,299
ATTORNEY OF RECORD

Typed date and time of completion of the original draft of this petition: 2-11-2019, at 11:42 AM.
Actual print time generated by WordPerfect code: February 11, 2019 (11:52am)
\\Files1\Y\Clients\SCOT Scott A Moskowitz and Wistaria Trading, Inc\90014138, USP9104842, SCOT0014-4\Drafts\2019-02-11_Petition_Inventorship_USP9104842.wpd

Reexamination Control Number: 90/014,138
Reexamination of US patent 9104842

Statement by Marc Cooperman - Agreement

I have reviewed the claims of US patent 9104842.

Upon review, I believe I contributed to the conception of at least one claim of US patent 9104842.

I state that I have no disagreement with the change to patent 9104842 adding my name as a named inventor to that patent.

Signed:



Marc Cooperman

MARC S. COOPERMAN

Y:\Clients\SCOT Scott A Moskowitz and Wistaria Trading, Inc\90014138, USP9104842,
SCOT0014-4\Drafts\MC_9104842Agreement.wpd



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
90/014,138	05/16/2018	9104842		7638

31518 7590 02/12/2019
NEIFELD IP LAW, PC
5400 Shawnee Road
Suite 310
ALEXANDRIA, VA 22312-2300

EXAMINER BONSHOCK, DENNIS G

ART UNIT 3992	PAPER NUMBER
------------------	--------------

MAIL DATE 02/12/2019	DELIVERY MODE PAPER
-------------------------	------------------------

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.



UNITED STATES PATENT AND TRADEMARK OFFICE

Commissioner for Patents
United States Patents and Trademark Office
P.O.Box 1450
Alexandria, VA 22313-1450
www.uspto.gov

THIRD PARTY REQUESTER'S CORRESPONDENCE ADDRESS

Date: February 12, 2019

FISCH SIGLER, LLP
5301 WISCONSIN AVENUE, NW
FOURTH FLOOR
WASHINGTON, DC 20015

EX PARTE REEXAMINATION COMMUNICATION TRANSMITTAL FORM

REEXAMINATION CONTROL NO. : 90014138
PATENT NO. : 9104842
ART UNIT : 3992

Enclosed is a copy of the latest communication from the United States Patent and Trademark Office in the above identified ex parte reexamination proceeding (37 CFR 1.550(f)).

Where this copy is supplied after the reply by requester, 37 CFR 1.535, or the time for filing a reply has passed, no submission on behalf of the ex parte reexamination requester will be acknowledged or considered (37 CFR 1.550(g)).



UNITED STATES PATENT AND TRADEMARK OFFICE

Commissioner for Patents
United States Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450
www.uspto.gov

NEIFELD IP LAW, PC :
5400 Shawnee Road :
Suite 310 : Patent Owner
Alexandria, VA 22312-2300 :

FISCH SIGLER, LLP :
5301 WISCONSIN AVENUE, NW :
FOURTH FLOOR : Third Party Requester
WASHINGTON, DC 20015 :

In re Application of: Scott Moskowitz : DECISION ON PETITION
Appl. No. 90/014,138 : FOR CORRECTION
Patent No. 9,104,842 : OF PATENT UNDER
Filed: May 16, 2018 : 37 C.F.R. § 1.324(b)
For: DATA PROTECTION METHOD :
AND DEVICE :

This is a decision on a petition under 37 C.F.R. § 1.324 filed February 11, 2019 to correct the inventorship of U.S. Patent No. 9,104,842 (the '842 patent) to add Marc Cooperman as inventor.

The petition is **Granted**.

37 C.F.R. §1.530(l)(1) provides:

When it appears in a patent being reexamined that the correct inventor or inventors were not named through error without deceptive intention on the part of the actual inventor or inventors, the Director may, on petition of all the parties set forth in §1.324(b)(1)-(3), including the assignees, and satisfactory proof of the facts and payment of the fee set forth in § 1.20(b), or on order of a court before which such matter is called in question, include in the reexamination certificate to be issued under § 1.570 or § 1.997 an amendment naming only the actual inventor or inventors. The petition must be submitted as part of the reexamination proceeding and must satisfy the requirements of § 1.324.

A petition to correct inventorship as provided by 37 C.F.R. § 1.324 requires (1) a statement from each person who is being added as an inventor that the inventorship error occurred without any deceptive intention on their part, (2) a statement from the current named inventors (including any "inventor" being deleted) who have not submitted a statement as per "(1)" either agreeing to the change of inventorship or stating that they have no disagreement in regard to the requested change, (3) a statement from all assignees of the parties submitting a statement under "(1)" and "(2)" agreeing to the change of inventorship in the patent; such statement must comply with the requirements of 37 CFR 3.73(b); and (4) the fee set forth in 37 CFR 1.20(b).

This petition complies with all requirements of 37 C.F.R. § 1.324 and 37 C.F.R. § 1.530(l)(1).

37 C.F.R. § 1.324(b)(1) requires a statement from each person who is being added as an inventor.

Regarding this requirement, patent owner submitted a signed statement from Marc Cooperman that he has no disagreement with the change of adding his name as an inventor of the '842 patent.

37 C.F.R. § 1.324(b)(2) requires a statement from the current named inventors either agreeing to the change of inventorship or stating that they have no disagreement in regard to the requested change.

Patent Owner has submitted statements under 37 C.F.R. § 1.324(b)(2) by Scott Moskowitz that provide an affirmative statement he agrees with the change to the inventorship.

37 C.F.R. § 1.324(b)(3) requires a statement from all assignees of the parties submitting a statement under paragraphs (b)(1) and (b)(2) of this section agreeing to the change of inventorship in the patent, which statement must comply with the requirements of 37 C.F.R. § 3.73(b).

Patent Owner has submitted a statement signed by Scott Moskowitz, who is duly authorized to act on behalf of the assignee Wistaria Trading Ltd. In this statement, the assignee affirmatively agreed to the change of inventorship.

37 C.F.R. § 1.324(b)(3) requires the fee set forth in 37 C.F.R. § 1.20(b).

Patent Owner has submitted the proper fee as set forth in 37 C.F.R. § 1.20(b).

Conclusion

Patent Owner has complied with all formal and procedural requirements of 37 C.F.R. § 1.324 and 37 C.F.R. § 1.530(l)(1).

Accordingly, Patent Owner's petition for a Correction of Inventorship of US 9,104,842 is **Granted.**

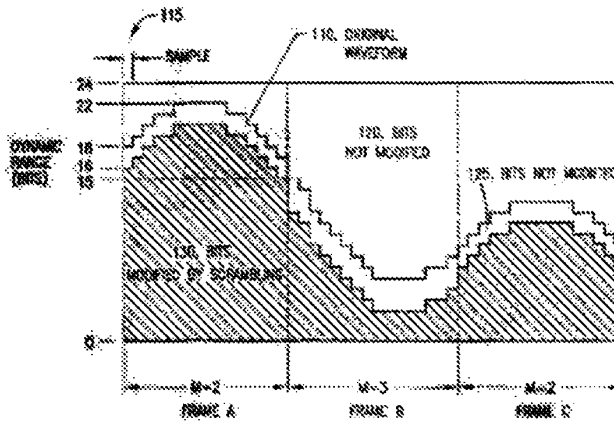
/Stephen Stein/
Quality Assurance Specialist
Central Reexamination Unit
(571) 272-3744



INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification ⁶ ; H04N 7/167		AI	(11) International Publication Number: WO 99/55089
(21) International Application Number: PCT/US99/08039		(43) International Publication Date: 28 October 1999 (28.10.99)	
(22) International Filing Date: 20 April 1999 (20.04.99)		(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GR, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LU, LV, MD, MG, MK, MN, MW, MX, MY, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TH, TM, TR, TT, UA, UG, US, UZ, VN, YU, ZW. ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TD, 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, LI, LU, MC, NL, PT, SE), OAPI patent (BF, BI, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).	
(30) Priority Data: 09/682,488 21 April 1998 (21.04.98) US		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.	
(71) Applicant [for all designated States except US]: SORANA TECHNOLOGY DEVELOPMENT CORPORATION [US/US]: Suite 500, 6256 Greenwich Drive, San Diego, CA 92122 (US).			
(72) Inventors; and (73) Inventors/Applicants [for US only]: LAM, S., Katherine [US/US]: 9888 Camino Cielo, San Diego, CA 92131 (US); MOJALIMBI, Kamno [US/US]: 4314 Vista De La Tierra, Del Mar, CA 92014-4104 (US); LEE, Chong U. [US/US]: 11710 Alderidge Lane, San Diego, CA 92131 (US); KATOH, Take [JP/JP]: 3-11-20, Jyomoyji, Kamakura, Kanagawa 248-0003 (JP); EDODOH, Naoki [JP/JP]: 2-10-56, Nishihara-cho, Fuchu-shi, Tokyo 183-0046 (JP).			
(74) Agent: LIPSITZ, Barry, S.; 755 Main Street, Monroe, CT 06468 (U.S.).			

(54) Title: MULTIMEDIA ADAPTIVE SCRAMBLING SYSTEM (CLASS)



(57) Abstract

A system (300, 500) for scrambling signal samples (115, 200, 250, 260, 270) of multimedia data, including audio and video data samples, such that the content of the samples is degraded but still recognizable, or otherwise provided at a desired quality level. The samples may be in any recognizable compressed or uncompressed digital format, including Pulse Code Modulation (PCM) samples, samples in floating point representation, samples in companding schemes (e.g., μ -law and A-law), and other compressed or streams. The quality level may be associated with a particular signal or noise ratio, or quality level that is determined by objective and/or subjective tests, for example. A number of LSSs can be scrambled in successive samples in successive frames (FRAME A, FRAME B, FRAME C). Moreover, the parameters for scrambling may change from frame to frame. Furthermore, all or part of the scrambling key (210) can be embedded (340) in the scrambled data and recovered at a decoder (400, 600) to be used in descrambling. After descrambling, the scramble key is no longer recoverable because the scramble key itself is scrambled by the descrambler.

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	GB	Great Britain	LV	Latvia	SS	Switzerland
AX	Årland	GD	Grenada	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TZ	Tanzania
BE	Belgium	GN	Guinea	MR	The Islamic Republic of Mauritania	TM	Turkmenistan
BF	Burkina Faso	GR	Greece	MT	Republic of Malta	TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MH	Marshall Islands	UG	Uganda
BV	Bolivia	IS	Iceland	MI	Maldives	US	United States of America
CA	Canada	IT	Italy	MR	Morocco	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VE	Venezuela
CG	Congo	KE	Kenya	NI	Nicaragua	VI	Virgin Islands
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	VU	Vanuatu
CI	Côte d'Ivoire	KR	Democratic People's Republic of Korea	NZ	New Zealand	ZW	Zimbabwe
CM	Cameroun	KW	Republic of Korea				
CN	China	KZ	Kazakhstan				
CO	Colombia	LC	Saint Lucia				
CR	Costa Rica	LI	Liechtenstein				
DE	Germany	LR	Liberia				
DK	Denmark	LS	Lesotho				
EE	Estonia	LT	Lithuania				

MULTIMEDIA ADAPTIVE SCRAMBLING SYSTEM (MASS)

BACKGROUND OF THE INVENTION

This application claims the benefit of U.S. Provisional Application No. 60/982,488, filed April 21, 1998.

The present invention relates to a method and apparatus for scrambling digital samples of multimedia data, e.g., such as audio or video data samples, such that the content of the samples is degraded but still recognizable. For example, the invention is suitable for use with digital broadcast streams and digital storage media, such as compact discs (CDs) and digital video discs (DVDs). The number of least significant bits (LSBs) scrambled in each sample is selected such that the scrambled samples are degraded but still recognizable.

Schemes for communicating and storing digital data have become increasingly popular, particularly in the mass consumer market for digital audio, video, and other data. Consumers may now send, receive, store, and manipulate digital television, audio and other data content, such as computer games and other software, stock ticker data, weather data and the like. This trend is expected to continue with the integration of telephone, television and computer network resources.

However, in many cases it is desirable to control or monitor the use of such digital data. In particular, copyright holders and other proprietary

interests have the right to control the distribution and use of their works, including audio, video and literary works.

5 In a copyright management system where audio and video content are to be protected, it would be desirable to provide data scrambling to deter theft of the content while it is in transit. The distance of transit can be half way around the world, as with delivery on the internet, or millimeters, such as 10 within a DVD player's internal data transfer from disc to DAC (Digital to Analog Converter).

15 It would be desirable to provide a method and apparatus that renders the audio/video content unsuitable for listening/viewing purpose but sufficient for identification of an audio/video passage, e.g., during fast forward playback, when there is insufficient time to descramble the samples.

20 It would be desirable to scramble bits in successive frames of digital data samples according to a scrambling key, where the scrambling key is embedded into the scrambled signal.

25 It would be desirable for the scrambling key to be associated with the scrambled data to allow the scrambling key to be easily changed without modifying the player (e.g., DVD or CD player) on which the data is played.

30 It would be desirable to scramble data such that the content is degraded sufficiently so that it no longer has any significant commercial value, but,

at the same time, is perceptually satisfactory for player functions such as cueing and fast forward.

The scrambled data should not damage the video or audio equipment even if it is played through any video or audio playback system. Some randomly
5 scrambled waveforms can result in such damage, e.g., to speakers or circuitry.

It would further be desirable for the scrambled data to be any conceivable digital data.

10 It would be desirable for the data to be scrambled at any time, including, for example, when the data signal is created (e.g., during a recording session for an audio track), when the data signal is being distributed (e.g., during a broadcast, or
15 during manufacture of storage media such as compact discs), or when the data is being played (e.g., on a player in a consumer's home).

The present invention provides a system having the above and other advantages.

SUMMARY OF THE INVENTION

A method for protecting digital samples of content from illicit use by scrambling the content is provided, wherein each sample includes a plurality of bits, ranging from least significant bits (LSBs) to most significant bits. The method includes the step of scrambling a number of LSBs in each sample according to a scrambling key, while preserving a number of MSBs in each sample, to provide corresponding scrambled samples. The number of LSBs scrambled in each sample is provided such that the scrambled samples are degraded but still recognizable.

The number of LSBs to be scrambled in each sample may be adaptively determined according to the dynamic range of the sample.

Alternatively, or in addition, the number of LSBs to be scrambled in each sample may be adaptively determined according to the particular frame a sample is in, where different frames can have a different number of LSBs scrambled.

The scrambling key may be a pseudo-random scrambling key, for example.

The LSBs may be scrambled within the same sample using intra-sample scrambling.

Alternatively, or in addition, the LSBs may be scrambled between different samples using inter-sample scrambling.

A particular type of inter-sample scrambling is horizontal inter-sample scrambling, where bits

having the same weight are interchanged between samples.

The scrambling key may be embedded into the scrambled samples for use at a decoder in
5 descrambling the scrambled samples.

In particular, the scrambling key for a current frame of scrambled samples may be embedded into a previous frame of samples. This avoids a delay in recovering the key at the decoder.

10 A corresponding descrambling method, and corresponding apparatuses are also presented.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 illustrates adaptive bit modification for successive data samples in successive data frames in accordance with the present invention.

5 FIG. 2(a) illustrates intra-sample scrambling in accordance with the present invention.

FIG. 2(b) illustrates original, unscrambled samples.

10 FIG. 2(c) illustrates the samples of FIG. 2(b) after horizontal, inter-sample scrambling in accordance with the present invention.

FIG. 3 illustrates a data scrambler and scramble key encoder in accordance with the present invention.

15 FIG. 4 illustrates a data descrambler and scramble key decoder in accordance with the present invention.

FIG. 5 illustrates a detailed data scrambler in accordance with the present invention.

20 FIG. 6 illustrates a detailed data descrambler in accordance with the present invention.

25 FIG. 7 illustrates adaptive bit modification with a fixed minimum dynamic range for successive data samples in successive data frames in accordance with the present invention.

DETAILED DESCRIPTION OF THE INVENTION

The present invention relates to a method and apparatus for scrambling digital samples of multimedia data.

5 This subject matter discussed in the following patents and patent applications, each of which is incorporated herein by reference, may be adapted for use with the present invention: U.S. Patent 5,822,360, entitled "Method and Apparatus for
10 Transporting Auxiliary Data in Audio Signals"; application no. 08/764,096, filed December 6, 1996, entitled "Method and Apparatus for Embedding Auxiliary Data in a Primary Data Signal"; U.S. Patent 5,687,191, entitled "Post Compression Hidden
15 Data Transport"; application no. 08/912,434, filed August 18, 1997, entitled "Post Compression Hidden Data Transport for Video"; U.S. Patent 5,719,937, entitled "Multi-Media Copy Management System"; application no. 08/977,719, filed November 25, 1997,
20 entitled "Multi-Media Copy Management System", and application no. _____, filed _____, entitled "Digital Hidden Data Transport". Similar systems for providing embedded information may also be adapted for use with the present invention.

25 Audio, video or other digital content is scrambled to degrade its quality, but still allow a listener, viewer, or other user to recognize the content.

Selective scrambling of a content waveform
30 consists of preservation of some of the MSBs (up to

and including the sign bit) and scrambling of some or all of the remaining LSBs. The scrambling noise is maintained at a desired level compared to the original data (e.g., audio, video or other data).

5 Each data sample's M most significant bits are dynamically preserved. The number M can be chosen before processing begins as a preset parameter, or can vary during processing.

10 Dynamic preservation of MSBs involves performing amplitude range detection on each sample and preserving (or masking) M of the most significant bits within the range of the sample, as shown in Figure 1. M may vary from 0 to the full dynamic range of the digitized samples.

15 M is the number of bits preserved in each sample's dynamic range and can be varied (for example, frame to frame) during processing. Moreover, as discussed further below, M can be part of the information embedded in the content material
20 for the decoder to use.

The digital content samples may be in any conceivable compressed or uncompressed digital format, including Pulse Code Modulation (PCM) samples, samples in floating point representation,
25 samples in companding schemes (e.g., μ -law and A-law), and other compressed bit streams as described below.

Most of the prevailing compression systems, for audio or video or images, tend to use frequency
30 domain techniques to reduce the perceptually

redundant information from the signal being compressed. Therefore, a majority of the compressed bit streams can be parsed and decoded into a set of parameters that include a set of frequency samples or transform coefficients. Examples are subband samples in MPEG audio coding, TDAC transform coefficients in AC-3 or AAC audio coding, and DCT coefficients in JPEG and MPEG image and video coding. These representations can be generalized as "frequency samples".

In a generalized compression scheme, a group of uncompressed signal samples are represented by a group of frequency samples, which are quantized according to the perceptual criteria for efficient storage and transmission. The ratio between the number of frequency samples required to represent the number of signal samples are usually fixed, e.g., at one. However, the number of frequency samples actually selected for transmission may be less, since some of the perceptually unimportant frequency samples are often not transmitted. This can be determined by extracting the bit allocation information contained in the compressed bit stream.

The frequency samples that are not transmitted usually have zero bits allocated, or otherwise indicated as zeros (e.g., a run length is indicated for a series of zeros). In essence, from the compressed bit stream, a binary representation of the frequency samples can be extracted with an augmentation information that indicates the number of bits allocated, dynamic range of the frequency

sample, or the scale factor needed to restore the frequency sample to its full value.

Once the binary representation of the frequency samples are extracted, the process of partial scrambling is no different than the process that applies to the uncompressed domain signal samples, e.g., as with a PCM representation.

FIG. 1 illustrates adaptive bit modification for successive data samples in successive data frames in accordance with the present invention.

Samples of digital data are selectively scrambled such that some of the Most Significant Bits (MSBs) are preserved, and some or all of the remaining data bits (e.g., LSBs) are scrambled. In addition, the key to unscramble the data can be concealed into the scrambled signal (e.g., using the techniques discussed in the aforementioned patents and applications) so that the key is no longer decodable once the content data is unscrambled.

In FIG. 1, the horizontal axis designates successive frames and samples of digital data. In this example, three frames are shown (FRAME A, FRAME B and FRAME C), each frame has fifteen samples, and each sample has twenty-four bits, although the present invention can accommodate any variation of these parameters. Each sample has a dynamic range between zero to twenty-four bits which is defined by the most significant non-zero bit, not including a sign bit, which may be the leftmost bit in each sample.

An original waveform 110 designates the dynamic range of each sample prior to scrambling in accordance with the present invention. Regions 120 and 125 designate the bits in each sample which are not modified, while a region 130 designates the bits in each sample which may be scrambled in accordance with the present invention.

For example, the first sample 115 of FRAME A has a dynamic range of 18 bits. An example of such a sample might be:

0₀,0₁,0₂,0₃,0₄,0₅,0₆,1₇,0₈,0₉,1₁₀,1₁₁,0₁₂,0₁₃,0₁₄,1₁₅,0₁₆,0₁₇.

where the subscripts denotes the bit position, ranging from bit0 (0₀) for the LSB to bit17 (1₁₇) for the MSB.

As shown by the region 125 which extends across FRAMES A, B and C, the number of MSBs which are not modified (e.g., preserved) for each frame may vary with each frame. For example, for FRAME A, M-2 MSBs are not modified, for FRAME B, M-3 MSBs are not modified, and for FRAME C, M-2 MSBs are not modified. The number of bits that are not modified in each frame may therefore be adaptive. In particular, the number of bits which are not modified may be inversely proportional to the maximum dynamic range of the frame. For example, for FRAME A, the maximum dynamic range is 22 bits, and M=2, while for FRAME B, the maximum dynamic range is only 15 bits, and M=1.

It is also possible to set M to a constant for each frame for simplicity.

Generally, the specific number of bits which are not modified can be selected by experimentation to yield data with a desired level of degradation.

5 The bits in the region 130 are subject to scrambling as discussed in greater detail below.

10 Perceptually, the selectively scrambled samples appear noisy. For example, audio samples will sound noisy to the listener, e.g., when played on a player in the user's home. Generally, the samples will sound noisier for samples where fewer MSEs are preserved, although the perception of the noise can vary based on the dynamic range of the each sample as well as the neighboring samples, the type of audio being played, the listening environment and other factors.

15 The noise is generally not objectionable for player functions such as cueing and fast forward playback. During fast forward playback (or fast reverse, if provided), there is insufficient time for the player to descramble the samples. During normal playback, the player has sufficient time to descramble the samples, so the content is heard or seen with its full dynamic range.

20 In accordance with the present invention, instead of scrambling the content over its entire dynamic range, a portion of the range is preserved so that the content is degraded but still recognizable, e.g., during fast forward playback. This allows the user to conveniently fast forward through audio or video content to locate a precise segment of interest.

Furthermore, the selectively scrambled content material limits the amplitude of the scrambled waveform such that it does not exceed the dynamic range of the original waveform. Thus, unlike a randomly scrambled waveform, the scrambled content does not damage the video or audio equipment even if it is played through any video or audio playback system.

Scrambling of the data bits can be done within each sample (intra-sample scrambling) or within a group of samples (inter-sample scrambling). Furthermore, a combination of intra-sample and inter-sample scrambling can be used.

Scrambling may take the form of interchanging the position of the data bits (inter-sample scrambling), masking the data bits with a sequence derived from the scramble key or parts of a scramble key (intra-sample scrambling), or combinations thereof. The purpose is to randomize or whiten the statistics of the scrambled data bits to make it "look" more random. The descrambler must undo this mapping of the bits.

Random or pseudo-random scrambling improves the security of the scrambled data since it will be more difficult for an attacker to detect patterns in the scrambled data.

FIG. 2(a) illustrates intra-sample scrambling in accordance with the present invention. An original sample 200 includes bits $S_1, S_1, S_1, S_1, S_1, S_1, S_1, S_1,$ where "S1" denotes "sample 1". For simplicity, the sample is shown having only

eight bits, although any number of bits may be used for each sample. Additionally, assume $M=3$, which means the three MSBs (i.e., S_1, S_1, S_1) are not modified, but the remaining bits (i.e.,

5 S_1, S_1, S_1, S_1, S_1) are. S_1 is assumed to be the MSB of the sample. After intra-sampling scrambling, the modified sample 250 may be obtained, which includes bits $S_1, S_1, S_1, S_1, S_1, S_1, S_1, S_1$. The modified bits may be scrambled using any known scrambling technique.

10 FIG. 2(b) illustrates original, unscrambled samples. Like-numbered elements correspond to one another in the figures. Sample 1 (200), Sample 2 (250), and Sample 3 (270) are shown. FIG. 2(c) illustrates the samples of FIG. 2(b) after

15 horizontal, inter-sample scrambling in accordance with the present invention. Samples 1 (200'), 2 (250') and 3 (270') correspond to Samples 1 (200), 2 (250) and 3 (270), respectively.

For inter-sample scrambling, the waveform

20 samples are grouped into frames. Each frame is associated with a scramble key, such as a pseudo-random key. The scramble key can be different for each frame and is determined at encoding time. A particularly useful case of inter-sample scrambling

25 is horizontal scrambling, where the bits with the same weight are interchanged among the samples.

For example, bit S_1 replaces bit S_3 , bit S_1 replaces bit S_3 , bit S_1 replaces bit S_2 , and so forth.

Note that it is also possible to use non-horizontal scrambling, wherein the weight of the scrambled bits is not maintained.

FIG. 3 illustrates a data scrambler and scramble key encoder in accordance with the present invention. In one possible implementation, the scramble key is embedded into a scrambled audio content.

A major advantage of using partially scrambled samples is that all or part of a key to descramble the samples can be concealed within the scrambled samples themselves. This is important because the scramble key is protected before and after descrambling.

The scrambler/encoder 100 includes a scramble key generator 110 for generating a scramble key.

The scramble key does not have to be the same length as the sample length. A longer scramble key length makes the data more secure.

The scramble key is provided to a key buffer 120, which stores the key, e.g., for one frame. In this manner, the key for scrambling a current frame is carried in the previous frame. After a one (or more) frame delay, the scramble key is provided to a scrambler for scrambling the samples in the original waveform, e.g., as discussed in connection with FIGS 2(a)-(c). Generally, the same scramble key may be used to scramble a number of samples in a frame.

The scrambled waveform is provided to a scramble key encoder 140 to encode the scramble key into the waveform, thereby providing the scrambled

waveform with the embedded scramble key. For example, the scramble key may be encoded as auxiliary data in the waveform using the techniques discussed in the aforementioned U.S. patents and applications.

The scramble key is concealed by the noise-like scrambled content.

FIG. 4 illustrates a data descrambler and scramble key decoder in accordance with the present invention. A descrambler/decoder 400 includes a scramble key decoder 410 that receives the scrambled waveform, e.g., from the scrambler/encoder 300 of FIG. 3. The scrambled waveform may be communicated via any communication channel, and/or recovered from a storage medium, for example.

The scramble key decoder 410 recovers the scramble key that was used to scramble the following frame and stores it in a key buffer 420. The scramble key decoder 410 operates using the corresponding technique used by the scramble key encoder 340.

The scramble key is retrieved from the buffer 420 for use by a descrambler 430 in descrambling the current frame of samples to provide the descrambled waveform. The descrambled waveform may undergo subsequent processing, such as digital-to-analog conversion, e.g., for viewing or listening by a user.

Advantageously, after descrambling, the scramble key is no longer recoverable even by an attacker who has the scramble key decoder 410

because the scramble key is scrambled by the descrambler 439.

FIG. 5 illustrates a detailed data scrambler in accordance with the present invention. In the scrambler 500, successive samples, each having W bits (sample width), are provided to sample frame buffers 510 and 510'. Non-scrambled buffered samples are output from the sample frame buffer 510 according to a linear address generator 515 for scrambling at a scrambler 330 according to a scrambling key.

Buffered samples in a pseudo-random order within a frame are output from the sample frame buffer 510' according to addresses from a scramble address generator 525, which receives the scramble key. The buffered samples are bit-wise ANDed at an AND function 530 with a mask bit string for the LSBs (LSB_MASK) to produce LSBs that are interchanged with other samples within the current frame.

LSB_MASK is a bit string with 1's corresponding to the LSBs to be scrambled in the buffered samples, and 0's for the other bits in the buffered samples.

A range detector and mask generator function 535 provides LSB_MASK, along with a mask bit string for the MSBs which are to be preserved (i.e., not scrambled), MSB_MASK. MSB_MASK is a bit string with 1's corresponding to the MSBs to be preserved in the buffered samples, and 0's for the other bits in the buffered samples.

MSB_MASK is bit-wise ANDed with the buffered samples from the sample frame buffer 510 at an AND function 540 to produce the preserved MSBs.

5 Scrambled samples from the scrambler 330 are bit wise ANDed at an AND function 545 with MSB_MASK/ (the inverse of MSB_MASK) and LSB_MASK/ (the inverse of LSB_MASK). MSB_MASK/ and LSB_MASK/ are obtained from the inverters 550 and 555, respectively. The output of the AND function 545 comprising the
10 scrambled middle bits is provided to a bit-wise OR function 560 for combining with the preserved MSBs and the LSBs that are interchanged with other samples in a frame.

15 The output of the OR function 560 comprises the scrambled samples, which may be provided to the scramble key encoder 340 to provide scrambled samples with the embedded scrambling key. The samples are now ready, e.g., for transmission across a network, or storage on a storage medium.

20 The scrambled key may be embedded in the scrambled samples by providing a scramble key encoder (such as element 340 in FIG. 3). Note that the embedding of the scrambling key is optional.

25 As shown in FIG. 5, one method to selectively scramble the audio (e.g., so that it is sufficient for identification of music passage but lacks commercial value for music enjoyment) is to preserve the sign bit and, e.g., two most significant bits of each sample. Amplitude range detection is performed
30 on each audio sample. A mask is generated for each sample corresponding to the amplitude range of that

particular sample. Examples of samples quantized to 16-bits in 2's complement format and corresponding masks are shown below:

Example Sample Value in Hexadecimal	Example Sample Value in Binary	Mask for sign bit + two MSBs in Hexadecimal (MSB_MASK)	Mask for sign bit + two MSBs in Binary (MSB_MASK)
0x0cH	0000101100111111	0x0c00	1111100000100000
0xff8H	1111111100110000	0xff80	1111111111100000

Each sample has sixteen bits, where the leftmost bit is a sign bit. For example, for the sample value, 0xff8H in hexadecimal, or 1111111100110000 in binary, bit15 ('1') indicates that this is a negative number, i.e., negative 0x61, or -97 in decimal. Thus, the dynamic range is seven bits (bit0 to bit6).

In one example implementation of the scrambler 500, the remaining bits are separated into two groups - the least significant bit, and all other remaining bits. The least significant bit is scrambled horizontally by exchanging positions within a frame. Inter-frame scrambling may also be used. The other remaining bits are scrambled by XORing with a bit pattern which is generated from all or part of the scramble key. A linear shift register can be used to vary the bit pattern for each sample.

The scrambled output consists of 1) the preserved MSBs, 2) the intra-sample scrambled middle bits, and 3) the horizontal inter-sample scrambled LSBs.

5 FIG. 6 illustrates a detailed data descrambler 600 in accordance with the present invention. The descrambler may be used for processing data received from the scrambler 500, for example.

10 Elements 610, 610', 615, 630, 635, 640', 645', 650', 655' and 660' correspond to elements 610, 610', 615, 630, 635, 640, 645, 650, 655 and 660, respectively in FIG. 5.

15 As shown in FIG. 6, the scramble key is decoded from the scrambled samples at the scramble key decoder 410, and provided to a buffer 420, and then to the descrambler 430. A linear address generator 515' is used, if necessary, for providing an address to a sample frame buffer 610 to restore the bit pattern for each received sample.

20 The received, scrambled samples are provided from the sample frame buffer 610 to the range detect/mask generator function 535'. The preserved MSBs in each sample are identified the same way as at the scrambler, by using amplitude range 25 detection. The descrambled middle bits in each sample are combined at the OR function 560' with the preserved MSBs and the reordered LSBs.

30 The inverse MSB mask (MSB_MASK/) is provided at the output of an inverter 550' to an AND function 545' for ANDing with the descrambled samples. The output of the AND function 545' is provided to the

OR function 560' for ORing with the preserved MSBs and the reordered LSBs within a frame.

The reordered LSBs within a frame are derived from ORing LSE_MASK with the reordered buffered samples, which are derived from the sample frame buffer 510'' in response to addresses from a descramble address generator 625.

At the output of the AND function 530', the LSB's position in a frame is restored, and the descrambled output is obtained by combining the three components at the OR 560'.

FIG. 7 illustrates adaptive bit modification with a fixed minimum dynamic range for successive data samples in successive data frames in accordance with the present invention.

The preserved dynamic range of the MSBs can be fixed, e.g., at 16 bits. N , the number of MSBs to be preserved, can be fixed as a preset parameter before processing (N may vary from 0 to the full dynamic range of the digitized samples). $N=8$ in the example shown.

The original waveform 110 indicates the dynamic range of each sample prior to scrambling. Region 720 designates the bits in each sample which are not modified, while a region 730 designates the bits in each sample which may be scrambled. For example, for FRAME A, bits 15 (corresponding to a dynamic range of 16) and higher in each sample are not modified, while bit 0 through bit 14 are modified. For FRAME B and FRAME C, all of the bits are modified, e.g., up to the dynamic range of each sample.

An advantage of this scheme is that there is no need to determine the dynamic range of each sample at the encoder and decoder.

The invention may be implemented with various other alternatives and enhancements, as follows.

- 5 1) For example, referring to FIG. 7, the number of MSBs to be preserved (N) can be dynamic, varying from frame to frame.
- 10 2) For video applications, scrambling can be done across video frames.
- 3) Amplitude range detect and mask generation (e.g., function 535 in FIG. 5, and function 650 in FIG 6) can be a look-up-table with heuristic rules.
- 15 4) Scrambling frames can vary in length.
- 5) Horizontal scrambling can be done on any number of LSBs.
- 6) All or part of the scramble key can be hidden in the same frame, the previous frame, or any other frame of the content. The implementation hinges on the amount of buffer memory available and the throughput delay requirement.
- 20 7) Quiet, silent, or any other special passages can be left unscrambled. In other words, the scrambling can be continuous or discontinuous.
- 25 8) All or part of the scramble key for one stream of the content (e.g., video portion of a DVD disc) can be hidden in a different stream of the same content (e.g., the audio portion of the same DVD disc).
- 30 9) All or part of the scramble key can be hidden in any part of the same media (disc) in any

form or fashion. For example, in the case of physical media such as CD or DVD, the key can be hidden electronically within sector data or physically with the use of techniques such as pit width modulation.

10) All or part of the scramble key for one stream of the content can be hidden in any separated media such as a pre-paid card. The key may also be a product of a transaction authorization via telephone, Internet, or any other communication means.

11) Scrambling can be done between two or more channels of the same data stream (e.g., front left and right channels of a multi-channel audio segment).

12) Multiple channels of the same content stream (e.g., front left and rear right channels of a multi-channel audio segment) can share the same scramble key or have different scramble keys.

13) Scrambling can be selectively done for selected channels of a multi-channel stream (e.g., front left and front right channels are scrambled but center, rear left, rear right and subwoofer channels are not).

14) Scramble Keys can be generated with either non-linear shift registers or non-linear feedback shift registers.

15) For contents in a compressed form, side information coded in the compressed stream, such as bit allocation information, can simplify the range detection process. However, for some frequency

samples, there may not be enough bits allocated to clearly differentiate the sign bit, MSE, and the middle bits. If there are no middle bits, the partial scrambling can simply skip those frequency samples. All other operations described for uncompressed domain partial scrambling should apply.

After the partial scrambling, the modified frequency samples must be re-packaged to conform to the original bit stream format. This in most cases should not require re-quantization, just re-packetization. This process will be specific to the compression technique used. Some compression schemes may require variable length encoding, such as Huffman code, and specific measures must be taken if the size of the compressed bit stream must remain unchanged. Most compression schemes inherently produce variable length bit streams, and the subsequent transport stream format usually accommodates the change in the size of the bit stream.

One side benefit of applying the partial scrambling technique of the present invention to the compressed bit stream may be a slightly easier self synchronization at the decoder. This may occur since the frame structure or the packet structure present in the compressed bit stream format can make it easier for the decoder to determine the scrambling frame boundary.

Accordingly, it can be seen that the present invention provides a system for scrambling digital samples of multimedia data, including audio and

video data samples, such that the content of the
samples is degraded but still recognizable, or
otherwise provided at a desired quality level. The
quality level may be associated with a particular
5 signal to noise ratio, or quality level that is
determined by objective and/or subjective tests, for
example. A number of LSBs can be scrambled in
successive samples in successive frames. Moreover,
the parameters for scrambling may change from frame
10 to frame. Furthermore, the scrambling key can be
embedded in the scrambled data and recovered at a
decoder to descramble the scrambled samples. After
descrambling, the scramble key is no longer
recoverable because the scramble key is scrambled by
15 the descrambler.

Although the invention has been described in
connection with various specific embodiments, those
skilled in the art will appreciate that numerous
adaptations and modifications may be made thereto
20 without departing from the spirit and scope of the
invention as set forth in the claims.

What is claimed is:

1. A method for protecting digital samples of content from illicit use by scrambling the content, wherein each sample includes a plurality of bits, ranging from least significant bits (LSBs) to most significant bits, comprising the step of:

scrambling a number of LSBs in each sample according to a scrambling key, while preserving a number of MSBs in each sample, to provide corresponding scrambled samples; wherein:

a number of LSBs is scrambled in each sample such that the scrambled samples are degraded but still recognizable.

2. The method of claim 1, comprising the further step of:

determining a dynamic range of each sample, and adaptively selecting the number of LSBs to be scrambled in each sample according to the dynamic range thereof.

3. The method of claim 1, wherein the samples are provided in successive frames, comprising the further step of:

adaptively selecting the number of LSBs to be scrambled in each sample according to the frame thereof.

4. The method of claim 1, wherein:

said scrambling key is a pseudo-random scrambling key.

5. The method of claim 1, wherein:
in said scrambling step, the number of LSEs are scrambled within the same sample using intra-sample scrambling.

6. The method of claim 1, wherein:
in said scrambling step, the number of LSEs are scrambled between different samples using inter-sample scrambling.

7. The method of claim 1, wherein:
in said scrambling step, the number of LSEs are scrambled between different samples, and within the same sample, using inter-sample and intra-sample scrambling, respectively.

8. The method of claim 1, wherein:
in said scrambling step, the number of LSEs are scrambled between different samples using horizontal inter-sample scrambling by interchanging bits having the same weight.

9. The method of claim 1, comprising the further step of:
embedding the scrambling key, at least in part, into the scrambled samples for use at a decoder in descrambling the scrambled samples.

range thereof.
described in each sample according to the dynamic
selectively selecting the number of bits to be
determining a dynamic range of each sample, and
further step of:
13. The method of claim 12, comprising the

still recognizable.
such that the scrambled samples are degraded but
a number of bits is scrambled in each sample
corresponding described samples, wherein:
number of bits in each sample, to provide
according to a scrambling key, while preserving a
describing a number of bits in each sample
bits, comprising the step of:
least significant bits (LSBs) to most significant
sample includes a plurality of bits, ranging from
scrambled digital samples of content, wherein each
12. A method for descrambling previously

into a current frame of samples.
scrambled samples is embedded, at least in part,
the scrambling key for a current frame of
11. The method of claim 9, wherein:

into a previous frame of samples.
scrambled samples is embedded, at least in part,
the scrambling key for a current frame of
10. The method of claim 9, wherein:

14. The method of claim 12, wherein the samples are provided in successive frames, comprising the further step of:
 adaptively selecting the number of LSBs to be descrambled in each sample according to the frame thereof.
15. The method of claim 12, wherein:
 said scrambling key is a pseudo-random scrambling key.
16. The method of claim 12, wherein:
 in said descrambling step, the number of LSBs are descrambled within the same sample using intra-sample descrambling.
17. The method of claim 12, wherein:
 in said descrambling step, the number of LSBs are descrambled between different samples using inter-sample descrambling.
18. The method of claim 12, wherein:
 in said descrambling step, the number of LSBs are descrambled between different samples, and within the same sample, using inter-sample and intra-sample descrambling, respectively.
19. The method of claim 12, wherein:
 in said descrambling step, the number of LSBs are descrambled between different samples using

horizontal inter-sample descrambling by interchanging bits having the same weight

20. The method of claim 12, wherein the scrambling key is embedded into the scrambled samples, comprising the further step of:
recovering the scrambling key from the scrambled samples for use in said descrambling step.

21. The method of claim 20, wherein:
the scrambling key for a current frame of scrambled samples is embedded into a previous frame of samples.

22. The method of claim 12, comprising the further steps of:
scrambling the scrambling key after descrambling the scrambled sample in said descrambling step.

23. An apparatus for protecting digital samples of content from illicit use by scrambling the content, wherein each sample includes a plurality of bits, ranging from least significant bits (LSBs) to most significant bits, comprising:
a scrambler for scrambling a number of LSBs in each sample according to a scrambling key, while preserving a number of MSBs in each sample, to provide corresponding scrambled samples; wherein:

a number of LSBs is scrambled in each sample such that the scrambled samples are degraded but still recognizable.

24. An apparatus for descrambling previously scrambled digital samples of content, wherein each sample includes a plurality of bits, ranging from least significant bits (LSBs) to most significant bits, comprising:

a descrambler for descrambling a number of LSBs in each sample according to a scrambling key, while preserving a number of MSBs in each sample, to provide corresponding descrambled samples; wherein:

a number of LSBs is scrambled in each sample such that the scrambled samples are degraded but still recognizable.

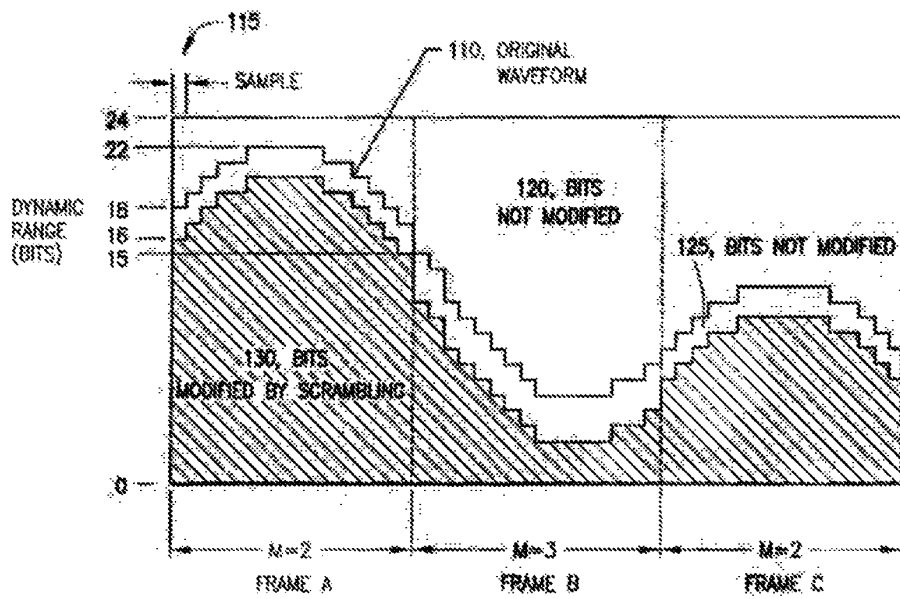


FIG. 1

95/032969

FIG. 1 (3) (2) (3) (4) (5)



FIG.2a

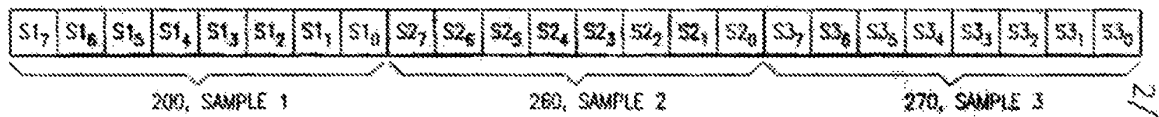


FIG.2b

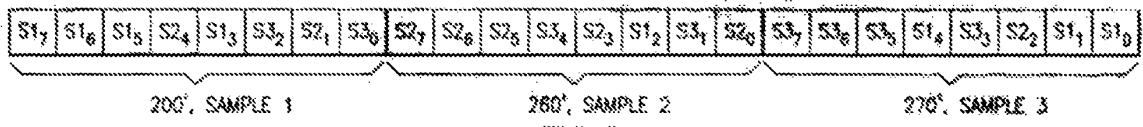


FIG.2c

10/10/2016 10:04:04 AM

10/10/2016 10:04:04 AM

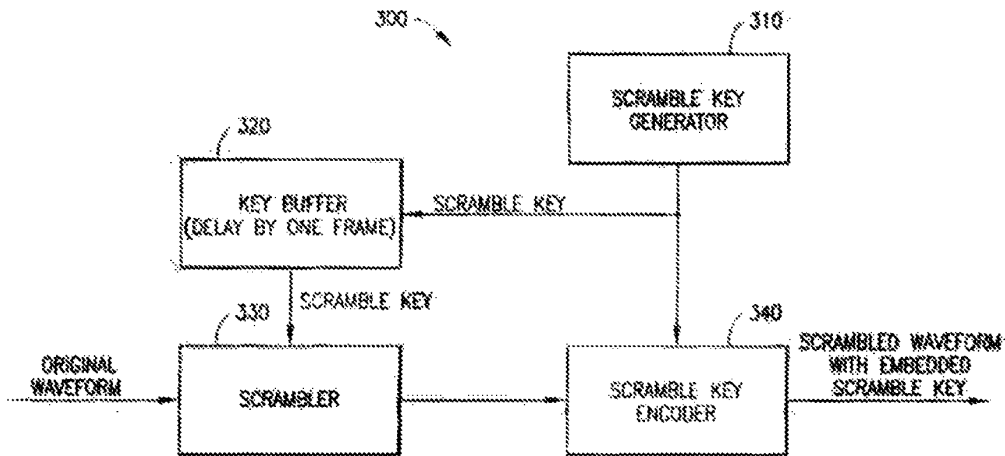


FIG.3

3/7

WFO 9/15/2019

PCCT19-0989424

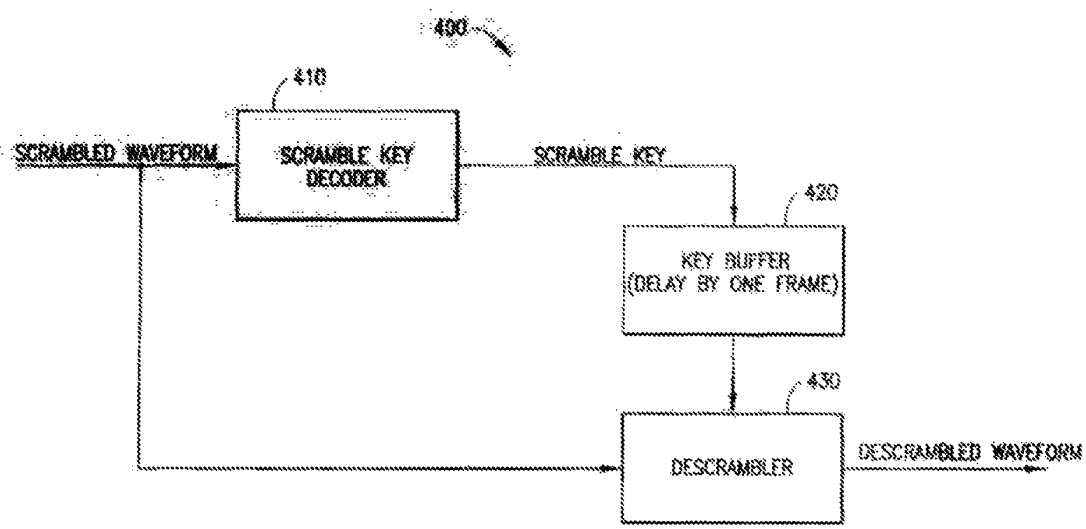


FIG. 4

8/13/2009

4/7

8/13/2009

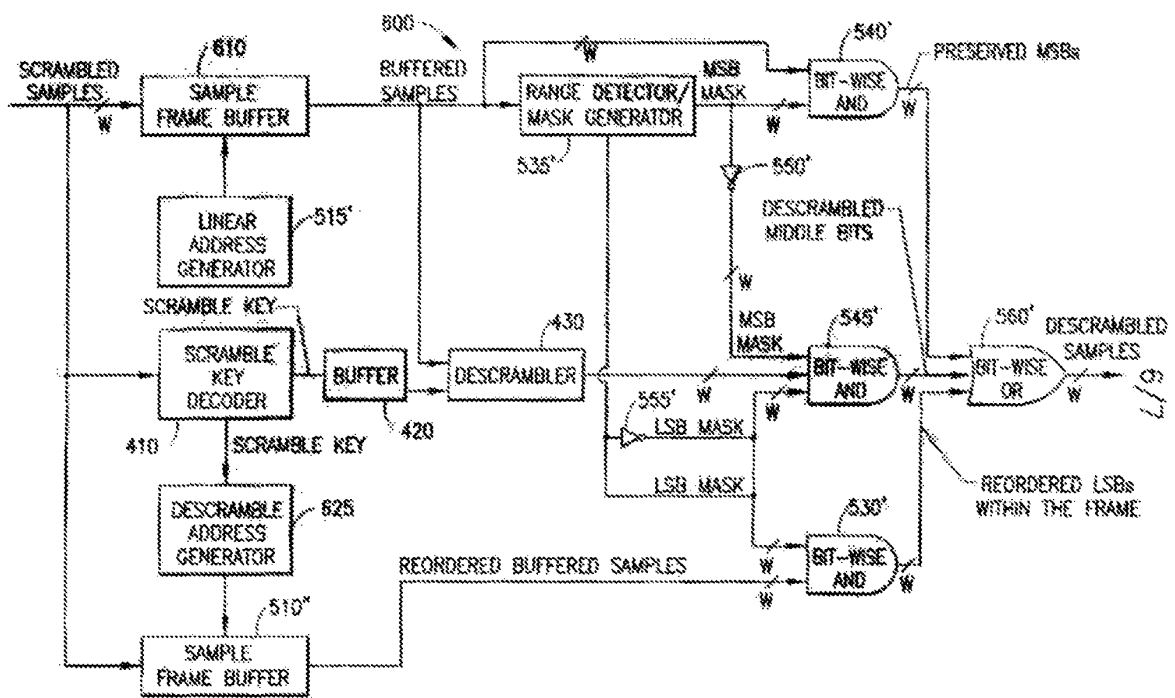


FIG. 6

60066969.02 WA

FIG. 1A (REV. 10/26/05)

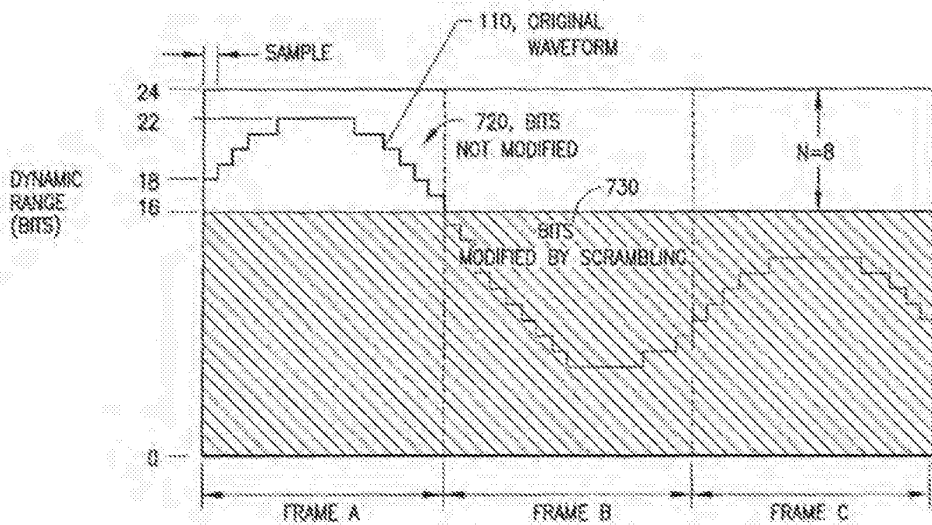


FIG. 7

1005/1005/01

7/7

1005/1005/01

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US99/08635

A. CLASSIFICATION OF SUBJECT MATTER
 IPC CL. 3B4R 7/167
 US CL. 713/206
 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)
 US 3B4, 5, 19, 20
 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 phase of this phase and, where practicable, search terms used)
 Phoenix Rev 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 5,535,275 A (SUGISAKI ET AL) 09 JULY 1996, abstract, col. 10, lines 56-67; col. 11, lines 1-4	1, 4, 9, 10, 12, 15, 20, 21, 23, 24
Y	US 3,784,743 A (SCHROEDER) 08 JANUARY 1974, abstract, col. 2, lines 40-50; claim 9	1, 4, 12, 15, 23, 24
Y	US 5,185,794 A (THOMPSON ET AL) 09 FEBRUARY 1993, col. 9, lines 36-68; col. 10, lines 1-2	9, 10, 20, 21

Further documents are listed in the continuation of Box C. See patent family sheet.

* Special categories of cited documents:
 (X) documents defining the general state of the art which is not considered to be of particular relevance
 (Y) earlier document published on or after the international filing date
 (Z) documents which may have doubt as to priority claims or which a check is essential to establish the publication date of another document or other special action (as specified)
 (A) document referring to an oral disclosure, use, exhibition or other means
 (B) document published prior to the international filing date but later than the priority date claimed
 (C) documents published after the international filing date or priority date and not in conflict with the application but cited to enlighten the principles or theory underlying the invention
 (D) document of particular relevance, the claimed invention cannot be distinguished therefrom or cannot be considered to derive therefrom step when the document is taken alone
 (E) disclosure of prior art, the claimed invention cannot be distinguished therefrom as a whole step when the document is considered with one or more other such documents, such combinations being obvious to a person skilled in the art
 (F) document transfer of the same priority

Date of the actual completion of the international search: 29 JULY 1999
 Date of mailing of the international search report: 30 AUG 1999

Name and mailing address of the ISA/OIS: Commissioner of Patents and Trademarks, Box PCT, Washington, DC 20531, Telephone No. (703) 305-0040
 Authorized officer: GAIL O. HAYES, James R. Matthews, Telephone No. (703) 306-5417

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US99/0635

B. FIELDS SEARCHED

Electronic data bases consulted (Name of data base and where practicable terms used):

AMS

Search terms: scrambling, encrypt, digital, signal, audio, sound, random, pseudorandom,
key, frame, bit



<p>(51) 国際特許分類6 G11B 20/10</p>	<p>A1</p>	<p>(11) 国際公開番号 WO99/42996</p>
<p>(21) 国際出願番号 PCT/JP99/00702 (22) 国際出願日 1999年2月18日(18.02.99) (30) 優先権データ 特願平10/37184 1998年2月19日(19.02.98) JP (71) 出願人 (米国を除くすべての指定国について) ソニー株式会社(SONY CORPORATION)[JP/JP] 〒141-0001 東京都品川区北品川6丁目7番35号 Tokyo, (JP) (72) 発明者; および (75) 発明者/出願人 (米国についてのみ) 福田真一(FUKUDA, Shinichi)[JP/JP] 〒141-0001 東京都品川区北品川6丁目7番35号 ソニー株式会社内 Tokyo, (JP) (74) 代理人 弁理士 杉浦正知(SUGIURA, Masatomo) 〒170-0013 東京都豊島区東池袋1丁目48番10号 25山京ビル420号 Tokyo, (JP)</p>		<p>(81) 指定国 JP, KR, US, 欧州特許 (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE) 添付公開書類 国際調査報告書</p>
<p>(54)Title: RECORDER / REPRODUCER, RECORDING / REPRODUCING METHOD, AND DATA PROCESSOR</p>		
<p>(54)発明の名称 記録再生装置、記録再生方法、ならびに、データ処理装置</p>		
<p>(57) Abstract When copying music data recorded on an HD to another electronic apparatus is ordered, a charging or a sound quality deterioration is selected. When a charging is selected, a specified charging procedure is performed before data is copied to the electronic apparatus. When the charging is performed, the data quality is kept at substantially the same level as the original level. When a sound quality deterioration is selected, data conversion is performed through a specified sound quality deterioration and the data quality is deteriorated before being outputted to the electronic apparatus. In this case, no charging is performed. When data is moved, neither charging nor sound quality deterioration is performed.</p>		

(57)要約

HDDに記録されている音楽データの他の電子機器への複製が指示されると、課金処理を行うか音質劣化処理を行うかが選択される。課金処理を行う場合には、所定の課金手続きがなされた後にデータの複製処理が行われ、複製先に対してデータが出力される。課金処理を行った場合には、データの品質は、オリジナルと略同一に保たれる。一方、音質劣化処理が選択された場合は、所定の音質劣化処理によってデータ変換がなされてデータの品質が落とされ、複製先に対して出力される。この場合、課金は行われぬ。なお、データの移動の場合は、課金処理ならびに音質劣化処理は施されない。

PCTに基づいて公開される国際出願のパンフレット第一頁に掲載されたPCT加盟国を同定するために使用されるコード(参考情報)

AE アラブ首長国連邦	ES スペイン	LJ リヒテンシュタイン	SG シンガポール
AL アルバニア	FI フィンランド	LK スリランカ	SI スロヴェニア
AM アルメニア	FR フランス	LR リベリア	SK スロヴァキア
AT オーストリア	GA ガボン	LS レソト	SL シェラ・レオネ
AU オーストラリア	GB 英国	LT リトアニア	SN セネガル
AZ アゼルバイジャン	GD グレナダ	LU ルクセンブルグ	SZ スワジランド
BA ボスニア・ヘルツェゴビナ	GE グルジア	LV ラトヴィア	TD チャード
BB バルバドス	GH ガーナ	MC モナコ	TG トーゴ
BE ベルギー	GM ガンビア	MD モルドヴァ	TJ タジキスタン
BF ブルキナ・ファソ	GN ギニア	MG マダガスカル	TM トルクメニスタン
BG ブルガリア	GW ギニア・ビサウ	MK マケドニア旧ユーゴスラヴィア	TR トルコ
BJ ベナン	GR ギリシャ	共和国	TT トリニダード・トバゴ
BR ブラジル	HR クロアチア	ML マリ	UA ウクライナ
BY ベラルーシ	HU ハンガリー	MN モンゴル	UG ウガンダ
CA カナダ	ID インドネシア	MR モーリタニア	US 米国
CC コクゾウ	IE アイルランド	MW マラウイ	UZ ウズベキスタン
CH スイス	IL イスラエル	MX メキシコ	VN ヴェトナム
CI コートジボアール	IN インド	NE ニジェール	YU ニューギニア
CM カメルーン	IS アイスランド	NL オランダ	ZA 南アフリカ共和国
CN 中国	IT イタリア	NO ノールウェー	ZW ジンバブエ
CU キューバ	JP 日本	NZ ニュー・ジールランド	
CY キプロス	KE ケニア	PL ポーランド	
CZ チェッコ	KG キルギスタン	PT ポルトガル	
DE ドイツ	KP 北朝鮮	RO ルーマニア	
DK デンマーク	KR 韓国	RU ロシア	
EE エストニア	KZ カザフスタン	SD スーダン	
	LC セントルシア	SE スウェーデン	

明細書

記録再生装置、記録再生方法、ならびに、データ処理装置

技術分野

本発明は、記録再生装置、記録再生方法、ならびに、データ処理装置
5 に関する。特に、本発明は、データの複製または複製と同等の動作
を行うための記録再生装置、記録再生方法、ならびに、データ処理装
置に関する。

背景技術

近年、デジタル技術の発達に伴い、CD (Compact Disc) などの記
10 録メディアを始めとして、オーディオ情報またはオーディオデータが
デジタル音楽データとして供給される例が多くなっている。ディ
ジタル音楽データは、高音質による再生を比較的容易に実現でき、記録
メディアに記録することによる音質の劣化が殆ど生じない。記録メ
ディアが従来のアナログのオーディオ信号を記録していた場合と比べ、
15 デジタル音楽データが記録されている記録メディアは比較的小型、
軽量であるなどの理由で、広く一般的に普及している。

一方、デジタル音楽データは、記録メディアによる音質の劣化が
殆ど生じないことと、完全な複製が容易に可能であることなどから、
無制限な複製などにより著作権を脅かすおそれがある。従来では、著
20 作権保護の対策として、著作権が含まれるデジタル音楽データの複
製（ダビング）に関して、SCMS (Serial Copy Management System
) による管理が行われていた。SCMSによる管理は、デジタル音
楽データに対して所定のフラグを設け、デジタルデータとしての複
製を、例えば1世代に制限するものである。SCMSは、民生用のD
25 AT (Digital Audio Tape) や直径略64mmの光磁気ディスクを用い
る記録再生装置などのデジタルインターフェイスとして採用されて

いる。

ところが、上述したSCMSでは、著作権保護を重視するあまり、データの共有性が著しく限定されてしまうという問題点があった。

例えば、デジタルデータを複製して入手した音楽データは、たと
5 え個人としての使用目的であっても、デジタルデータが複製された記録メディアから他の記録メディアへと2度と移すことができず、非常に不便であるという問題点があった。

発明の開示

したがって、本発明の目的は、デジタル音楽データの他の記録メ
10 ディアへのデジタル的な移動ならびに複製を、著作権を保護しながら行えるような記録システムを提供することにある。

この発明は、上述の問題を解決するために、少なくともひとつのデータが記憶されている第1の記憶部と、少なくともひとつのデータが記録される第2の記憶部と、第1の記憶部から読み出されたデータが
15 第2の記憶部に記憶される際に第1の記憶部から読み出されたデータが供給され、供給されたデータの劣化処理を行って第2の記憶部に供給する信号処理手段を備えている記憶再生装置である。また、この発明は、第1の記憶部から読み出されたデータが第2の記憶部に記憶される際に、第1の記憶部から読み出されたデータにデータの劣化処
20 理を施して第2の記憶部に供給し、第2の記憶部に書き込まれる記録再生方法である。

また、この発明は、少なくともひとつのデータが記憶されている第1の記憶部と、少なくともひとつのデータが記録される第2の記憶部と、第1の記憶部から読み出されたデータが供給されるとともに、第
25 1の記憶部から読み出されたデータが第2の記憶部に記憶される際に課金処理と第1の記憶部から供給されたデータの劣化処理のうちのい

ずれ一方の処理を行って第 2 の記憶部に供給する制御手段を備えている記録再生装置である。

また、この発明は、第 1 の記憶部から読み出されたデータが第 2 の記憶部に記憶される際に課金処理と第 1 の記憶部から供給されたデータの劣化処理のうちのいずれか一方の処理を行って第 2 の記憶部に供給する記録再生方法である。

また、この発明は、少なくともひとつのデータが記憶されている記憶部と、記憶部から読み出されたデータが他の記憶媒体に複製される際に記憶される際に記憶部から読み出されたデータが供給され、供給されたデータの劣化処理を行い、データの劣化処理が施されたデータを出力データとして出力する信号処理手段を備えているデータ処理装置である。

上述したように、請求の範囲 1 および請求の範囲 1 3 に記載のこの発明によれば、少なくともひとつのデータが記憶されている第 1 の記憶部から読み出されたデータが少なくともひとつのデータが記憶される第 2 の記憶部に記憶される際に、第 1 の記憶部から読み出されたデータに対して劣化処理を行うようにしているため、第 1 の記憶部から読み出されたデータの不正な利用が防がれる。

また、請求の範囲 2 5 および請求の範囲 3 9 に記載のこの発明によれば、少なくともひとつのデータが記憶されている第 1 の記憶部から読み出されたデータが少なくともひとつのデータが記憶される第 2 の記憶部に記憶される際に、課金処理と、第 1 の記憶部から読み出されたデータの劣化処理とのうち、何れか一方を選択できるようにしているため、課金処理を行えばデータの劣化処理が行われないようにできる。

また、請求の範囲 4 3 に記載のこの発明によれば、少なくともひとつ

つのデータが記憶されている記憶部から読み出されたデータが他の記憶媒体に複製される際に記憶される際に、記憶部から読み出されたデータに対して劣化処理を施して出力データとしているため、記憶部から読み出され他の記憶媒体に複製されるために出力されるデータの不正な利用が防がれる。

図面の簡単な説明

第1図は、本発明によるミュージックサーバおよびミュージックサーバを用いたシステムを概略的に示す略線図、第2図は、ミュージックサーバの構成の一例を示すブロック図、第3図は、CD-ROMドライブで読み出された音楽データがハードディスクドライブに記録されるまでの信号の流れを概略的に示す図、第4図は、ハードディスクドライブから読み出された圧縮音楽データが再生処理されて端子に導出されるまでの信号の流れを概略的に示す図、第5図は、携帯記録再生装置の構成の一例を示すブロック図、第6図は、携帯記録再生装置の他の例を示すブロック図、第7図は、ミュージックサーバのCDの音楽データをハードディスクドライブに記録する際の処理の一例のフローチャート、第8図は、CDの音楽データをハードディスクドライブに高速記録する際の課金処理の一例を示すフローチャート、第9図は、本発明による音楽データの移動の処理の一例のフローチャート、第10図は、音質劣化処理の方法の一例を示す略線図、第11図は、音質劣化処理の方法の一例を示す略線図、第12図は、音質劣化処理の方法の一例を示す略線図、第13図は、音質劣化処理の方法の一例を示す略線図、第14図は、音質劣化処理の方法の一例を示す略線図、第15図は、音質劣化処理の方法の一例を示す略線図、第16図は、音楽データ複製の際の課金処理と音質劣化処理を選択する処理の一例を示すフローチャートである。

発明を実施するための最良の形態

以下、この発明を実施するための最良の形態を、図面を参照しながら説明する。第1図は、この発明が適用されたミュージックサーバおよびミュージックサーバを用いたシステムを概略的に示している。ミュージックサーバ50は、サーバ本体51と左右のスピーカユニット52L、52Rとからなる。サーバ本体51には、例えばLCD(Liquid Crystal Display)パネルからなる表示部53と、CDをサーバ本体51に挿入するためのCD挿入部54とが設けられる。

なお、第1図では省略されているが、サーバ本体51の機能をユーザが操作するための複数の操作スイッチからなる操作部がサーバ本体51に設けられる。サーバ本体51の機能をリモートコマンドによって遠隔操作するための、例えば赤外線信号を受信する信号受信部を設けるようにしてもよい。サーバ本体51は、後述するようにコントローラを有し、サーバ本体51は、予め例えばROMに記憶される所定のプログラムに基づいてコントローラにより各種動作が制御される。

ユーザは、CD55をCD挿入部54を介してサーバ本体51に装填し、図示しない操作部を用いて所定の操作を行うことで、CD55を再生し、CD55より再生された再生信号をスピーカユニット52L、52Rから出力することによって、CD55に記録されている音楽を楽しむことができる。CD55が曲名などのテキストデータを含む場合は、表示部53にテキストデータに基づいて曲名などが表示される。

ミュージックサーバ50は、内部に例えばハードディスクによる大容量の記録媒体を有している。図示されない操作部を用いてユーザが所定の操作を行うことによって、CD挿入部54からサーバ本体51に装填されたCD55から再生された再生データを、このハードディ

スクからなる記録媒体に記録することができる。この際、CD 55の標準の再生速度と同一の転送速度で記録する方法と、CD 55の標準の再生速度より高速の転送速度で記録を行う高速記録とを選択することができる。高速の転送速度で記録を行う場合には、所定の手続きで

5 以て課金処理を行うことによって、CDの選択またはCDに記録されている曲の選択と、CDから再生された音楽データとしての再生データを、CDの標準の再生速度で再生されたデータの転送速度よりも速い転送速度で記録することができる。

ミュージックサーバ50において、CD 55から再生された音楽データは、上述したATRACなどの所定の方法で圧縮符号化され圧縮音楽データとされて記録され、例えば6 G B y t eの容量を持つハードディスクに、1000曲程度を記憶または格納できる。ハードディスクに記憶または格納された曲目のリストが例えば表示部53に表示され、ユーザは、表示部53に表示されている曲名リストに基づき、

15 ハードディスクに記憶または格納されている曲のうちの任意の曲を選択して再生させることができる。ハードディスクは、ランダムアクセスが可能であるため、多数記憶または格納された音楽データを任意の順序で読み出して、連続再生させることが可能である。

圧縮符号化には様々な方法を用いることが可能であるが、本例では

20 、例えば米国特許5717821号に開示されているような、ATRAC 2 (Adaptive Transform Acoustic Coding 2) と称される方法が用いられている。この米国特許5717821号に開示されている圧縮符号化方法は、聴覚の性質に基づくマスキング効果および最小可聴限の周波数依存性を利用し、変換符号化とエントロピー・コーディング

25 とを併用して音声データの圧縮符号化を行うものであり、比較的小規模なハードウェアで、高音質を維持しつつ、高速にエンコード/デコ

ードを行うことができる。

このミュージックサーバ50は、例えば公衆電話回線である通信回線61を介して外部のシステム、例えばインターネットに接続されたサーバであるインターネットサーバ60に接続できる。ミュージックサーバ50から通信回線61を介してこのインターネットサーバ60に接続することで、インターネット上にある様々な情報を得ることができる。インターネットサーバ60は、例えば市販の音楽CDのタイトル情報などのデータベースを有する。ユーザには、このデータベースを利用するためのユーザ固有のキーを与え、データベースを利用する際に固有のキーに基づいて認証を行い、ユーザの認証が行われた後に、CDに付随したデータ、例えばCDのタイトル情報をユーザに送信することで、ユーザは、CDに付随したデータを得ることができる。

インターネットサーバ60では、ユーザに供給するサービスに応じてミュージックサーバ50に対する課金処理も行う。上述した、CD55の上述した高速記録を行う場合は、インターネットサーバ60にミュージックサーバ50が高速記録を行う旨のデータの通信を行うことによって、高速記録を行うユーザに対する課金処理が行われ、CDの選択や曲の選択、ならびに、高速記録の実行が可能とされる。

なお、ここでは、課金処理を、CDの付加情報を多数有するインターネットサーバ60で行うこととしたが、これは上述した例に限定されない。例えば、インターネットに接続された別のサーバで上述した課金処理を行うようにしてもよい。インターネットとは別の、例えば専用のネットワークで以て上述した課金処理を行うようにすることも可能である。

携帯記録再生装置70は、ハードディスクあるいはフラッシュメモ

リからなる記憶媒体を有する。音楽の再生速度に追従できるのであれば、他の記憶媒体または記録媒体を利用することもできる。この携帯記録再生装置70を接続線71で以てミュージックサーバ50と接続することによって、ミュージックサーバ50に記録されている音楽データ5を携帯記録再生装置70に転送し、携帯記録再生装置70の記憶媒体に記録することができる。このとき、ミュージックサーバ50側では、装置70に転送された音楽データは、ハードディスクやフラッシュメモリの記憶媒体上には存在するが再生不可の状態にされる。携帯記録再生装置70で用いられる記憶媒体または記録媒体は、例えば10 200 M B y t e程度の容量とされ、数10曲分の音楽データの記憶または格納することができる。なお、以下の説明では、フラッシュメモリなどの半導体メモリからなる記憶素子または記憶媒体と、ハードディスクなどのディスク状記録媒体などの記録媒体を総称して、記憶媒体と称することにする。

15 この発明において用いられる上述の転送方法、すなわち、音楽データを転送した場合、転送先の記憶媒体に音楽データが記録されると共に、転送元の記憶媒体においては、転送された音楽データが記憶媒体上には存在するが再生不可の状態にされることを、「移動」と称する。このように音楽データを移動することで、音楽データの無制限な複製を防ぐことができる。

20 なお、上述した例では、ミュージックサーバ50と携帯記録再生装置70とが接続線71で接続されたとしたが、これはこの例に限定されない。例えば、ミュージックサーバ50および携帯記録再生装置70とに、互いに対応する装着部を設け、ミュージックサーバ50に携帯記録再生装置70を直接装着してサーバ50と装置70との間でデータ25のやり取りを行うようにできる。電氣的な接続だけでなく、例え

ば赤外線信号によりデータのやり取りを行う I r D A (Infrared Data Association) に対応したインターフェイスをサーバ 5 0 と装置 7 0 との双方に設け、赤外線信号により音楽データの転送をサーバ 5 0 と装置 7 0 との間で行うようにしてもよい。

- 5 さらに、ミュージックサーバ 5 0 に所定のインターフェイスを設けることで、様々なメディアと情報交換を行うことができるようになる。例えば、サーバ 5 0 に P C カード 8 0 に対応したインターフェイスを設けることで、P C カード 8 0 で配信される音楽データをミュージックサーバ 5 0 に取り込んだり、パーソナルコンピュータとミュージックサーバ 5 0 との間でデータのやり取りを行うことが可能となる。
- 10 サーバ 5 0 に光ケーブルなどによるシリアルなデジタルインターフェイスを設けることによって、例えば直径略 6 4 m m の小型の光磁気ディスクを用いるディスクレコーダ 8 1 のような、他のデジタル音楽データ記録再生装置との音楽データのやり取りを行うことが可能となる。
- 15 この例では、ディスクレコーダ 8 1 に上述した小型の光磁気ディスクが収納されたディスクカートリッジ 8 2 が装着され、ディスクカートリッジ 8 2 の光磁気ディスクから再生された音楽データがミュージックサーバ 5 0 に対して供給される。同様に、サーバ 5 0 に I E E E 1 3 9 4 などのインターフェイスを設け、例えば C A T V (Cable Television) や衛星放送などのためのセットトップボックス 8 3 を
- 20 接続するようにもできる。

- P C カードは、米国の P C M C I A (Personal Memory Card International Association) と日本の J E I D A (日本電子工業振興会) の共同制定による、パーソナルコンピュータ用のカード型周辺機器の規格である。I E E E 1 3 9 4 は、米国電気電子技術者協会に採択されたインターフェイス規格である。
- 25

ミュージックサーバ50は、内蔵アプリケーションとして、WWW (World Wide Web) ブラウザを持つようにできる。通信回線61を介してインターネットサーバ60と接続することによって、インターネット上にある、例えばHTML (Hypertext Markup Language) によって
5 記述された様々なコンテンツを検索し、表示部53上に表示させることができる。

このような構成で以て、ユーザは、例えばミュージックサーバ50に記憶または格納されている音楽データを再生してスピーカユニット52L, 52Rで聴くことができると共に、CD55をCD挿入部5
10 4を介してサーバ50に装填して、CD55を再生することができる。

ミュージックサーバ50とインターネットサーバ60とで通信を行うことによって、CD挿入部54を介してサーバ50に装填されたCD55のタイトル情報などを、通信回線61を介してサーバ60から
15 自動的に得ることができる。サーバ60から得られた情報は、ミュージックサーバ50内に保存されると共に、保存されたタイトル情報は、必要に応じてサーバ50の表示部53に表示される。より具体的には、ミュージックサーバ50からインターネットサーバ60に対して、サーバ50のユーザIDデータなどのユーザ固有の情報（以下、
20 ユーザ情報と称する）が送られる。インターネットサーバ60側では、受け取ったユーザ情報に基づき、照合処理や課金処理が行われる。また、ミュージックサーバ50からインターネットサーバ60に対して、ユーザが必要とするCDまたは再生しているCDのメディア情報が送られる。インターネットサーバ60では、受け取ったメディア情
25 報に基づき、例えば曲のタイトル、演奏者名、作曲者や作詞者名、歌詞、ジャケットイメージといった、音楽データに対する付加情報の検

索が行われる。そして、インターネットサーバ60では、ユーザから要求されたCDに関する所定の情報をミュージックサーバ50に返信する。

例えば、メディア情報として、CD55のTOC (Table Of Contents) 5 情報をインターネットサーバ60に対して送る。インターネットサーバ60には、このTOC情報に基づいて上述の音楽データに対する付加情報が検索可能なデータベースが構築されている。インターネット上の他のWWWサーバを検索することで付加情報を得るようにしてもよい。インターネットサーバ60は、受け取ったTOC情報をメ
10 デディア情報として、音楽データの付加情報の検索を行う。これは、例えば、TOC情報に含まれる、CD55に収録されている楽曲それぞれの時間情報に基づき検索することが可能である。

検索されて得られた付加情報がインターネットサーバ60からミュージックサーバ50に対して送られる。ミュージックサーバ50では
15 、受信した付加情報が表示部53に表示されると共に、後述するCPU8により、例えばハードディスクドライブにCD55のTOC情報と共に書き込まれる。なお、検索された付加情報をHTMLファイルに埋め込んでサーバ60から送ることで、ミュージックサーバ50において、内蔵されるWWWブラウザソフトで付加情報の表示を行うこ
20 とができる。

付加情報にインターネット上の他のURL (Uniform Resource Locator) が記述されていれば、このミュージックサーバ50においてその他のURLで示される、インターネット上のホームページなどにアクセスするようにできる。

25 さらに、インターネットサーバ60とサーバ50との間でデータの通信を行うことによって、CD挿入部54を介してサーバ50に装填

されたCD 55の音楽データを、ミュージックサーバ50の記憶媒体に、CD 55の規定されている標準の再生速度よりも高速で、例えばCD 55の1枚分の音楽データを2分程度で記録することができる。インターネットサーバ60とサーバ50との間で通信を行わないとき
5 には、CD 55の規定されている標準の再生速度と等しい速度、1倍速でサーバ50の記憶媒体に記録される。

サーバ50は、携帯記録再生装置70と接続線71で接続することで、ミュージックサーバ50に記憶または格納されている音楽データを携帯再生装置71に転送して移動することができる。移動された音
10 楽データは、サーバ50と装置71とが接続線71によって接続されていない状態でも、携帯記録再生装置70で再生することができ、例えばヘッドホン72で聴くことができる。転送され移動された音楽データは、ミュージックサーバ50では、再生不可の状態とされる。

第2図は、ミュージックサーバ50の構成の一例を示す。まず、このミュージックサーバ50において、通常のパーソナルコンピュータ
15 構成と同様に、互いにバスで結合されたRAM5、ROM6、フラッシュメモリ7、およびCPU8とが設けられる。CPU8がバス40に接続される。CPU8は、マイクロコンピュータなどから構成され、コントローラとしてミュージックサーバ50の全体の動作を制御
20 する。

ROM6には、このミュージックサーバ50の動作を制御するためのプログラムが予め記憶される。ミュージックサーバ50において、このプログラムに基づき、CPU8が後述する入力操作部1の操作に対応した動作を実行させる。RAM5、フラッシュメモリ7には、プ
25 ログラムを実行する上でのデータ領域、タスク領域が一時的に確保される。ROM6にはプログラムローダが記憶されており、ROM6の

プログラムローダにより、フラッシュメモリ 7 にプログラム自体がロードされることも可能である。

入力操作部 1 は、例えば、複数のプッシュ式および回動式のキー操作キーと、これらの操作キーによって各々操作されるスイッチなどからなる。入力操作部 1 は、これに限らず、ジョグダイヤルと呼ばれる回動プッシュ式の操作キー、LCD 上のタッチパネルなどでもかまわない。勿論、押下することで反応するスイッチ機構を用いることもできる。この入力操作部 1 の操作に応じた信号がバス 40 を介して CPU 8 に供給される。CPU 8 において、入力操作部 1 からの信号に基づきミュージックサーバ 50 の動作を制御するための制御信号が生成される。ミュージックサーバ 50 は、CPU 8 で生成された制御信号に応じて動作される。

バス 40 に対して、赤外線インタフェース (IrDA I/F) ドライバ 3 および/または USB (Universal Serial Bus) ドライバ 4 が接続される。これらのドライバ 3、4 に対してキーボード 2 が通信あるいは接続可能なようにされている。キーボード 2 を用いることによって、例えば記録される音楽データに対応する曲名、アーティスト名等の入力を容易に行うことができる。また、赤外線インターフェースドライバ 3 あるいは USB ドライバ 4 を介してデータ転送を行うように構成してもよい。なお、これら赤外線インターフェース 3 および USB ドライバ 4 は、省略することが可能である。CD-ROM ドライブ 9 がバス 40 に接続され、CD-ROM ドライブ 9 に、上述したようにディスク挿入部 54 から挿入された CD 55 が装填される。この CD-ROM ドライブ 9 では、セットされた CD 55 から規定されている標準の再生速度で音楽データが読み出される。また、この CD-ROM ドライブ 9 では、規定されている標準の再生速度よりも

高速な、例えば規定されている標準の再生速度の1.6倍や3.2倍といった速度で、CD 55の音楽データを読み出すことができる。

なお、CD-ROMドライブ9は、上述の例に限らず、音楽データが記憶されている他のディスク状の記録媒体、例えば光磁気ディスク5やDVD (Digital Versatile Disc: 商標) に対応するようによい。メモ리카ードに対応したドライブを用いることもできる。さらに、CD-ROMドライブ9から読み出されるデータは、音楽データに限られない。画像データやテキストデータ、プログラムデータなどを読み出すようにもできる。

10 バス40に対して、ハードディスクドライブ（以下、HDDと略称する）10が接続される。HDD10には、CD-ROMドライブ9から読み出された音楽データが記録される。HDD10に音楽データが記録される前処理として、CD-ROMドライブ9で読み出された音楽データは、バス40ならびにオーディオ用のDRAM11を介し
15 て、圧縮エンコーダ12に供給される。

圧縮エンコーダ12では、例えば、上述した例えば米国特許5717821号などに開示されている圧縮方法によって音楽データの圧縮符号化処理が行われる。なお、圧縮エンコーダ12による音楽データの圧縮の速度は、CPU8の制御に基づき、低速および高速の2つの
20 速度が用意される。低速圧縮速度は、CD-ROMドライブ9でCD55に規定されている標準の再生速度に対応する。圧縮の速度は、例えばCD-ROMドライブ9によるCD55の再生速度に応じて切り替えられる。圧縮エンコーダ12において、例えば、圧縮速度に応じたエンコードアルゴリズムが駆動される。

25 圧縮エンコーダ12における圧縮速度の変更は、上述した方法に限定されない。例えば、圧縮エンコーダ12のクロック周波数を切り替

えることによつて行つてもよいし、それぞれ別のハードウェアを用意するようにしてもよい。さらに、高速圧縮が可能な圧縮エンコーダ 12 において、処理を間引きして行い低速圧縮速度に対応するようにしてもよい。

- 5 圧縮エンコーダ 12 で圧縮符号化された圧縮音楽データは、D R A M 11 を介して H D D 10 に記録され蓄積される。

ここで、圧縮エンコーダ 12 により圧縮符号化された圧縮音楽データが H D D 10 に蓄積されるように構成されているが、C D - R O M ドライブ 9 から読み出される音楽データを直接的に H D D 10 に供給
10 して H D D 10 のハードディスクに記録ならびに蓄積するようにもできる。

本例では、端子 13 に接続されたマイクロホンからアンプ 14 を介して入力される音声信号や、ライン入力端 15 から入力される音声信号が A / D コンバータ 16 を介して圧縮エンコーダ 12 に供給される
15 。 A / D コンバータ 16 から出力されたこれらの音声信号を、エンコーダ 12 で圧縮符号化して H D D 10 に供給して記録することができる。さらに、光デジタル信号が光デジタル入力端 17 から I E C 9 5 8 (International Electrotechnical Commission 958) エンコーダ 18 を介して圧縮エンコーダ 12 に供給される。光デジタル信
20 号として供給された音声信号をエンコーダ 12 で圧縮符号化して H D D 10 に記録することが可能である。

上述した例では、圧縮エンコーダ 12 は、例えば米国特許 5 7 1 7 8 2 1 に開示されているようなエンコードアルゴリズムを用いている場合を例示したが、上述した例に限定されない。すなわち、圧縮エン
25 コーダ 12 では、情報圧縮されるエンコードアルゴリズムであれば、他のものを用いることも可能である。圧縮エンコーダ 12 は、例えば

、MPEG (moving picture coding experts group)、PASC (precision adaptive sub-band coding)、TwinVQ (商標)、RealAudio (商標)、LiquidAudio (商標) といったエンコードアルゴリズムを用いるようにしてもよい。

- 5 バス40に対してモデム20が接続される。モデム20には、例えば公衆電話回線やCATV、あるいはワイヤレス通信といった外部ネットワーク19が接続される。このミュージックサーバ50は、モデム20によって外部ネットワーク19を介しての通信が可能とされる。
- 10 外部ネットワーク19を介して、ミュージックサーバ50が例えばインターネットに接続され、ミュージックサーバ50と、遠隔地のインターネットサーバ60との間で通信が行われる。ミュージックサーバ50からインターネットサーバ60に対して、リクエスト信号やCD-ROMドライブ9に装着されているCD55に関連する情報であるメディア情報、ミュージックサーバ50のそれぞれに予め与えられたユーザIDデータならびにユーザ情報、また、ユーザに対する課金情報などの各種情報が送信、送出される。

- メディア情報ユーザ情報などの各種情報がインターネットサーバ60に送信され、サーバ60は受信したユーザIDデータなどのユーザ
- 20 情報に基づき、ユーザの認証、照合処理や課金処理が行われると共に、受信したメディア情報に基づき、音楽データの付加情報が検索され、検索された付加情報がサーバ60からミュージックサーバ50に送信される。

- ここでは、音楽データの付加情報を返信する例を示したが、ユーザ
- 25 の要求に基づき、音楽データが外部ネットワーク19から直接的に供給されるようにすることも可能である。すなわち、ユーザは、ミュー

ジックサーバ50を用いてインターネットサーバ60から音楽データをダウンロードすることができる。メディア情報に対応して音楽データが返信されるようにできる。これによれば、例えば、所定のCD55のボーナストラックが配信により取得されるようにできる。

- 5 圧縮エンコーダ12により圧縮符号化されてHDD10に記録され蓄積された圧縮音楽データは、再生のためにHDD10のハードディスクから読み出されると、バス40を介して圧縮デコーダ21に供給される。HDD10のハードディスクから読み出された圧縮音楽データは、圧縮デコーダ21で圧縮符号化を解かれ、D/Aコンバータ2
- 10 2およびアンプ23を介して端子24に導出される。端子24からスピーカユニット52L, 52Rに対して供給され、音楽が再生される。なお、第2図では省略されているが、D/Aコンバータ22からアンプ23を介して端子24に到る経路は、ステレオ出力に対応して2
- 15 系統設けられる。同様に、端子24も、ステレオに対応して2つ設けられている。

- 圧縮デコーダ21では、圧縮エンコーダ12におけるエンコードアルゴリズムに対応したデコードアルゴリズムが用いられる。この圧縮デコーダ21および上述の圧縮エンコーダ12は、ハードウェアを持たずに、CPU8によるソフトウェア処理であってもよい。表示部
- 20 53を構成する液晶表示素子（以下、LCDと略称する）26がLCD駆動回路25を介してバス40に接続される。CPU8からバス40を介してLCD駆動回路25に描画制御信号が供給される。供給された描画制御信号に基づきLCD駆動回路25によってLCD26が駆動され、表示部53、すなわちLCD26上に所定の表示がなされ
- 25 る。

LCD26には、例えば、ミュージックサーバ50の操作メニュー

が表示される。LCD 26には、HDD 10に記録され蓄積された圧縮音楽データの、例えばタイトルリストが表示される。LCD 26へのタイトルリストの表示は、インターネットサーバ60から送信されてきた付加情報をデコードしたデータがHDD 10に供給されているので、HDD 10に記憶されているデータに基づいて行われる。さらに、LCD 26には、例えば選択され再生される圧縮音楽データに対応するフォルダやジャケットイメージがインターネットサーバ60から送信されてきた付加情報に基づいて表示される。

このLCD 26の表示に基づき、入力操作部1のポインティングデバイスや、キーボード2を操作することで、CPU 8は、指示された音楽データの再生制御を行う。選択された音楽データの消去や、選択された音楽データの外部の機器への複製や移動の制御も、LCD 26の表示に基づき行うことが可能である。例えば、入力操作部1がLCD 26上に設けられたタッチパネルである場合、LCD 26の表示に従いタッチパネルを触れることで、ミュージックサーバ50の操作を行うことができる。このように、LCD 26をインタフェースとして、HDD 10に記録され蓄積された音楽データがユーザにより管理ならびに制御される。

この実施の第1の形態では、ミュージックサーバ50と外部の一般的な情報機器とのインタフェースとして、IEEE 1394とPCカードに対応している。バス40に対して、IEEE 1394ドライバ29を介してIEEE 1394インタフェース28が接続される。同様に、バス40に対して、PCカードドライバ30を介してPCカードスロット31が接続される。

IEEE 1394インタフェース28によって、ミュージックサーバ50と例えばパーソナルコンピュータとの間で、データのやり取

5
10
15
20

りを行うことができる。IEEE 1394 インターフェイス 28 によって、衛星放送用の IRD (Integrated Receiver/Decoder) や、直径略 64 mm の小型の光磁気ディスクや光ディスク、DVD (Digital Versatile Disc: 商標)、デジタルビデオテープなどから音楽データを
5
10
15
20

を取り込むようにできる。PC カードスロット 31 に PC カードを装着することで、外部記憶装置やその他のメディアドライブ、あるいは、モデム、ターミナルアダプタ、キャプチャボードなどの様々な周辺機器の拡張が容易である。

10
15
20

インターフェイス 34 は、このミュージックサーバ 50 と、対応する他の記録再生装置との間で音楽データなどのやり取りを行うためのインターフェイスである。他の記録再生装置には、例えば上述の第 1 図に示される、携帯記録再生装置 70 が適用される。これに限らず、他の記録再生装置は、別のミュージックサーバ 50 であってもよい。

15
20

バス 40 に対して、インターフェイスドライバ 33 を介してインターフェイス 34 が接続される。対応する他の記録再生装置には、インターフェイス 34 と対になるインターフェイス 35 が設けられている。インターフェイス 34 および 35 とを所定の接続線 71 で電氣的に接続することで、例えば、HDD 10 に記録され蓄積された音楽データを、ミュージックサーバ 50 から他の記録再生装置に転送することができる。

25

第 3 図は、CD-ROM ドライブ 9 で読み出された音楽データが HDD 10 に記録されるまでの信号の流れを、概略的に示す。CD-ROM ドライブ 9 から読み出された音楽データは、バス 40 を介して、一旦バッファメモリとしての DRAM 11 に記憶される。DRAM 11 から音楽データが所定のタイミングで読み出され、バス 40 を介し
25

圧縮エンコーダ 12 に供給される。圧縮エンコーダ 12 は、上述し

たように、CD-ROMドライブ9の再生速度に応じた所定の圧縮速度とされている。音楽データは、圧縮エンコーダ12で圧縮符号化され、再びバッファメモリとしてのDRAM11に一旦記憶される。DRAM11から所定のタイミングで読み出された圧縮音楽データがバス40を介してHDD10に供給され、HDD10のハードディスク記録される。このとき、上述したように、インターネットサーバ60にCD-ROMドライブ9で再生されているCD55の情報を送信し、サーバ60から送信されてきたCD55の付加情報もHDD10のハードディスクに記録され、CD55から読み出された音楽データに基づく圧縮音楽データと共に、一つのデータとしてCPU8などによって管理される。

第4図は、HDD10から読み出された圧縮音楽データが再生処理されて端子24に導出されるまでの信号フローを、概略的に示す。HDD10から読み出された圧縮音楽データは、バス40を介して、バッファメモリとしてのDRAM11に一旦記憶される。そして、DRAM11から圧縮音楽データが所定のタイミングで読み出され、バス40を介して圧縮デコーダ21に供給される。圧縮音楽データは、圧縮デコーダ21で伸長処理を施されることによって圧縮符号化を解かれ、音楽データとされてD/Aコンバータ22に供給される。そして、音楽データは、D/Aコンバータ22でアナログ音声信号に変換され、アンプ23で増幅され端子24に再生出力として導出される。端子24にスピーカが接続されていれば、スピーカで再生された音楽を楽しむことができる。このとき、HDD10のディスクから読み出された付加情報は、CPU8などによって必要に応じてデコード処理され、表示部53に曲名などが表示される。

第5図は、この他の記録再生装置として用いられる、携帯記録再生

装置 70 の構成の一例を示す。この携帯記録再生装置 70 は、概ね、
上述の第 2 図に示したミュージックサーバ 50 と同等の構成を有する。
この携帯記録再生装置 70 は、通常は、ミュージックサーバ 50 側
のインターフェイス 34 と携帯記録再生装置 70 側のインターフェイ
5 ス 35 とが切り離され、単体として携帯されて用いられる。 先ず、
5 携帯記録再生装置 70 において、通常のパーソナルコンピュータ
の構成と同様に、互いにバスで結合された RAM 103, ROM 10
4, および CPU 105 とが設けられる。勿論、上述のミュージック
サーバ 50 の構成と同様に、フラッシュメモリを設けるようにしても
10 よい。マイクロコンピュータなどから構成される CPU 105 がバス
100 に接続される。CPU 105 がコントローラとして機能し、C
PU 105 によって携帯記録再生装置 70 の全体の動作が制御される
。

ROM 104 には、この携帯記録再生装置 70 の動作を制御するた
15 めのプログラムが予め記憶される。携帯記録再生装置 70 において、
このプログラムに基づき、後述する入力操作部 102 の操作に対応し
た動作がなされる。RAM 103 には、プログラムを実行する上での
データ領域、タスク領域が一時的に確保される。

入力操作部 102 は、例えば、複数のプッシュ式および回動式の操
20 作キーと、これらの操作キーによって操作される複数のスイッチから
なる。入力操作部 102 は、これに限らず、ジョグダイヤルと呼ばれ
る回動プッシュ式の操作キーや、後述する装置 70 に設けられている
LCD 上のタッチパネルなどでもかまわない。勿論、押下することで
反応する機械的なスイッチ機構を用いることもできる。この入力操作
25 部 102 の操作に応じた信号がバス 130 を介して CPU 105 に供
給される。CPU 105 は、入力操作部 102 の操作キーを操作する

ことよって発生する出力信号に基づき携帯記録再生装置 70 の動作を制御するための制御信号が生成される。携帯記録再生装置 70 は、CPU 105 で生成された制御信号に基づいて動作が切り替えられると共に動作が制御される。

- 5 ミュージックサーバ 50 において、HDD 10 から読み出され、この携帯記録再生装置 70 に対する転送を指示された音楽データは、インターフェイス 34、インターフェイス 35、およびインターフェイス 34 とインターフェイス 35 とを接続する接続線を介して、この携帯記録再生装置 70 に転送または供給される。このとき同時に、転送
10 を指定された音楽データと共に、転送を指示された音楽データの付加情報も装置 70 に送信される。また、ミュージックサーバ 50 と携帯記録再生装置 70 とに、互に対応する装着部が各々設けられている場合は、インターフェイス 34 とインターフェイス 35 とが直接的に接続され、サーバ 50 と装置 70 との間で音楽データの転送が行われ
15 る。さらに、装置 70 とサーバ 50 の双方に IrDA によるインターフェイスが設けられている場合は、赤外線信号で以てサーバ 50 と装置 70 との間で音楽データの転送が行われる。

- サーバ 50 から装置 70 に転送され供給された音楽データは、インターフェイスドライバ 101 からバス 130 を介して、この携帯記録
20 再生装置 70 の音楽データ記録媒体である HDD 106 に供給され、HDD 106 のハードディスクに記録される。

- なお、この携帯記録再生装置 70 の音楽データ記録媒体としては、HDD 106 に限らず、例えばフラッシュメモリを用いることもできる。音楽データの再生速度に追従できるものであれば、音楽データの
25 記録媒体として、例えば光磁気ディスクといった他の記録媒体を用いることもできる。装置 70 の音楽データ記録媒体としては、例えば 2

00 MByte程度の記憶容量のものを用いることによって、数10曲が記録可能である。装置70のHDD106のディスクには、サーバ50から送信されてきた音楽データと当該音楽データの付加情報も記録される。

- 5 この例では、転送されHDD106に記録される音楽データは、既にミュージックサーバ50において圧縮符号化されが圧縮音楽データである。この携帯記録再生装置70では、この例に限らず、圧縮符号化されていない音楽データを供給され、HDD106のディスクに記録することもできる。例えば、ミュージックサーバ50のCD-ROM
- 10 Mドライブ9に装着されたCD55から再生され読み出された音楽データを、インターフェイスドライバ101を介して、直接携帯記録再生装置70に供給する。但し、直接装置70に供給する場合には、記録可能な音楽データの数が大幅に制限されることはいうまでもない。

- HDD106のディスクに音楽データが記録される前処理として、
- 15 供給された音楽データは、バス130に接続されるオーディオ用のDRAM107に対して一時的に記憶される。DRAM107から読み出された音楽データがバス130を介して圧縮エンコーダ108に供給される。圧縮エンコーダ108は、ミュージックサーバ50における圧縮エンコーダ12と同等のエンコードアルゴリズムによって音楽
- 20 データの圧縮符号化処理を行う。圧縮エンコーダ108で圧縮符号化された圧縮音楽データは、DRAM107に供給され、再びDRAM107に一時的に記憶される。最終的に、このDRAM107に記憶された圧縮音楽データが読み出され、HDD106のディスクに記録される。

- 25 上述したように、ミュージックサーバ50においてHDD10に蓄積されている圧縮音楽データが移動を指示されてこの携帯記録再生装

置 70 に送信、転送されたときには、HDD 10 の圧縮音楽データは、HDD 10 上にデータとして存在するが HDD 10 から読み出して再生することのできない状態とされる。装置 70 に移動された圧縮音楽データは、再び移動元の記録媒体、すなわち、サーバ 50 の HDD 10 に戻されることで、移動元、すなわち、サーバ 50 で HDD 10 から読み出して再生することができる。このとき、移動先の記録媒体としての装置 70 の HDD 106 のハードディスクからは、サーバ 50 に戻された圧縮音楽データが削除される。

この例では、端子 109 に接続されたマイクロホンからアンプ 110 を介して入力される音声信号や、ライン入力端 111 から入力される音声信号が A/D コンバータ 112 を介して圧縮エンコーダ 108 に供給される。圧縮エンコーダ 108 で A/D コンバータ 112 から供給された音声信号に圧縮符号化処理を施して HDD 106 に記録することができる。さらに、光デジタル信号が光デジタル入力端 113 から IEC 958 エンコーダ 114 を介して圧縮エンコーダ 108 に供給される。光デジタル信号として供給された音声信号をエンコーダ 108 で圧縮符号化処理を施して HDD 106 のディスクに記録することができる。装置 70 が圧縮された音楽データを再生するのみの再生専用の携帯再生装置であれば、上述した A/D コンバータ 112、エンコーダ 108 などを全て省略することもできる。

HDD 106 から圧縮音楽データが再生のために読み出され、バス 130 を介して圧縮デコーダ 115 に供給される。圧縮デコーダ 115 で、供給された圧縮音楽データに伸長処理を施されて圧縮符号化を解かれた音楽データは、D/A コンバータ 116 およびアンプ 117 を介して端子 118 に導出される。端子 118 には、例えばヘッドホン 72 が接続される。ユーザは、このヘッドホン 72 を装着すること

によって、再生された音楽を聴くことができる。なお、第5図では省略されているが、D/Aコンバータ116からアンプ117を介して端子118に到る信号経路は、Lチャンネル、Rチャンネルのステレオ出力に対応して2系統設けられる。同様に、端子118も、L

5 ーチャンネル、Rチャンネルのステレオに対応して2つ設けられている。

LCD120がLCD駆動回路119を介してバス130に接続される。CPU105からバス130を介してLCD駆動回路119に対して描画制御信号が供給され、LCD120が供給された描画制

10 信号に基づいて駆動されてLCD120に所定の表示がなされる。LCD120には、携帯記録再生装置70の操作メニューやHDD106に記憶された音楽データのタイトルリストなどが表示される。LCD120に、例えばHDD106に記憶されている音楽データから選

15 択され再生される音楽データに対応するフォルダやジャケットイメージをHDD106に記憶されている付加情報に基づいて表示させるようにしてもよい。

このLCD120の表示に基づき、ユーザが入力操作部102のポインティングデバイスを操作することで、HDD106に記憶されている圧縮音楽データのうちの一つの圧縮音楽データが選択され、再生

20 される。選択された圧縮音楽データの消去や複製ならびに移動の制御も、LCD120の表示に基づき行うことが可能である。例えば、LCD120の表示に従い、ユーザが入力操作部102のタッチパネルを触れることで、携帯記録再生装置70の操作入力を行うことができる。

このように、LCD120をインタフェースとして、HDD106に

25 記録された圧縮音楽データがユーザにより管理ならびに記録、再生などが制御される。

なお、第5図では省略されているが、この携帯記録再生装置70は、
、バッテリで駆動される。そのため、装置70は、一般的な2次電池
や乾電池を電源供給源とする電源部が設けられると共に、充電部が設
けられる。充電部は、ミュージックサーバ50と携帯記録再生装置7
5 0とが接続線あるいは装着部によって直接的に接続される場合、音楽
データの転送と共に、ミュージックサーバ50から電力が供給され装
置70の2次電池の充電が行われる。勿論、外部の充電電源によって
装置70の2次電池の充電をするようにもできる。なお、電源の供給
源としては、乾電池による電源および2次電池を用いる充電電源の何
10 方か一方だけを用いるまたは装置70に設けるようにしてもよい。

第6図は、上述の携帯記録再生装置70の他の例を示す。なお、こ
の第6図において、上述の第5図と共通する部位に対しては同一の番
号を付し、詳細な説明を省略する。第6図に示される携帯記録再生装
置170は、上述の第5図の構成に対して、HDD（あるいはフラッ
15 シュメモリ）106aとバス130との間にスイッチ回路200が挿
入される。スイッチ回路200の一方の選択端200aがバス130
と接続され、他方の選択端200bがインターフェイス35と接続さ
れる。スイッチ回路200によって、HDD106aがバス130と
分離される。

20 ミュージックサーバ50からの圧縮音楽データ転送の際は、スイッ
チ回路200において選択端200bに切り替えまたは選択端200
bが選択される。インターフェイス34および35を介して、HDD
106aとミュージックサーバ50のバス40とが直接的に接続され
る。HDD106aは、サーバ50のCPU8から見ると、恰もミュ
25 ージックサーバ50の記録媒体であるかのように見える。ミュージ
ックサーバ50のCPU8によって、HDD106aの直接的な制御が

可能とされる。ミュージックサーバ50および携帯記録再生装置70との間での、圧縮音楽データの移動や複写などを容易に行える。

次に、上述のように構成されたシステムの動作について説明する。まず、ミュージックサーバ50単独で実行される機能について説明する。第7図は、CD-ROMドライブ9に装着されたCD55の音楽データを、ミュージックサーバ50のHDD10のディスクに記録する際の処理の一例のフローチャートである。

最初のステップS10では、ユーザによる、CD55の音楽データのHDD10への記録要求が待たれる。例えばユーザによって入力操作部1を用いて記録要求が入力されると、処理はステップS11へ移行する。ステップS11では、ユーザによって要求された記録が「高速記録」か「1倍速での記録」かが判断される。例えば、上述のステップS10で記録要求が出される際に、ユーザによって、記録の方法、すなわち、記録を高速で行うか1倍速で行うかが共に指定される。ここでいう「1倍速の記録」とは、CD55を規定されている標準速度で読み出してHDD10のディスクに記録する動作を指し、「高速記録」とは、CD55で規定されている標準速度の2倍以上の速度で読み出してHDD10のディスクに記録する動作をいう。

若し、ステップS11で、「高速記録」を行うことが指定された場合、処理はステップS12に移行し、サーバ50、60の課金システムが起動される。サーバ50、60の課金システムによる処理は、後述する。サーバ50の課金システムによる課金処理が行われ、インターネットサーバ60他の装置から高速記録が許可されると、処理はステップS13に移行し、圧縮エンコーダ12において高速圧縮処理が起動され、処理はステップS15へ移行する。

一方、ステップS11で「1倍速での記録」を行うことが指定され

た場合、処理はステップS 1 4へ移行し、圧縮エンコーダ1 2で、低速圧縮処理が起動される。処理はステップS 1 5へ移行する。ステップS 1 5では、CPU 8の制御に基づき、所定の速度で以てCD-ROMドライブ9が駆動され、CD-ROMドライブ9に装填されたCD 5 5に記録された音楽データが読み出される。読み出された音楽データは、圧縮エンコーダ1 2で圧縮符号化され、HDD 1 0のディスクに転送され記録される。

ステップS 1 6で、HDD 1 0へのCD 5 5から読み出された圧縮音楽データの転送が終了したとされたら、次のステップS 1 7でCD-ROMドライブ9からHDD 1 0へのデータの転送が禁止とされ、さらに次のステップS 1 8で圧縮エンコーダ1 2の圧縮処理が停止される。

第8図は、上述の第7図のフローチャートのステップS 1 2における課金システムの課金処理の一例を示すフローチャートである。課金処理は、ミュージックサーバ5 0とインターネットサーバ6 0との間でデータ通信が行われることによってなされる。第8図Aは、ミュージックサーバ5 0での課金処理システムでの課金処理を示し、第8図Bは、インターネットサーバ6 0での課金処理システムの課金処理を示す。

課金処理が開始されると、先ず、第8図AのステップS 2 0で、ミュージックサーバ5 0とインターネットサーバ6 0との間で、所定のプロトコルで以て通信が開始される。ステップS 2 1で、サーバ5 0とサーバ6 0との接続が確立されサーバ5 0とサーバ6 0との間で通信可能なことが確認されると、処理はステップS 2 2に移行する。

ステップS 2 2では、CD-ROMドライブ9に装着されHDD 1 0に転送し記録するCD 5 5のTOC情報がミュージックサーバ5 0

からインターネットサーバ60に対して送出される。CD55のTOC情報と共に、高速記録を行う旨を示す高速記録情報がミュージックサーバ50からインターネットサーバ60に送出される。

一方、第8図Bにおいて、インターネットサーバ60では、ミュージックサーバ50からの高速記録情報ならびにTOC情報の供給または送信されてくるのが待たれる(ステップS30)。サーバ60でこれらの高速記録情報、TOC情報が受信されたら、ステップS31で、送信されてきたTOC情報に基づいてサーバ60内のデータベース若しくは外部のデータベースを用いて、送信されてきたTOC情報の検索が行われる。TOC情報に対応する情報を検索することによりCD55が特定される。

次のステップS32で課金処理がなされる。高速記録が行われた曲数などの情報に基づいて課金する金額が算出されると共に、課金は、例えば、予め登録されたユーザのクレジットカード番号に基づき、ユーザによって指定された銀行の口座から引き落とされることで行うことができる。課金方法は、これに限らず、例えば、ミュージックサーバ50にプリペイドカードを読み取る機能を設けておき、設定された課金額がミュージックサーバ50に対して送出され、ユーザがプリペイドカードから課金された金額が減額されることによって課金額を支払うという方法も考えられる。また、TOC情報に基づき、CD55の内容によって課金額を変えたり、CD55から読み出された音楽データのHDD10のディスクへの記録を禁止することもできる。

ステップS33で、課金情報がミュージックサーバ50に対して送出される。そして、第8図Aにおいて、ミュージックサーバ50側で、送信されてきた課金情報の内容が確認がなされる(ステップS23)。インターネットサーバ60側でも、ミュージックサーバ50で課

金情報が受信されたかどうかを確認される（ステップS34）。例えば、ミュージックサーバ50側で受信された課金情報にエラーが無く、正しく受信されたことが確認されたときに、ミュージックサーバ50からサーバ60に確認済みを表すデータを送信することによって行われる。

第8図Aに戻り、ステップS23でミュージックサーバ50側で受信した課金情報が確認されると、処理はステップS24に移行し、受信された課金情報などが表示部53に表示される。ステップS25で、CD-ROMドライブ9によってCD55の規定されている標準速度よりも高速で音楽データが読み出され、圧縮エンコーダ12で高速圧縮速度で圧縮処理が行われ、圧縮エンコーダ12からの圧縮音楽データがHDD10に供給され、HDD10のディスクに記録される。このステップS25は、上述の第7図におけるステップS15に対応する。

ところで、この実施の一形態では、ミュージックサーバ50と携帯記録再生装置70との間で、連携動作が可能とされる。例えば、ミュージックサーバ50から携帯記録再生装置70に対して音楽データを移動する際には、サーバ50と装置70の間での連携動作がなされる。第9図は、この移動の一例のフローチャートを示す。

まず、最初のステップS40で、ミュージックサーバ50と携帯記録再生装置70とが、インターフェイス34および35で接続されているかどうか判断される。サーバ50と装置70との接続の検知は、例えばインターフェイス34および35との間で所定の信号のやり取りを行うことでなされる。サーバ50と装置70との接続の検知は、これに限らず、ミュージックサーバ50および携帯記録再生装置70とを接続する部分に、スイッチ機構を設け、機械的な検出機構を用

いてサーバ50と装置70との接続の検知を行うこともできる。

サーバ50と装置70との接続がステップS40で確認されると、次のステップS41で、HDD10に記録され蓄積されている音楽データの、携帯記録再生装置70への移動が要求されているかどうか
5 判断される。例えば、表示部53に対してHDD10に蓄積されている圧縮音楽データが曲名をはじめとする情報のリスト表示され、ユーザによって、入力操作部1の所定のポインティングデバイスにより、表示部53に表示されているリスト表示から所定の圧縮音楽データが
10 音楽データに対して、携帯記録再生装置70への移動の指示が入力される。

入力操作部1を用いる移動の指示の入力方法は、様々に考えられる。例えば、表示部53に移動を指示するボタンが表示され、このボタンを入力操作部1のポインティングデバイスを用いて指定することで
15 行うことができる。例えば、圧縮音楽データ毎にアイコンを表示部53に表示し、表示部53に表示されているアイコンを、やはり表示部53に表示されている移動先の携帯記録再生装置70を示すアイコン上へと移動する、所謂ドラッグ&ドロップによって行うことも可能である。勿論、入力操作部1に設けられた操作スイッチの操作により移
20 動の指示を行ってもよい。

ステップS41で圧縮音楽データの移動要求があるとされたら、ステップS42で、サーバ50側の例えばCPU8によって移動が指定された圧縮音楽データのファイルサイズ、すなわちデータ量が調べられる。次のステップS43で、例えば携帯記録再生装置70のCPU
25 105によってHDD106の空き容量、すなわち、記録可能な記憶容量が調べられる。このHDD106の空き容量と、ステップS42

で調べられた移動が指定された圧縮音楽データのファイルサイズとが例えばサーバ50のCPU8で比較される。ステップS42での比較結果に基づき、CPU8で移動が指定された圧縮音楽データがこのHDD106に記録可能であるかどうか判断される。若し、HDD106への記録が可能であるとされれば、処理はステップS45に移行し、サーバ50から装置70に向けて移動が指定された圧縮音楽データの転送が開始される。

一方、ステップS43で、携帯記録再生装置70のHDD106に空き容量が不足していると判断されれば、処理はステップS44に移行する。ステップS44では、移動が指定された圧縮音楽データのHDD106への記録が可能ないように、装置70のCPU105によって、HDD106に既に記録されている圧縮音楽データが自動的にまたは後述する手順、手法に基づいて削除され、処理はステップS45に移行する。

15 ステップS44での圧縮音楽データの削除は、HDD106に既に記録されている圧縮音楽データの、所定のパラメータに基づき、CPU105の制御のもとに自動的に行われる。例えば、携帯記録再生装置70において、HDD106に記録されている圧縮音楽データ毎に再生回数をカウントしておき、再生回数の少ないものから順にHDD20 106から削除することが考えられる。また、HDD106に記録された日付の古い順に、HDD106に記録されている圧縮音楽データを削除するようにもできる。

ステップS44でHDD106から圧縮音楽データを自動的に削除する際に、ユーザにとって重要な圧縮音楽データがHDD106から削除25 されてしまうこともあり得る。これを防止するために、ミュージックサーバ50の表示部53や携帯記録再生装置70のLCD120に、

HD 1 0 6 から自動的に圧縮音楽データが削除される動作状態になっていること、削除されるデータのリストを表示するなどの警告表示を行い、ユーザの確認を得てからHD 1 0 6 から圧縮音楽データを削除するようにもできる。ミュージックサーバ50の表示部53や携帯記録再生装置70のLCD120に対して、HDD106に既に記録されている圧縮音楽データのリストを表示させ、削除する圧縮音楽データをユーザ自身が選択するという方法もとれる。

上述のステップS43およびステップS44の処理により、HD10に記憶されている圧縮音楽データのうち移動が指定された圧縮音楽データの、HDD106への記録が可能な状態にされると、ステップS45で、ミュージックサーバ50から携帯記録再生装置70への圧縮音楽データの送信、すなわち転送が開始される。すなわち、HDD10から読み出された圧縮音楽データは、バス40ならびにインターフェイス34を介して携帯記録再生装置70に供給される。携帯記録再生装置70において、インターフェイス34を介して供給された圧縮音楽データがインターフェイス35を介してHDD106に記録される。

転送された圧縮音楽データは、ミュージックサーバ50側のHDD10にも装置70への転送前と同様に存在している。この実施の一形態では、装置70への転送済み、すなわち装置70に移動され、HDD10に存在する、該当する圧縮音楽データの再生が禁止とされる（ステップS46）。例えば、装置70への移動が完了した時点でHDD10の圧縮音楽データに対して再生禁止を示す再生禁止フラグが立てられる。この再生禁止フラグにより、サーバ50のCPU8によって装置70に移動された圧縮音楽データの再生が禁止されると共に、HDD10に記憶されている圧縮音楽データがミュージックサーバ50

から携帯記録再生装置 70 へと、仮想的に音楽データが移されたことになる。したがって、複数の圧縮音楽データのうちサーバ 50 または装置 70 で再生できる音楽データは、常に一つしか存在しないように管理され、不正な音楽データの複製が防止される。

- 5 次のステップ S 47 では、次の圧縮音楽データの装置 70 への移動要求があるかどうか判断される。若し、さらに他の圧縮音楽データの移動を行いたい場合には、処理はステップ S 42 に戻される。これ以上の音楽データの移動要求が無い場合には、一連の音楽データの移動の処理が終了される。
- 10 なお、上述では、第 9 図のフローチャートのステップ S 42 ～ステップ S 46 で HDD 10 に記憶されている複数の圧縮音楽データのうちの 1 つの圧縮音楽データを、サーバ 50 から装置 70 へ移動するように説明されているが、これに限定されず、複数の圧縮音楽データをまとめてサーバ 50 から装置 70 へ移動するようにもできる。
- 15 また、上述した例では、ステップ S 46 の処理で、移動元であるミュージックサーバ 50 の HDD 10 において、移動された圧縮音楽データは、再生禁止とされるだけで、圧縮音楽データ自身は存在はしているように説明したが、これは例に限定されず、移動された圧縮音楽データを HDD 10 から削除、すなわちデータ自身を消去するように
- 20 してもよい。
- 上述した例では、圧縮音楽データをミュージックサーバ 50 から携帯記録再生装置 70 へ移動する例について説明したが、逆方向への移動、すなわち、携帯記録再生装置 70 の HDD 106 に記録されている圧縮音楽データを、ミュージックサーバ 50 の HDD 10 へと移動
- 25 させることも、第 9 図に示したフローチャートと同様の処理に従って実行が可能である。

このとき、ミュージックサーバ50から携帯記録再生装置70へ移動した圧縮音楽データを、再び携帯記録再生装置70からミュージックサーバ50へ移動することによって、ミュージックサーバ50において、HDD10に記憶されている複数の圧縮音楽データのうち、装置70から移動されてきた圧縮音楽データの再生禁止フラグが解除される。すなわち、再生禁止フラグが解除されることによって、移動元となっている圧縮音楽データは、再びミュージックサーバ50において再生することができるようになる。この際、装置70のHDD106に記憶されていた、移動された圧縮音楽データは、データ自身をHDD106から消去するか、またはHDD106の管理テーブル上から移動された圧縮音楽データの管理データを削除される。

ところで、上述したミュージックサーバ50ならびに携帯記録再生装置70では、音楽データの、他のミュージックサーバ50や他のデジタル記録媒体に対するデジタル的な複製は、著作権保護の考えから制限され、例えば上述したように、デジタル的な複製動作が行われる都度課金が必要とされる。上述したように、ミュージックサーバ50から携帯記録再生装置70に対して、音楽データの移動は行えるが、デジタル的な複製は、上述した理由から制限される。

しかしながら、音楽データのデジタル的な複製の度に課金を行ったり、完全に音楽データのデジタル的な複製を禁じたりしてしまうと、民生用の装置としては使いにくく、ユーザにとって甚だ不親切な製品になってしまうおそれがある。そこで、本発明においては、音楽データに対して一定の制限を加えることで、音楽データの複製を許可するようにした。この実施の一形態においては、複製の際に、音楽データの質を劣化させるようにしている。

第10図A～第15図Cは、音楽データの音質劣化の方法の例を示

す。第10図A、第10図Bは、例えばデジタルフィルタを用いて、元の音楽データの周波数特性を劣化させる例である。第10図A中、実線で示されているように、第10図A中点線で示されている元の音楽データの周波数特性の高域成分を劣化させる。勿論、第10図Bの点線で示すように、元の音楽データの周波数特性の低域成分を劣化させてもよい。第11図A、第11図Bは、音楽データの量子化ビット数を劣化させる例である。例えば第11図Aに示される、元の音楽データの量子化ビット数が16ビット/サンプルであれば、第11図Bに概念的に示されるように、8ビット/サンプルに量子化ビット数を落とす。

第12図A、第12図Bは、音楽データに含まれるノイズ成分を増大させる方法の例を示す。元の音楽データには、第12図Aに示されるように、ノイズのレベルは、小音量の音楽データ、例えば最小音量の音楽データまたは再生された再生信号の最小レベルよりもさらに小さく抑えられている。これを、第12図Bに一例が示されるように、小音量の音楽データのレベルまたは音価るデータを再生した再生信号の最小レベルよりもノイズレベルが大きくなるように、ノイズ成分を増大させる。音楽データなどに元来含まれている、または存在しているノイズ成分にノイズ成分を新たに加える。第13図A、第13図Bは、元の音楽データまたは音楽データを再生した信号のピークをクリッピングする例を示す。第13図Aに実線で示される元の音楽データまたは元の音楽データを再生した再生信号の波形に対して、第13図Bに一例が示されるように、第13図B中、一点鎖線で示すような所定のレベルにスレシヨルドTHDを設け、音楽データまたは音楽データを再生した再生信号の波形を、第13図B中の実線で示すようにクリッピングする。

第14図A、第14図Bに示されている例は、元の音楽データの周波数分布を変化させる例である。例えば、元の音楽データの周波数分布が第14図Aに示されるように平坦であった場合、第14図Bに示されるように、周波数分布の低域および高域で、特性を圧縮する。

5 15図A、第15図Bおよび第15図Cに示されている例は、元の音楽データに対して、別の信号を加える方法の例である。第15図Aに示されている元の音楽データを再生した再生信号に対して、第15図Bに一例が示されるような、波形が異なる別の信号を加算し、第15図Cに示されているような波形を有する音楽データとしての信号を生成することによって、音楽データの質、例えば音楽データの再生時における音質を劣化させる。

上述に示した音質劣化の各方法は一例に過ぎず、元の音楽データを変質させ、データを劣化させるようなものであれば、他の方法を用いることもできる。例えば、元の音楽データを、より低いサンプリング周波数の音楽データに変換するという方法も考えられる。元の音楽データのダイナミックレンジを圧縮する方法も考えられる。元の音楽データに対してバンドパスフィルタをかけ、音楽データの周波数帯域を狭める方法も考えられる。ステレオの音楽データをモノラルの音楽データに変換してもよい。

20 上述したような音楽データの質を劣化させる処理は、例えばHDD10から読み出された音楽データを、ミュージックサーバ50で一旦圧縮デコーダ21に供給し圧縮符号化を解き、圧縮符号化を解かれた音楽データをCPU8に供給してソフトウェア処理を行うことによつて行われる。上述した第2図あるいは第3図、第4図に示される構成

25 にDSP (Digital Signal Processor)を追加して、ハードウェア的に元の音楽データの質を劣化させる処理を行うようにしてもよい。すな

わち、HDD 10から例えば複製のために読み出された音楽データを、圧縮デコーダ21を介して圧縮符号化を解き、追加されたDSPに供給する。追加されたDSPで所定の処理を施された音楽データがバス40を介して、例えばインターフェイス34に導出される。

- 5 上述した、音楽データの複製の際の課金処理と音質劣化処理は、選択的に行うことが可能である。第16図は、課金処理と音質劣化処理を選択的に行う処理の一例のフローチャートを示す。HDD10の記録されている音楽データを、他の記録媒体、例えば携帯記録再生装置70のHDD106のディスクに対して移動を行う際には（ステップ
- 10 S50）、課金ならびに音質劣化処理が行われず、処理はステップS54に移行し、HDD10から読み出された音楽データが例えばインターフェイス34を介して出力される。

音楽データの移動および複製処理の指示は、例えばミュージックサーバ50の入力操作部1に設けられた操作スイッチをユーザが操作す

15 ることで入力される。表示部53の表示に従い、入力操作部1によって例えばアイコンのドラッグ&ドロップを行うことによって行うこともできる。

ステップS50で音楽データの複製を行う指示が入力操作部1などを用いて入力されると、処理はステップS51に移行し、複製の方式

20 として、音質劣化処理か課金処理かが選択される。ステップS51での課金処理はまたは音質劣化処理の選択は、ユーザによって、例えばミュージックサーバの入力操作部1のスイッチ操作や、表示部53の表示に従ってなされる入力操作部1による操作がされることによって行われる。ユーザが課金処理か音質劣化処理の何れかの処理を選択す

25 る以外に、このミュージックサーバ50がインターネットサーバ60に接続されていない場合に、自動的に音質劣化処理を選択するように

もできる。

ステップ S 5 1 で課金処理が選択されると、処理はステップ S 5 2 に移行し、追加課金の手続きが行われる。すなわち、ミュージックサーバ 5 0 が通信回線 6 1 を介してインターネットサーバ 6 0 と接続される。ミュージックサーバ 5 0 とインターネットサーバ 6 0 との間でデータの授受が行われ、上述した第 8 図 B のステップ S 3 2 と同様な課金処理が行われる。ステップ S 5 3 で、音楽データの複製処理が行われる。この音楽データの複製処理は、例えば HDD 1 0 から指定された音楽データが読み出され、インターフェイス 3 4 から出力される (ステップ S 5 4)。複製元の音楽データは、HDD 1 0 上にそのまま保存されている。ステップ S 5 4 でインターフェイス 3 4 から出力された音楽データは、記録再生装置に供給され、他の記憶媒体に記録される。

一方、ステップ S 5 1 で音質劣化処理が選択されると、ステップ S 5 5 で、音質劣化処理システムが起動される。ステップ S 5 6 で、HDD 1 0 から指定の音楽データが読み出され、読み出された音楽データが上述した音質劣化方法のうちの何れかの方法によりデータ変換され、音楽データの音質が劣化するようにデータ処理が行われる。音質劣化処理による音質劣化の方法は、予め一方式に固定しておいてもよいし、複数方式または方法の中からユーザが選択できるようにしてもよい。データ変換され音質が劣化させられた音楽データは、ステップ S 5 4 で例えばインターフェイス 3 4 から出力され、上述したように、他の記録再生装置に供給され、他の記憶媒体に記憶される。

移動または複製された音楽データに対して、移動ならびに複製が行われたことを示す情報を付加してもよい。例えば、移動または複製された音楽データのヘッダ部分に、移動や複製されたデータであること

を示すフラグが付加される。一例として、例えばHDD10から移動されて移動先に存在する音楽データには、フラグ「M」が立てられ、HDD10から読み出されて複製された音楽データには、複製された音楽データのヘッダ部分にフラグ「C」が立てられる。音楽データのヘッダ部分のフラグを用いて、音楽データの移動や複製の管理を、音楽データが移動または複製されてきた記録再生装置など行うことができる。移動や複製の回数をカウントして、カウントした値をデータとして移動若しくは複製された音楽データのヘッダ部分に付加するようにしてもよい。移動や複製の回数をカウントしたデータをヘッダ部分に記録することによって、その音楽データが何回目の移動後のデータであるか、あるいは、オリジナルの音楽データに対して、何回目の複製処理動作による音楽データであるかなどを知ることができる。

移動または複製の回数をカウントした値をデータとして移動または複製された音楽データのヘッダ部分に記録することによって、音楽データの移動や複製の回数を制限することが可能となる。例えば、予め音楽データの移動の回数の上限となる回数を設定しておき、設定された回数を越えた場合、違反であると設定または判断する。音楽データのヘッダ部分に記録するデータに移動の相手先を設定、特定するためのデータを記録しておき、設定された相手以外にデータを移動した場合に、音楽データのヘッダ部分に記録されている移動の相手先を示すデータに基づいて違反とすることもできる。音楽データのヘッダ部分に記録されている移動の回数を示すデータに基づいて違反である移動を禁止したり、違反である移動に対して上述した音質劣化処理や課金処理を施すといった制御を行うことが可能とされる。

以上説明したように、本発明によれば、ミュージックサーバに記録されている音楽データを、他の記録媒体にデジタル的に複製を行う

際に、課金処理を行うようにされているため、デジタル音楽データのデジタル的な複製を、著作権を保護しながら行えるという効果がある。

また、本発明によれば、ミュージックサーバに記録されている音楽
5 データを、音質劣化させて他の記録媒体にデジタル的に複製を行うようにされているため、課金を行わなくても、デジタル音楽データの複製を、著作権を保護しながら行うことができる効果がある。

また、本発明を用いることによって、デジタル音楽データの著作権を保護しながらの複製が可能となるため、音楽データを広範に流布
10 させることが可能となり、著作者側にとっても非常に好ましい効果がある。

請求の範囲

1. 少なくともひとつのデータが記憶されている第1の記憶部と、
少なくともひとつのデータが記録される第2の記憶部と、
上記第1の記憶部から読み出されたデータが上記第2の記憶部に記
5 憶される際に上記第1の記憶部から読み出されたデータが供給され、
上記供給されたデータの劣化処理を行って上記第2の記憶部に供給す
る信号処理手段を備えている記憶再生装置。
2. 請求の範囲1に記載の記録再生装置において、
上記第1の記憶部に記憶されるデータは音楽データであるとともに
10、上記信号処理手段は上記第1の記憶部から読み出された音楽デー
タの周波数特性を劣化される処理を行うことを特徴とする記録再生装置
。
3. 請求の範囲2に記載の記録再生装置において、
上記信号処理手段は、上記第1の記憶部から読み出された音楽デー
15 タの周波数特性の高域周波数成分を劣化されることを特徴とする記録
再生装置。
4. 請求の範囲2に記載の記録再生装置において、
上記信号処理手段は、上記第1の記憶部から読み出された音楽デー
タの周波数特性の低域周波数成分を劣化させることを特徴とする記録
20 再生装置。
5. 請求の範囲1に記載の記録再生装置において、
上記信号処理手段は、上記第1の記憶部から読み出されたデータの
量子化ビット数を劣化させる処理を行うことを特徴とする記録再生装
置。
- 25 6. 請求の範囲1に記載の記録再生装置において、
上記信号処理手段は、上記第1の記憶部から読み出されたデータに

ノイズ成分を付加する処理を行わせることを特徴とする記録再生装置
。

7. 請求の範囲6に記載の記録再生装置において、

上記信号処理手段によって付加されるノイズ成分は、最小音量の音楽データのレベルよりも大きいレベルを有するものであることを特徴とする記録再生装置。

8. 請求の範囲1に記載の記録再生装置において、

上記信号処理手段は、上記第1の記憶部から読み出されるデータを再生した信号をクリッピングする処理を行うことを特徴とする記録再生装置。

9. 請求の範囲1に記載の記録再生装置において、

上記第1の記憶部に記憶されるデータは音楽データであるとともに、上記信号処理手段は、上記第1の記憶部から読み出されたデータの周波数分布を変化させる処理を行うことを特徴とする記録再生装置。

10. 請求の範囲1に記載の記録再生装置において、

上記信号処理手段は、上記第1の記憶部から読み出されたデータに他のデータを付加する処理を行うことを特徴とする記録再生装置。

11. 請求の範囲10に記載の記録再生装置において、

上記信号処理手段は、上記第1の記憶部から読み出されたデータを再生した再生信号に、上記再生信号とは異なる波形を有する信号を加算する処理を行うことを特徴とする記録再生装置。

12. 請求の範囲1に記載の記録再生装置において、

上記信号処理手段は、上記第1の記憶部に記憶されているデータが読み出されて上記第2の記憶部に複製される際に上記データの劣化処理を行うことを特徴とする記録再生装置。

13. 第1の記憶部から読み出されたデータが第2の記憶部に記憶さ

れる際に、上記第 1 の記憶部から読み出されたデータにデータの劣化処理を施して上記第 2 の記憶部に供給し、上記第 2 の記憶部に書き込まれる記録再生方法。

1 4. 請求の範囲 1 3 に記載の記録再生方法において、

- 5 上記第 1 の記憶部に記憶されるデータは音楽データであるとともに、上記データの劣化処理は、上記第 1 の記憶部から読み出された音楽データの周波数特性を劣化させる処理を行うことを特徴とする記録再生方法。

1 5. 請求の範囲 1 4 に記載の記録再生方法において、

- 10 上記データの劣化処理は、上記第 1 の記憶部から読み出された音楽データの周波数特性の高域周波数成分を劣化させることを特徴とする記録再生方法。

1 6. 請求の範囲 1 4 に記載の記録再生方法において、

- 15 上記データの劣化処理は、上記第 1 の記憶部から読み出された音楽データの周波数特性の低域周波数成分を劣化させることを特徴とする記録再生方法。

1 7. 請求の範囲 1 3 に記載の記録再生方法において、

- 20 上記データの劣化処理は、上記第 1 の記憶部から読み出されたデータの量子化ビット数を劣化させる処理を行うことを特徴とする記録再生方法。

1 8. 請求の範囲 1 3 に記載の記録再生方法において、

上記データの劣化処理は、上記第 1 の記憶部から読み出されたデータにノイズ成分を付加する処理を行わせることを特徴とする記録再生方法。

- 25 1 9. 請求の範囲 1 8 に記載の記録再生方法において、

上記データの劣化処理によって付加されるノイズ成分は、最小音量

の音楽データのレベルよりも大きいレベルを有するものであることを特徴とする記録再生方法。

20. 請求の範囲13に記載の記録再生方法において、

上記データの劣化処理は、上記第1の記憶部から読み出されるデータを再生した信号をクリッピングする処理を施すことを特徴とする記録再生方法。

21. 請求の範囲13に記載の記録再生方法において、

上記第1の記憶部に記憶されるデータは音楽データであるとともに、上記データの劣化処理は、上記第1の記憶部から読み出されたデータの周波数分布を変化させる処理を行うことを特徴とする記録再生方法。

22. 請求の範囲13に記載の記録再生方法において、

上記データの劣化処理は、上記第1の記憶部から読み出されたデータに他のデータを付加する処理を行うことを特徴とする記録再生方法。

23. 請求の範囲22に記載の記録再生方法において、

上記データの劣化処理は、上記第1の記憶部から読み出されたデータを再生した再生信号に、上記再生信号とは異なる波形を有する信号を加算する処理を行うことを特徴とする記録再生方法。

24. 請求の範囲13に記載の記録再生方法において、

上記方法は、上記第1の記憶部に記憶されているデータが読み出されて上記第2の記憶部に複製される際に上記データの劣化処理が行われることを特徴とする記録再生方法。

25. 少なくともひとつのデータが記憶されている第1の記憶部と、

少なくともひとつのデータが記録される第2の記憶部と、

上記第1の記憶部から読み出されたデータが供給されるとともに、

上記第 1 の記憶部から読み出されたデータが上記第 2 の記憶部に記憶される際に課金処理と上記第 1 の記憶部から供給されたデータの劣化処理のうちのいずれ一方の処理を行って上記第 2 の記憶部に供給する制御手段を備えている記録再生装置。

5 26. 請求の範囲 25 に記載の記録再生装置において、

上記制御手段は、上記課金処理と上記データの劣化処理のうち選択された処理を実行することを特徴とする記録再生装置。

27. 請求の範囲 25 に記載の記録再生装置において、

10 上記制御手段は、上記第 1 の記憶部から読み出されたデータが上記第 2 の記憶部に複製される際に課金処理と上記データの劣化処理のうちのいずれか一方の処理を行うことを特徴とする記録再生装置。

28. 請求の範囲 27 に記載の記録再生装置において、

15 上記制御手段は、上記第 1 の記憶部から読み出されたデータが上記第 2 の記憶部に記憶され、上記第 1 の記憶部で上記第 2 の記憶部に供給されて記憶されたデータの再生が禁止されるときには上記課金処理及び上記データの劣化処理の何れも実行しないことを特徴とする記録再生装置。

29. 請求の範囲 25 に記載の記録再生装置において、

20 上記第 1 の記憶部に記憶されるデータは音楽データであるとともに、上記制御手段は、上記第 1 の記憶部から読み出された音楽データの周波数特性を劣化させる処理を行うことを特徴とする記録再生装置。

30. 請求の範囲 29 に記載の記録再生装置において、

25 上記制御手段は、上記第 1 の記憶部から読み出された音楽データの周波数特性の高域周波数成分を劣化させることを特徴とする記録再生装置。

31. 請求の範囲 29 に記載の記録再生装置において、

上記制御手段は、上記第 1 の記憶部から読み出された音楽データの周波数特性の低域周波数成分を劣化させることを特徴とする記録再生装置。

3 2. 請求の範囲 2 5 に記載の記録再生装置において、

- 5 上記制御手段は、上記第 1 の記憶部から読み出されたデータの量子化ビット数を劣化させる処理を行うことを特徴とする記録再生装置。

3 3. 請求の範囲 2 5 に記載の記録再生装置において、

上記制御手段は、上記第 1 の記憶部から読み出されたデータにノイズ成分を付加する処理を行わせることを特徴とする記録再生装置。

- 10 3 4. 請求の範囲 3 3 に記載の記録再生装置において、

上記制御手段によって付加されるノイズ成分は、最小音量の音楽データのレベルよりも大きいレベルを有するものであることを特徴とする記録再生装置。

3 5. 請求の範囲 3 3 に記載の記録再生装置において、

- 15 上記制御手段は、上記第 1 の記憶部から読み出されるデータを再生した信号をクリッピングする処理を施すことを特徴とする記録再生装置。

3 6. 請求の範囲 3 3 に記載の記録再生装置において、

- 20 上記第 1 の記憶部に記憶されるデータは音楽データであるとともに、上記制御手段は、上記第 1 の記憶部から読み出されたデータの周波数分布を変化させる処理を行うことを特徴とする記録再生装置。

3 7. 請求の範囲 3 3 に記載の記録再生装置において、

上記制御手段は、上記第 1 の記憶部から読み出されたデータに他のデータを付加する処理を行うことを特徴とする記録再生装置。

- 25 3 8. 請求の範囲 3 7 に記載の記録再生装置において、

上記制御手段は、上記第 1 の記憶部から読み出されたデータを再生

した再生信号に、上記再生信号とは異なる波形を有する信号を加算する処理を行うことを特徴とする記録再生装置。

39. 第1の記憶部から読み出されたデータが第2の記憶部に記憶される際に課金処理と上記第1の記憶部から供給されたデータの劣化処理のうちのいずれか一方の処理を行って上記第2の記憶部に供給する記録再生方法。

40. 請求の範囲39に記載の記録再生方法において、

上記方法は、上記課金処理と上記データの劣化処理のうち選択された処理を実行することを特徴とする記録再生方法。

10 41. 請求の範囲39に記載の記録再生方法において、

上記方法は、上記第1の記憶部から読み出されたデータが上記第2の記憶部に複製される際に上記課金処理と上記データの劣化処理のうちのいずれか一方の処理を行うことを特徴とする記録再生方法。

42. 請求の範囲42に記載の記録再生方法において、

15 上記方法は、上記第1の記憶部から読み出されたデータが上記第2の記憶部に記憶され、上記第1の記憶部で上記第2の記憶部に供給されて記憶されたデータの再生が禁止されるときには上記課金処理及び上記データの劣化処理の何れも実行しないことを特徴とする記録再生方法。

20 43. 少なくともひとつのデータが記憶されている記憶部と、

上記記憶部から読み出されたデータが他の記憶媒体に複製される際に記憶される際に上記記憶部から読み出されたデータが供給され、上記供給されたデータの劣化処理を行い、上記データの劣化処理が施されたデータを出力データとして出力する信号処理手段を備えているデータ処理装置。

44. 請求の範囲43に記載のデータ処理装置において、

上記記憶部に記憶されるデータは音楽データであるとともに、上記信号処理手段は上記記憶部から読み出された音楽データの周波数特性を劣化させる処理を行うことを特徴とするデータ処理装置。

4 5. 請求の範囲 4 4 に記載のデータ処理装置において、

- 5 上記信号処理手段は、上記記憶部から読み出された音楽データの周波数特性の高域周波数成分を劣化させることを特徴とするデータ処理装置。

4 6. 請求の範囲 4 4 に記載のデータ処理装置において、

- 10 上記信号処理手段は、上記記憶部から読み出された音楽データの周波数特性の低域周波数成分を劣化させることを特徴とするデータ処理装置。

4 7. 請求の範囲 4 4 に記載のデータ処理装置において、

上記信号処理手段は、上記記憶部から読み出されたデータの量子化ビット数を劣化させる処理を行うことを特徴とするデータ処理装置。

- 15 4 8. 請求の範囲 4 4 に記載のデータ処理装置において、

上記信号処理手段は、上記記憶部から読み出されたデータにノイズ成分を付加する処理を行わせることを特徴とするデータ処理装置。

4 9. 請求の範囲 4 8 に記載のデータ処理装置において、

- 20 上記信号処理手段によって付加されるノイズ成分は、最小音量の音楽データのレベルよりも大きいレベルを有するものであることを特徴とするデータ処理装置。

5 0. 請求の範囲 4 4 に記載のデータ処理装置において、

- 25 上記信号処理手段は、上記記憶部から読み出されるデータを再生した信号をクリッピングする処理を行うことを特徴とするデータ処理装置。

5 1. 請求の範囲 4 4 に記載のデータ処理装置において、

上記記憶部に記憶されるデータは音楽データであるとともに、上記信号処理手段は、上記記憶部から読み出されたデータの周波数分布を変化させる処理を行うことを特徴とするデータ処理装置。

5 2. 請求の範囲 4 4 に記載のデータ処理装置において、

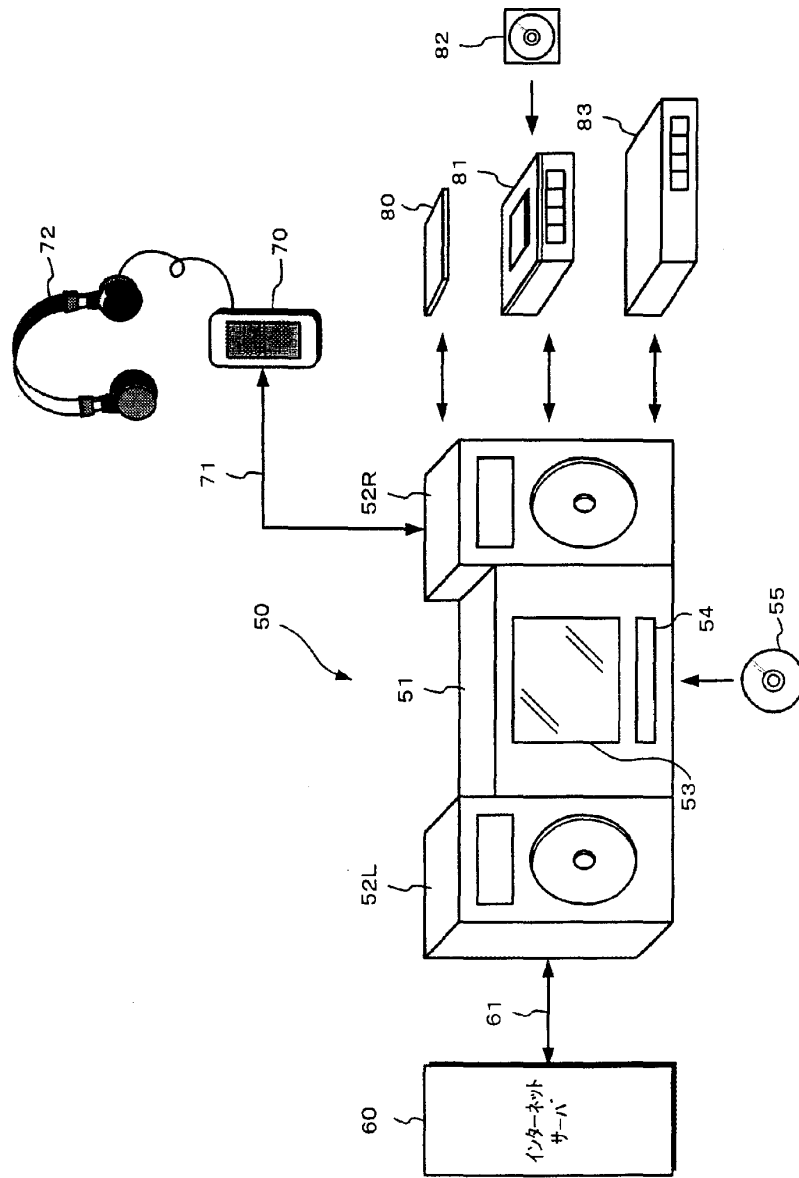
- 5 上記信号処理手段は、上記記憶部から読み出されたデータに他のデータを付加する処理を行うことを特徴とするデータ処理装置。

5 3. 請求の範囲 5 2 に記載のデータ処理装置において、

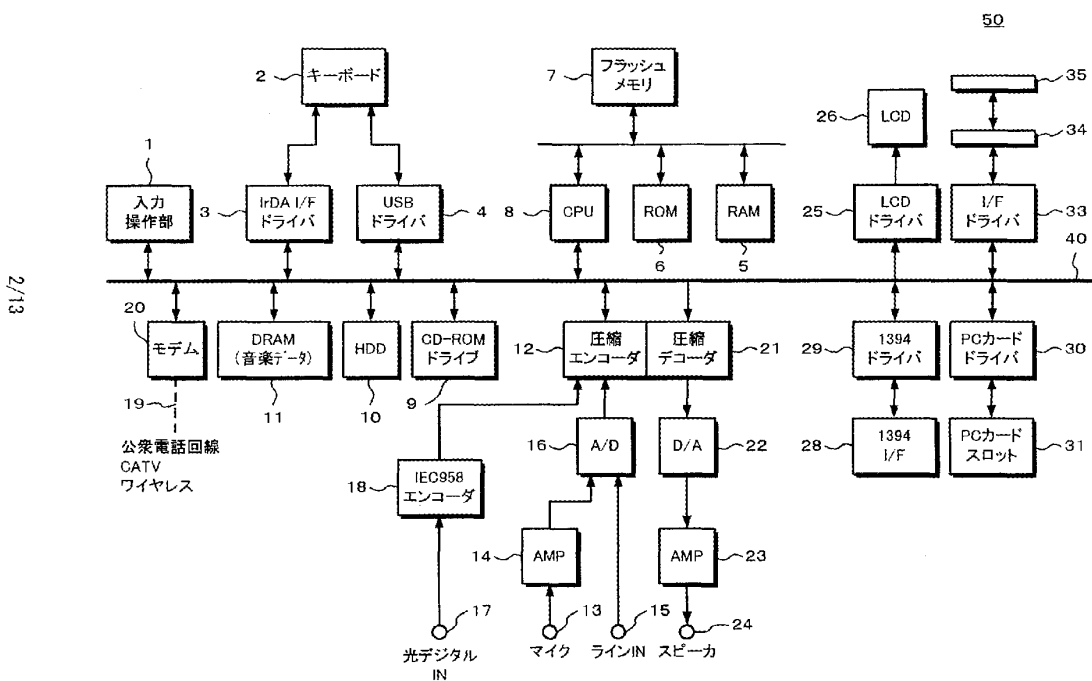
上記信号処理手段は、上記記憶部から読み出されたデータを再生した再生信号に、上記再生信号とは異なる波形を有する信号を加算する

- 10 処理を行うことを特徴とするデータ処理装置。

第1図



第2図

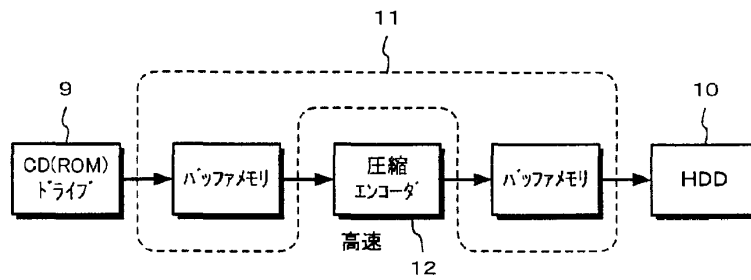


2/13

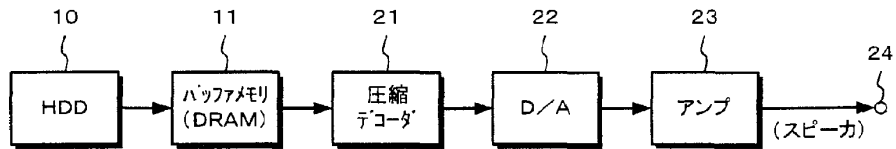
WO 99/42996

PCT/JP99/00702

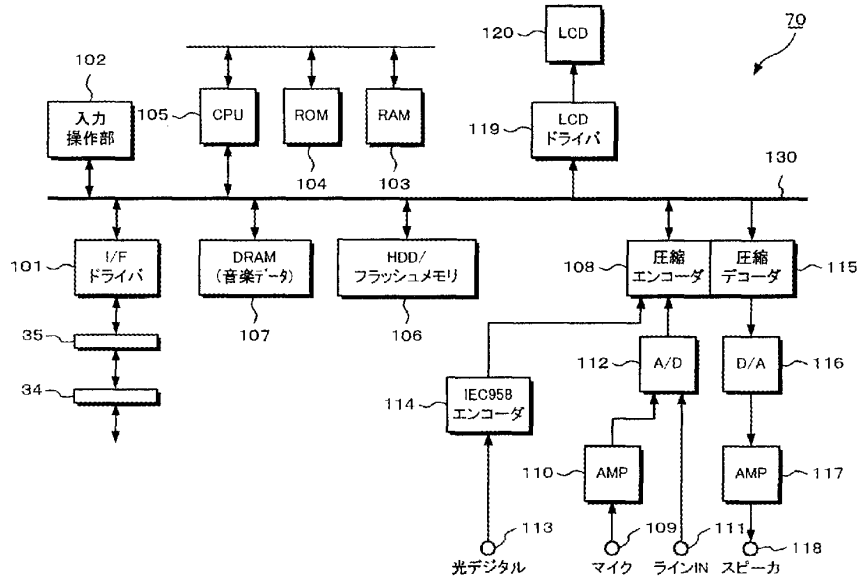
第3図



第4図



第5図

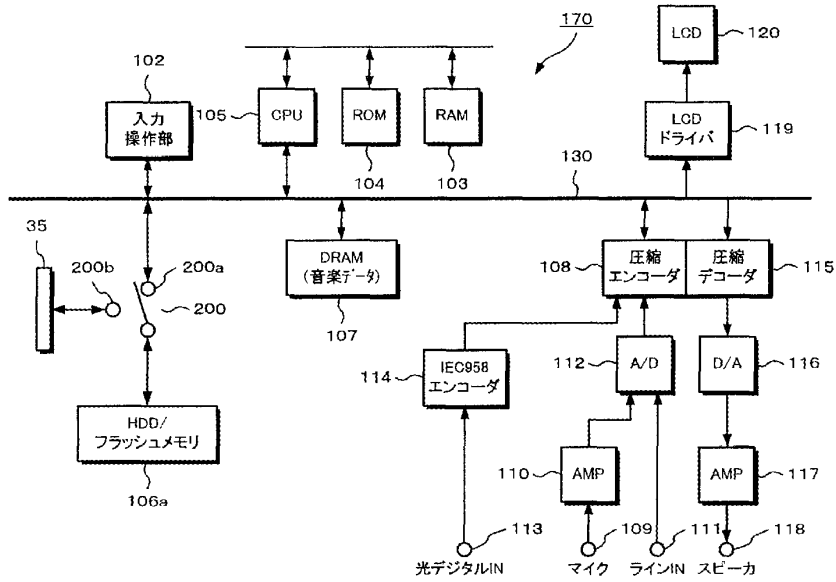


4/13

WO 99/42996

PCT/JP99/00702

第6図

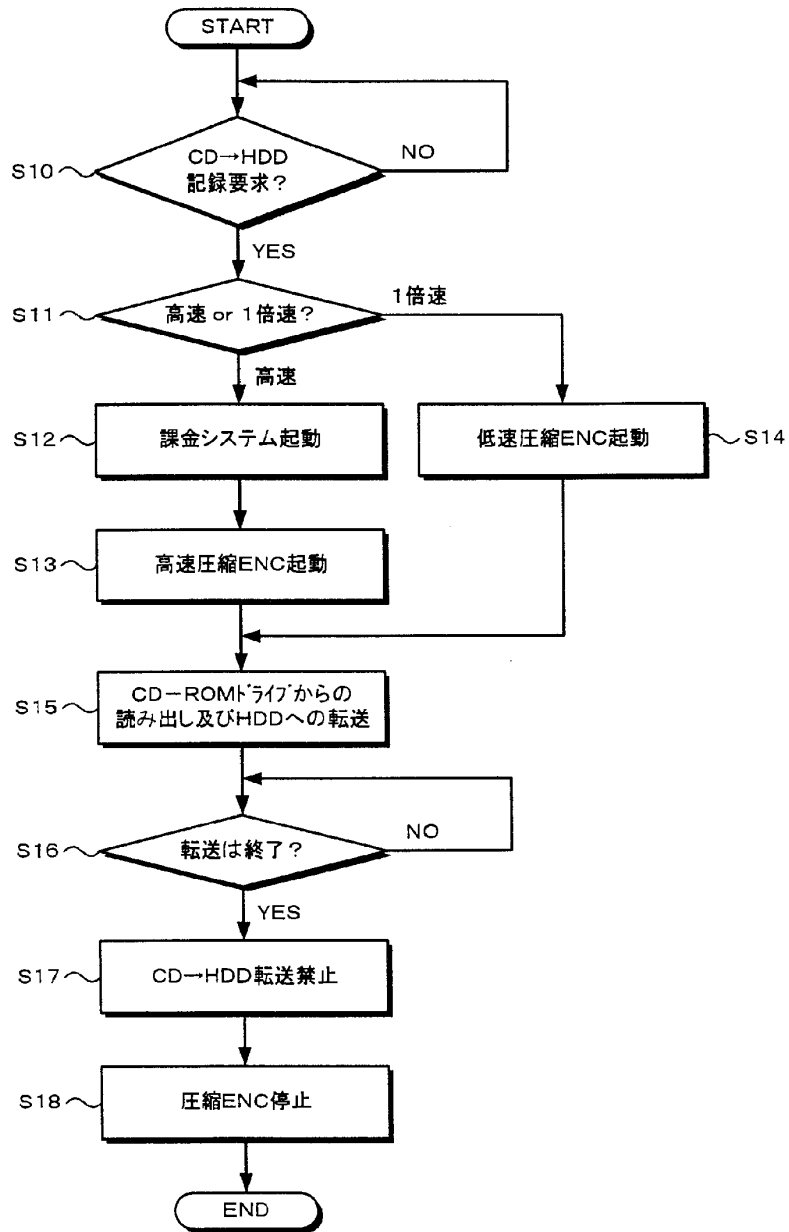


5/13

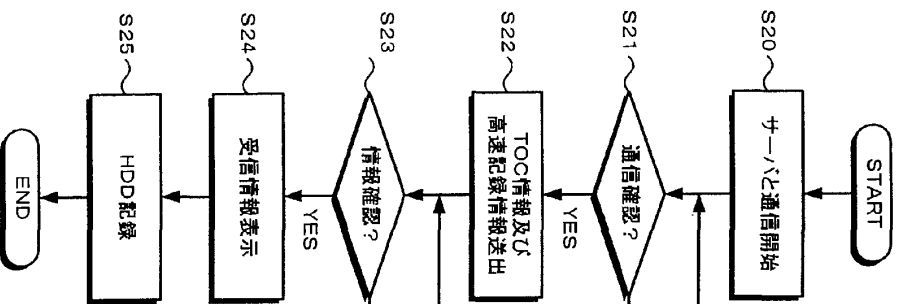
WO 99/42996

PC11JP99/00702

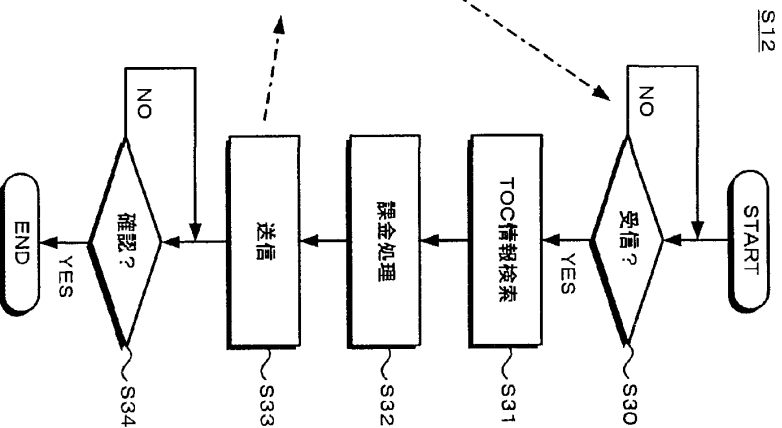
第7図



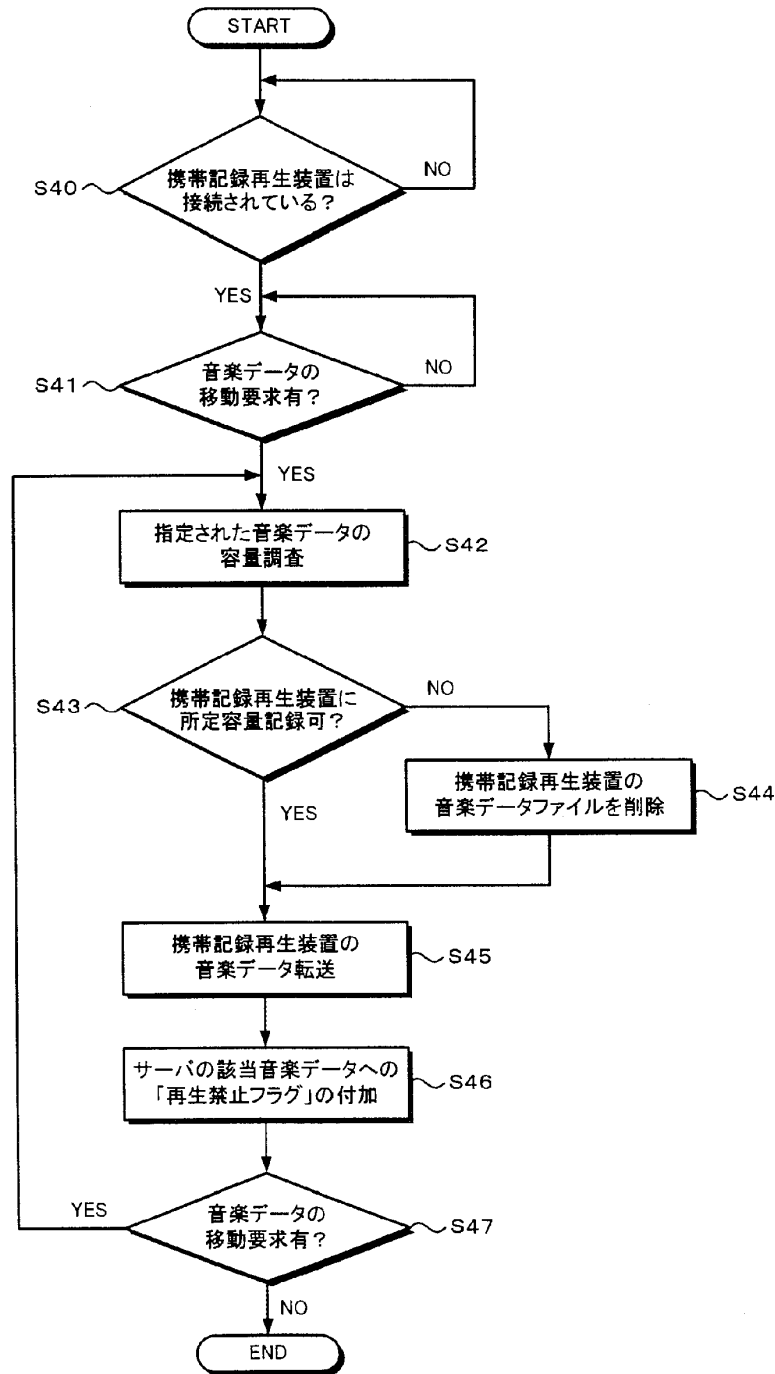
第8図A



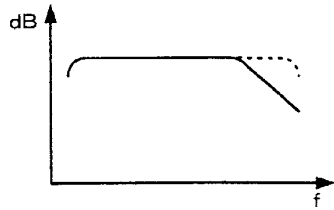
第8図B



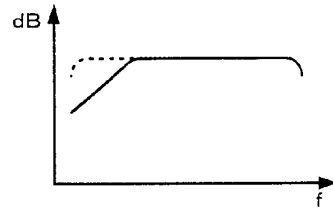
第9図



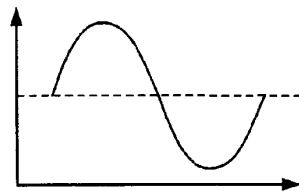
第10図A



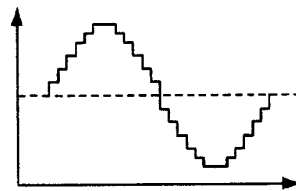
第10図B



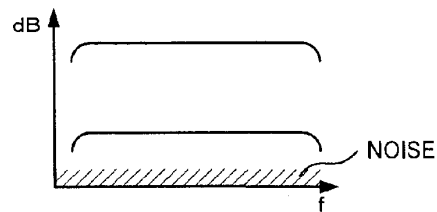
第11図A



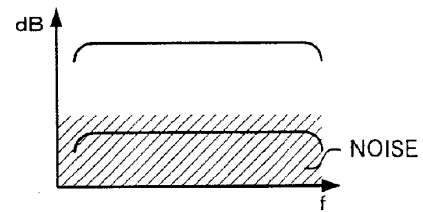
第11図B



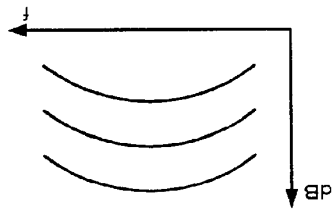
第12図A



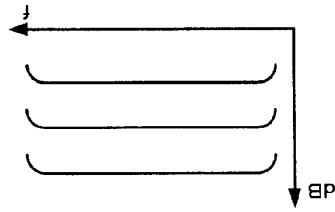
第12図B



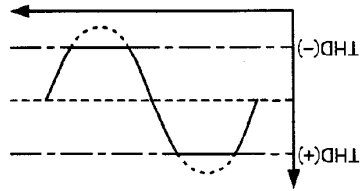
10/13



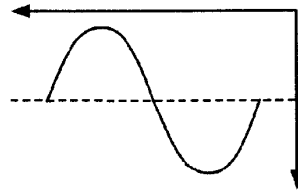
第 1 4 图 B



第 1 4 图 A

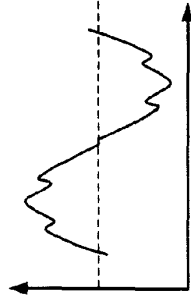


第 1 3 图 B

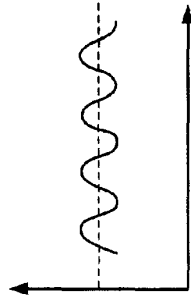


第 1 3 图 A

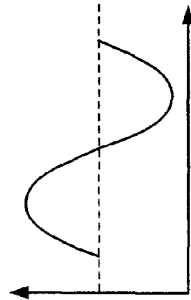
第15図C



第15図B



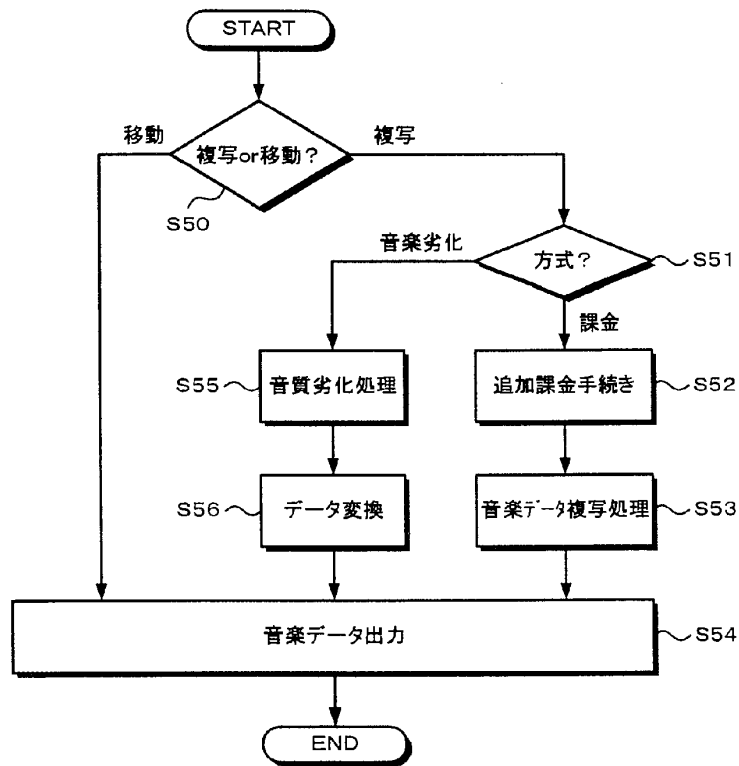
第15図A



+



第 1 6 図




- 1 ミュージックサーバの入力手段
- 8 ミュージックサーバのCPU
- 9 CD-ROMドライブ
- 10 ミュージックサーバのハードディスクドライブ
- 11 ミュージックサーバのバッファメモリとしてのDRAM
- 12 ミュージックサーバの圧縮エンコーダ
- 19 通信回線
- 20 モデム
- 21 ミュージックサーバの圧縮デコーダ
- 26 ミュージックサーバのLCD
- 34, 35 インターフェイス
- 40 バス
- 50 ミュージックサーバ
- 55 CD
- 60 インターネットサーバ
- 70 携帯記録再生装置
- 106 携帯記録再生装置のハードディスクドライブあるいはフラッシュRAM
- 107 携帯記録再生装置のバッファメモリとしてのDRAM
- 108 携帯記録再生装置の圧縮エンコーダ
- 115 携帯記録再生装置の圧縮デコーダ
- 120 携帯記録再生装置のLCD
- 130 携帯記録再生装置のバス
- 200 スイッチ回路

INTERNATIONAL SEARCH REPORT

International application No.
PCT/JP99/00702

<p>A. CLASSIFICATION OF SUBJECT MATTER Int.Cl⁶ G11B20/10</p> <p>According to International Patent Classification (IPC) or to both national classification and IPC</p>																	
<p>B. FIELDS SEARCHED</p> <p>Minimum documentation searched (classification system followed by classification symbols) Int.Cl⁶ G11B20/10</p> <p>Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched Jitsuyo Shinan Koho 1922-1999 Toroku Jitsuyo Shinan Koho 1994-1999 Kokai Jitsuyo Shinan Koho 1971-1999 Jitsuyo Shinan Toroku Koho 1996-1999</p> <p>Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)</p>																	
<p>C. DOCUMENTS CONSIDERED TO BE RELEVANT</p> <table border="1"> <thead> <tr> <th>Category*</th> <th>Citation of document, with indication, where appropriate, of the relevant passages</th> <th>Relevant to claim No.</th> </tr> </thead> <tbody> <tr> <td>X</td> <td>JP, 9-106624, A (Fujitsu Ten Ltd.), 22 April, 1997 (22. 04. 97), Full text ; Figs. 1 to 3</td> <td>1, 12, 24, 43</td> </tr> <tr> <td>Y</td> <td>Full text ; Figs. 1 to 3</td> <td>2-11, 14-23, 44-53</td> </tr> <tr> <td>A</td> <td>Full text ; Figs. 1 to 3 (Family: none)</td> <td>25-42</td> </tr> <tr> <td>A</td> <td>JP, 9-265731, A (Sony Corp.), 7 October, 1997 (07. 10. 97), Full text ; Figs. 1 to 9 (Family: none)</td> <td>25-42</td> </tr> </tbody> </table>			Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.	X	JP, 9-106624, A (Fujitsu Ten Ltd.), 22 April, 1997 (22. 04. 97), Full text ; Figs. 1 to 3	1, 12, 24, 43	Y	Full text ; Figs. 1 to 3	2-11, 14-23, 44-53	A	Full text ; Figs. 1 to 3 (Family: none)	25-42	A	JP, 9-265731, A (Sony Corp.), 7 October, 1997 (07. 10. 97), Full text ; Figs. 1 to 9 (Family: none)	25-42
Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.															
X	JP, 9-106624, A (Fujitsu Ten Ltd.), 22 April, 1997 (22. 04. 97), Full text ; Figs. 1 to 3	1, 12, 24, 43															
Y	Full text ; Figs. 1 to 3	2-11, 14-23, 44-53															
A	Full text ; Figs. 1 to 3 (Family: none)	25-42															
A	JP, 9-265731, A (Sony Corp.), 7 October, 1997 (07. 10. 97), Full text ; Figs. 1 to 9 (Family: none)	25-42															
<p><input type="checkbox"/> Further documents are listed in the continuation of Box C. <input type="checkbox"/> See patent family annex.</p>																	
<p>* 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 "I" 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</p>		<p>"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</p>															
<p>Date of the actual completion of the international search 4 March, 1999 (04. 03. 99)</p>		<p>Date of mailing of the international search report 16 March, 1999 (16. 03. 99)</p>															
<p>Name and mailing address of the ISA/ Japanese Patent Office</p>		<p>Authorized officer</p>															
<p>Facsimile No.</p>		<p>Telephone No.</p>															

Form PCT/ISA/210 (second sheet) (July 1992)

A. 発明の属する分野の分類 (国際特許分類 (IPC)) Int. Cl ⁸ G11B20/10		
B. 調査を行った分野 調査を行った最小限資料 (国際特許分類 (IPC)) Int. Cl ⁸ G11B20/10		
最小限資料以外の資料で調査を行った分野に含まれるもの 日本国実用新案公報 1922-1999年 日本国公開実用新案公報 1971-1999年 日本国登録実用新案公報 1994-1999年 日本国実用新案登録公報 1996-1999年		
国際調査で使用した電子データベース (データベースの名称、調査に使用した用語)		
C. 関連すると認められる文献		
引用文献の カテゴリー*	引用文献名 及び一部の箇所が関連するときは、その関連する箇所の表示	関連する 請求の範囲の番号
X	J P, 9-106624, A (富士通テン株式会社) 22. 4月. 1997 (22. 04. 97) 全文, 第1-3図	1, 12, 2 4, 43
Y	全文, 第1-3図	2-11, 1 4-23, 4 4-53
A	全文, 第1-3図 (ファミリーなし)	25-42
A	J P, 9-265731, A (ソニー株式会社) 7. 10月. 1997 (07. 10. 97) 全文, 第1-9図 (ファミリーなし)	25-42
<input type="checkbox"/> C欄の続きにも文献が列挙されている。 <input type="checkbox"/> パテントファミリーに関する別紙を参照。		
* 引用文献のカテゴリー 「A」 特に関連のある文献ではなく、一般的技術水準を示すもの 「E」 国際出願日前の出願または特許であるが、国際出願日以後に公表されたもの 「L」 優先権主張に疑義を提起する文献又は他の文献の発行日若しくは他の特別な理由を確立するために引用する文献 (理由を付す) 「O」 口頭による開示、使用、展示等に言及する文献 「P」 国際出願日前で、かつ優先権の主張の基礎となる出願日の後に公表された文献 「T」 国際出願日又は優先日後に公表された文献であって出願と矛盾するものではなく、発明の原理又は理論の理解のために引用するもの 「X」 特に関連のある文献であって、当該文献のみで発明の新規性又は進歩性がないと考えられるもの 「Y」 特に関連のある文献であって、当該文献と他の1以上の文献との、当業者にとって自明である組合せによって進歩性がないと考えられるもの 「&」 同一パテントファミリー文献		
国際調査を完了した日	04. 03. 99	国際調査報告の発送日 16.03.99
国際調査機関の名称及びあて先 日本国特許庁 (ISA/J P) 郵便番号100-8915 東京都千代田区霞が関三丁目4番3号	特許庁審査官 (権限のある職員) 小松 正 電話番号 03-3581-1101 内線 6922	5D 7736 

様式PCT/ISA/210 (第2ページ) (1998年7月)

(19)日本国特許庁(J P)

(12) 公開特許公報(A)

(11)特許出願公開番号

特開平5-334072

(43)公開日 平成5年(1993)12月17日

(51)Int.Cl.⁵

識別記号 庁内整理番号

F I

技術表示箇所

G 0 6 F 9/06

4 5 0 C 7232-5B

審査請求 有 請求項の数28(全 20 頁)

(21)出願番号 特願平3-308350

(22)出願日 平成3年(1991)10月29日

(31)優先権主張番号 6 2 9 2 9 5

(32)優先日 1990年12月14日

(33)優先権主張国 米国(U S)

(71)出願人 390009531

インターナショナル・ビジネス・マシーンズ・コーポレーション

INTERNATIONAL BUSINESS MACHINES CORPORATION

アメリカ合衆国10504、ニューヨーク州アーモンク (番地なし)

(72)発明者 ロバート・カール・ビーチャー

アメリカ合衆国55904、ミネソタ州ロチェスター、17番ストリート、サウス・イースト 910番地

(74)代理人 弁理士 頼宮 孝一 (外4名)

最終頁に続く

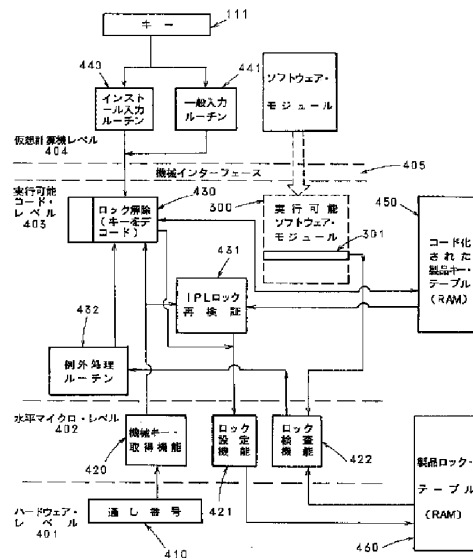
(54)【発明の名称】 ソフトウェアの使用を管理するための装置及び方法

(57)【要約】

【目的】本発明の目的は、コンピュータ・システムにおけるソフトウェアの使用を管理する改善された方法及び装置を提供することにある。

【構成】ソフトウェアは、実行する資格付与をなして配布され、別途配布される暗号化された資格付与キーによって、はじめてそのソフトウェアの実行が可能になる。このキーは、ソフトウェアのライセンスが与えられるコンピュータの通し番号と、どのソフトウェア・モジュールが機械上で走行する資格を付与されているかを示す、複数の資格付与ビットを含んでいる。配布されたソフトウェアは、複数の資格検証トリガを含む。各トリガは、ソフトウェア・モジュールの製品番号を識別する、目的コード形式の単一の機械語命令である。

【効果】本発明によって、コンピュータ・システムにおけるソフトウェアの使用を管理する、改善された方法及び装置が提供される。



【特許請求の範囲】

【請求項1】コンピュータ・システムにソフトウェア・モジュールを実行する資格を付与する手段と、上記ソフトウェア・モジュール内に配置された、資格の検証をトリガする複数の独立したトリガ手段と、上記の複数の独立したトリガ手段のそれぞれに 대응して、上記コンピュータ・システムが上記ソフトウェア・モジュールを実行する資格をもつことを検証する資格検証手段と、

上記の資格検証手段に 대응して、上記コンピュータ・システムが上記のソフトウェア・モジュールを実行する資格をもたないと上記資格検証手段が判定した場合に、上記ソフトウェア・モジュールの実行を打ち切る手段とを備える、コンピュータ・システムで実行されるソフトウェア・モジュールの使用を管理するための装置。

【請求項2】上記の複数の独立したトリガ手段が、上記資格検証手段をトリガする単一の目的コード命令である、請求項1に記載の、ソフトウェア・モジュールの使用を管理するための装置。

【請求項3】上記資格検証手段をトリガする上記目的コード命令が、上記ソフトウェア・モジュールの適切な実行に必要な追加ステップをも実行し、その結果、上記目的コード命令を除去することによって上記ソフトウェア・モジュールが修正される場合は、上記の追加ステップが実行されず、上記ソフトウェア・モジュールが適切に実行されなくなる、請求項2に記載の、ソフトウェア・モジュールの使用を管理するための装置。

【請求項4】上記資格検証手段をトリガする上記目的コード命令が、上記のソフトウェア・モジュールの製品番号を含む、請求項2に記載の、ソフトウェア・モジュールの使用を管理するための装置。

【請求項5】上記資格検証手段が、資格付与情報を含みかつそれぞれが製品番号に関連する複数のエントリを備える、上記コンピュータ・システム内の製品ロック・テーブルと、上記目的コード命令に 対応して、上記目的コード命令に含まれる上記製品番号に関連する上記製品ロック・テーブル中のエントリに含まれる上記資格付与情報にアクセスする手段とを含む、請求項4に記載の、ソフトウェア・モジュールの使用を管理するための装置。

【請求項6】上記のコンピュータ・システムに上記のソフトウェア・モジュールを実行する資格を付与する上記手段が、

データから構成される資格付与キーを生成する手段と、上記の資格付与キーを上記コンピュータ・システムに入力する手段とを備える、請求項1に記載の、ソフトウェア・モジュールの使用を管理するための装置。

【請求項7】上記コンピュータ・システムが独自の識別子を含み、

資格付与キーを生成する上記手段が、上記の独自の識別

子を用いて上記資格付与キーを生成し、上記資格付与キーが、同じ独自の識別子を含むコンピュータ・システム上でのみソフトウェアを実行する資格を与える、請求項6に記載の、ソフトウェア・モジュールの使用を管理するための装置。

【請求項8】上記の独自の識別子を用いて上記資格付与キーを暗号化し、その結果得られる暗号化された資格付与キーが同じ独自の識別子をもつコンピュータ・システム上でしか解読できないようにする手段と、上記コンピュータ・システムの独自の識別子にアクセスする、コンピュータ・システム内の手段と、上記の独自の識別子にアクセスする上記手段に 対応して、上記の暗号化された資格付与キーを解読する、上記のコンピュータ・システム内の手段とを備える、請求項7に記載の、ソフトウェア・モジュールの使用を管理するための装置。

【請求項9】資格の検証をトリガする複数の独立したトリガ手段をソフトウェア・モジュール内に配置する段階と、

上記ソフトウェア・モジュールを実行する段階と、上記ソフトウェア・モジュールの実行中に上記複数の独立したトリガ手段の一つに出会ったとき、コンピュータ・システムがそのソフトウェア・モジュールを実行する資格をもつことを検証する資格検証動作を上記コンピュータ・システム中でトリガする段階と、

上記資格検証で、上記コンピュータ・システムには上記ソフトウェア・モジュールを実行する資格がないと判定された場合、上記ソフトウェア・モジュールの実行を打ち切る段階とを含む、コンピュータ・システム上で実行されるソフトウェア・モジュールの使用を管理するための方法。

【請求項10】複数の独立したトリガ手段を上記ソフトウェア・モジュール内に配置する上記段階が、上記ソフトウェア・モジュール内の複数の別々の位置に、上記資格検証動作をトリガする単一の目的コード命令を配置することを 含む、請求項9に記載の、ソフトウェア・モジュールの使用を管理するための方法。

【請求項11】上記資格検証動作をトリガする上記目的コード命令が、上記ソフトウェア・モジュールの適切な実行に必要な追加ステップをも実行し、その結果、目的コード命令を除去することによって上記ソフトウェア・モジュールが修正される場合は、上記の追加ステップが実行されず、上記ソフトウェア・モジュールが適切に実行されなくなる、請求項10に記載の、ソフトウェア・モジュールの使用を管理するための方法。

【請求項12】上記資格検証動作をトリガする上記目的コード命令が、上記ソフトウェア・モジュールの製品番号を含む、請求項10に記載の、ソフトウェア・モジュールの使用を管理するための方法。

【請求項13】資格付与情報を含みかつそれぞれが製品番号に関連する複数のエントリを備えた製品ロック・テーブルを、上記のコンピュータ・システム内で維持する段階を含み、資格検証動作をトリガする上記段階が、上記目的コード命令中に含まれる製品番号に関連する製品ロック・テーブル中のエントリに含まれる上記資格付与情報にアクセスすることを含む、請求項12に記載の、ソフトウェア・モジュールの使用を管理するための方法。

【請求項14】複数のデータ・ビットから構成され、上記コンピュータ・システムが上記ソフトウェア・モジュールを実行する資格をもつかどうかを判定できるようにする情報を提供する資格付与キーを生成する段階と、上記資格付与キーを上記コンピュータ・システムに入力する段階とを含む、請求項9に記載の、ソフトウェア・モジュールの使用を管理するための方法。

【請求項15】上記コンピュータ・システムが独自の識別子を含み、資格付与キーを生成する上記段階が、上記の独自の識別子を用いて上記資格付与キーを生成し、上記資格付与キーが、同じ独自の識別子を含むコンピュータ・システム上でのみソフトウェアを実行する資格を与える、請求項14に記載の、ソフトウェア・モジュールの使用を管理するための方法。

【請求項16】ソフトウェア・モジュールを実行する資格を受け取る手段を有し、かつ上記ソフトウェア・モジュール内のトリガ手段にตอบสนองして、上記コンピュータ・システムが上記ソフトウェア・モジュールを実行する資格をもつことを検証する資格検証手段を有するコンピュータ・システム上で、上記ソフトウェア・モジュールが実行される、資格の付与を管理するためのプログラム製品であって、記録媒体上に記録された少なくとも一つのソフトウェア・モジュールと、コンピュータ・システム上で資格の検証をトリガする、上記ソフトウェア・モジュール中の複数の独立したトリガ手段とを備える、プログラム製品。

【請求項17】上記複数の独立したトリガ手段のそれぞれが、上記資格の検証をトリガする単一の目的コード命令である、請求項16に記載の、資格付与を管理するためのプログラム製品。

【請求項18】上記資格検証手段をトリガする上記目的コード命令が、上記ソフトウェア・モジュールの適切な実行に必要な追加ステップをも実行し、その結果、上記目的コード命令を除去することによって上記ソフトウェア・モジュールが修正される場合は、上記追加ステップが実行されず、ソフトウェア・モジュールが適切に実行されなくなる、請求項17に記載の、資格付与を管理するためのプログラム製品。

【請求項19】上記資格検証手段をトリガする上記目的コード命令が、上記ソフトウェア・モジュールの製品番号を含み、

上記コンピュータ・システム上の上記資格検証手段が、資格付与情報を含みかつそれぞれが製品番号に関連する複数のエントリを備えた製品ロック・テーブルをコンピュータ・システム内に有する、請求項17に記載の、資格付与を管理するためのプログラム製品。

【請求項20】ソフトウェア・モジュールを実行する資格を受け取る手段を有し、かつ上記ソフトウェア・モジュール内のトリガ手段にตอบสนองして、上記コンピュータ・システムが上記ソフトウェア・モジュールを実行する資格をもつことを検証する資格検証手段を有するコンピュータ・システム上で、上記ソフトウェア・モジュールが実行される、ソフトウェア・モジュールを配布する方法であって、

資格の検証をトリガする複数の独立したトリガ手段を上記ソフトウェア・モジュール内に配置する段階と、上記ソフトウェア・モジュールを上記コンピュータ・システムに配布する段階と、上記コンピュータ・システムに上記ソフトウェア・モジュールを実行する資格を付与する段階とを含む、ソフトウェア・モジュールを配布する方法。

【請求項21】上記複数の独立したトリガ手段を上記ソフトウェア・モジュール内に配置する上記段階が、上記資格の検証をトリガする単一の目的コード命令を、上記ソフトウェア・モジュール中の複数の別々の位置に配置することを含む、請求項20に記載の、ソフトウェア・モジュールを配布する方法。

【請求項22】上記資格検証をトリガする上記目的コード命令が、上記ソフトウェア・モジュールの適切な実行に必要な追加ステップをも実行し、その結果、上記目的コード命令を除去することによって上記ソフトウェア・モジュールが修正される場合は、上記の追加ステップが実施されず、ソフトウェア・モジュールが適切に実行されなくなる、請求項21に記載の、ソフトウェア・モジュールを配布する方法。

【請求項23】上記資格検証をトリガする上記目的コード命令が、上記ソフトウェア・モジュールの製品番号を含み、コンピュータ・システム上の上記資格検証手段が、資格付与情報を含みかつそれぞれが製品番号に関連する複数のエントリを備えた製品ロック・テーブルを、コンピュータ・システム内に有する、請求項21に記載の、ソフトウェア・モジュールを配布する方法。

【請求項24】上記コンピュータ・システムに上記ソフトウェア・モジュールを実行する資格を付与する上記段階が、

複数のデータ・ビットから構成され、上記コンピュータ・システムが上記ソフトウェア・モジュールを実行する資格をもつかどうかを判定できるようにする情報を提供する、資格付与キーを生成する段階と、上記資格付与キーをコンピュータ・システムに伝送する段階とを含む、請求項20に記載の、ソフトウェア・モジュールを配布する方法。

【請求項25】上記コンピュータ・システムが独自の識別子を含み、資格付与キーを生成する上記段階が、上記の独自の識別子を用いて上記の資格付与キーを生成し、上記資格付与キーが、同じ独自の識別子を含むコンピュータ・システム上でのみソフトウェアを実行する資格を与える、請求項24に記載の、ソフトウェア・モジュールを配布する方法。

【請求項26】上記ソフトウェア・モジュールを配布する上記段階が、それぞれ単一の記憶媒体上に資格検証をトリガする複数の独立したトリガ手段を有する、複数のソフトウェア・モジュールを配布することを含み、上記コンピュータ・システムに上記ソフトウェア・モジュールを実行する資格を付与する上記段階が、上記の単一の記憶媒体上の上記複数のソフトウェア・モジュールのうち少なくとも2つに別々の資格を与えることを含む、請求項20に記載の、ソフトウェア・モジュールを配布する方法。

【発明の詳細な説明】

【0001】

【産業上の利用分野】本発明は、コンピュータ・ソフトウェアの使用に関し、より具体的には、コンピュータ・ユーザがライセンス・ソフトウェアをライセンスに従わない形で使用できるのを制限することに関する。

【0002】

【従来の技術】現代のコンピュータ・システムは、数えきれない時間にわたる技術上及びプログラミング上の熟練の成果をその設計に取り入れた、極めて複雑な機械である。コンピュータ・システムは多くの構成要素を含んでいるが、それらはハードウェアとソフトウェアに大別できる。ハードウェアは、システムを構成する、有形の回路、ボード、ケーブル、記憶装置、格納装置などである。しかし、タイプライタでビューリッター賞を受賞できる脚本を書けないのと同様、ハードウェアは、それ自体で、現実の世界の問題を解決することはできない。ハードウェアは、何をすべきかを告げる命令を必要とする。「ソフトウェア」とは、そのもっとも純粋な形では、ハードウェアに有用な仕事を実行させる命令を言う。(ただし漠然と、ソフトウェアが記憶され配布されている媒体に適用されることもある)。ハードウェアと同様に、ソフトウェアも人間の創意工夫の産物である。

高品質のソフトウェアは、プログラマとも呼ばれる作成者の側の、かなりの創造性、訓練、知能を必要とする。多くの大学が、人々にソフトウェアを作成する技術を教える目的で、「コンピュータ科学」及び類似の学科の教育課程を設けている。この業界全体が、何千人ものキャリアを抱え、有用な仕事をするソフトウェアを作成するまでに成長して来た。ソフトウェアの開発に莫大な労力が費やされる結果、コンピュータ・システムの一部分であるソフトウェアの価値が、ハードウェアの価値を上回ることも珍しくはない。

【0003】現代のコンピュータ・システムの特徴の一つは、非常に高速度で容易にデータを伝送し複製できることである。一般的に、これは必要かつ有益な能力である。しかし、このことはまた、何年も労力をかけて作成したソフトウェアが、比較的安価な磁気媒体で1秒の何分の一かで複製でき、濫用される可能性があることを意味している。許可を得ていない人々が、僅かな費用と労力で、価値のあるソフトウェアを複製することができ、この慣行は、一般に、ソフトウェア著作権侵害と呼ばれている。近年、ソフトウェア著作権侵害を抑制するため、刑法及び民法上の義務を課する様々な法律が制定されてきている。しかしながら、ソフトウェアが比較的容易に複製でき、かつソフトウェアが高価なためにそうしたくなる誘惑にかられるため、ソフトウェアの著作権侵害は依然として問題となっている。

【0004】「パーソナル・コンピュータ」と呼ばれる小型コンピュータ用の大量配布される安価なソフトウェアの場合、全顧客に対して同額の固定した一括払い料金で、ソフトウェアの使用ライセンスを認めるのが普通である。これは、大型のメインフレーム・コンピュータ・システムでは、それほど行われていない。このような大型システム用のソフトウェアは、数百万行のコードを必要とすることがあり、開発及び維持に非常に大量の投資を必要とする。開発者がこの投資を回収するのに十分な単一の固定した一括払い料金を設定すると、多くの小口ユーザにとって、使用できないほど高価になりがちである。したがって、メインフレームの場合は、使用量に基づくソフトウェアのライセンス料を請求するのが普通である。「段階的価格設定」と呼ばれるこのような一つの方法は、可変料金体系に従って顧客の機械の性能に基づいた料金を請求するものである。より多くの端末が接続されたより高速の機械を使用する顧客ほど、同じソフトウェアに対し、性能の劣る機械の顧客よりも高いライセンス料を支払う。もう一つの通常の慣行は、ソフトウェアの保守グレードアップに対して別途料金を請求するものである。

【0005】高品位のソフトウェアを作成するのに必要な専門知識のレベルと、それにかかる時間及び労力の量を考えると、このようなソフトウェアの作成には金がかかる。ソフトウェアの開発者がその訓練と努力に比して

報われない場合は、ソフトウェアを作成する人がいなくなる。開発者が行ったソフトウェアへの投資を保護することは、実際上の要請であるだけでなく、道徳上の要請でもある。したがって、ソフトウェアの正当な所有者は、ソフトウェアを許可なしに使用することが難しく、ソフトウェアの開発者がその製品に対して正当に報われる、ソフトウェア配布方法の開発を求めて来た。

【0006】ソフトウェアは、正当な所有者から多数の異なる方法で配布される。無許可の使用を防止する観点から、これらの配布方法は、無制限資格付与方法、制限資格付与方法、非資格付与方法の3つのグループに大別できる。

【0007】無制限資格付与方法とは、配布されたソフトウェアが、その設計の対象であったどの機械でも、制限なしに走行するという意味である。無制限資格付与方法のソフトウェアを配布する所有者は、各ユーザに、ユーザがそれに対して対価を支払いそれを実行する資格を有するソフトウェアだけを配布しなければならない。このようなソフトウェアの受取側がこれを許可されない形で複製したり使用したりするのを妨げるものは、法的及び契約上の義務だけである。大部分のパーソナル・コンピュータで使用されているものなど、一括払い料金でライセンスが与えられる安価なソフトウェアでは、これがもっとも普通の配布方法である。

【0008】制限資格付与方法とは、ソフトウェアが、ユーザがそれを複写していくらでも多くの機械上で実行できるのを制限する、ある種の制限を内蔵していることを意味する。いくつかの異なる制限資格付与方法がある。その一つは、配布されたオリジナルから作成できるコピーの数を制限する、複写制限である。複写制限は、ソフトウェア著作権侵害に対してあるレベルの保護を実現するが、若干の欠点をもつ。無制限資格付与方法と同様に、複写制限でも、所有者が各ユーザに、そのユーザが実行する資格を有するソフトウェアだけを配布することが必要となる。これはフルブープではなく、保護機構を打ちやぶって、複写保護されたソフトウェアの文字通りのコピーをとることのできるプログラムも存在する。最後に、これは、ユーザがバックアップの目的で正当なコピーをとれる、またはソフトウェアを高速記憶装置から走行できるのを妨げる。もう一つの制限資格付与方法は、ユーザまたは機械に特有の情報をソフトウェア自体の中にコード化するものである。ソフトウェアを実行する際、機械は、ソフトウェアがその機械またはユーザに許可されていることを確かめるために検査を行う。この方法は、ユーザが正当なコピーをとれるのを妨げずに、保護を実現する。ただし、これは、配布されたソフトウェアの各コピーをそれぞれ独自にコンパイルし、配布媒体に載せ、発送しなければならないので、非常に高価な配布システムが必要となる。

【0009】非資格付与方法は、配布されたソフトウェア

が使用禁止にされていて、実行するには別途配布される資格付与方法を必要とすることを意味する。ある非資格付与方法は、特注ソフトウェアの配布を完全に避ける可能性がある。ただし、必ずしもこのような方法の全てがこうした能力をもっているわけでない。たとえば、所有者は、同じ1組の多数のソフトウェア・プログラムを単一の総称媒体上でその全ての顧客に配布し、顧客が支払いを済ませたプログラムだけを実行することを許す、個別化された許可キーを各顧客に別途配布することができる。非資格付与方法は、他の方法に付随する多くの問題を回避するが、現在の設計は、資格付与方法の偽造や著しい性能劣化を受ける恐れが大きい。大抵の場合、ソフトウェアを実行するための資格付与方法を差し止めるために用いられる機構は、性能劣化という検証上のオーバーヘッドを避けるため、資格検証手段をソフトウェア・モジュール内に（データあるいは命令として）集中することを必要とする。場合によっては、この資格付与方法のオーバーヘッドが、製品識別子のサイズ、及び配布されたソフトウェア内の保護ルーチンの実装によることがある。あるいはまた、そのオーバーヘッドがソフトウェアを走行させながら複雑な復号手段を実行する必要によることもある。こうした資格検査の集中の結果、経験あるプログラマが、「パッチング」すなわち目的コードの選択された小部分を無効にすることによって、保護機構を無効にするのは、比較的容易である。別の場合、目的コードは資格検証を行わず、モジュールを、それからビット署名を作成することによって識別する、安全呼出経路がそれを行う。これはパッチングを受け難くするが、不可避的に呼出し機構の重大な性能劣化をひき起こす。

【0010】従来技術で教示されている保護方法は、保護レベルと使いやすさとの折合いをつけるものである。機械特有の情報をソフトウェア中にコード化することによって比較的高レベルの保護を得ることが可能であるが、配布される各ソフトウェア・コピーが独自のものとなる、非常に複雑な配布システムを維持するという犠牲を払わなければならない。より安価な配布も可能であるが、保護が一部失われるという犠牲を払わなければならない。高レベルの保護を実現し、大量配布技法を用いて容易に配布でき、かつシステムの性能、ユーザが正当にバックアップ複写用コピーをとる可能性、その他の必要な機能を不当に妨げない方法が求められている。同時に、段階的価格設定、及び異なるソフトウェアの異なるバージョンごとに別々のライセンス料をサポートする方法も求められている。

【0011】

【発明が解決しようとする課題】本発明の目的は、コンピュータ・システムにおけるソフトウェアの使用を管理する改善された方法及び装置を提供することにある。

【0012】本発明の他の目的は、コンピュータ・システムにおける無許可のソフトウェア使用に対するより高

レベルの保護を提供することにある。

【0013】本発明の他の目的は、ソフトウェアを無許可使用に対して保護する費用を削減することにある。

【0014】本発明の他の目的は、無許可使用に対して保護されたソフトウェアを実行するコンピュータ・システムの性能を増強することにある。

【0015】本発明の他の目的は、無許可使用に対して保護されたソフトウェアの配布者の配布システムを簡単にするにある。

【0016】本発明の他の目的は、このような保護がソフトウェアの正当な使用に及ぼす影響を削減する、ソフトウェアを無許可使用から保護する方法及び装置を提供することにある。

【0017】本発明の他の目的は、ユーザがソフトウェアの正当なバックアップ用コピーをとるのを許しながら、ソフトウェアを無許可使用から保護する改善された方法及び装置を提供することにある。

【0018】本発明の他の目的は、ソフトウェアを無許可使用の使用が可能となるように改変するのを難しくすることにある。

【0019】本発明の他の目的は、段階的に価格設定したソフトウェアを配布する、改善された方法及び装置を提供することにある。

【0020】

【課題を解決するための手段】本発明によれば、ソフトウェアは、実行するための資格付与なしに配布される。別々に配布される暗号化された資格付与キーにより、ソフトウェアの実行が可能となる。この資格付与キーは、それに対してソフトウェアのライセンスが与えられている機械の通し番号と、どのソフトウェア・モジュールがその機械で走行する資格をもつかを指示する複数の資格付与ビットを含んでいる。安全解読機構が機械に内蔵されている。安全解読機構が機械の通し番号を取り出し、これを資格付与キーを解読するためのキーとして使用する。次いで、資格付与情報が、メモリ内の製品ロック・テーブルに記憶される。

【0021】配布されたソフトウェアは、複数の資格検証トリガを含んでいる。好ましい実施例では、各トリガは、ソフトウェア・モジュールの製品番号を識別する、目的コード中の単一機械語命令である。ソフトウェア・モジュールの実行中にこのような資格検証トリガに出会うと、機械は、ソフトウェア・モジュールの製品番号に対応する製品ロック・テーブルのエントリを検査する。製品が走行する資格をもつ場合、正常に実行が継続され、そうでない場合は、実行が打ち切られる。この検証はただ1個の機械語命令しか要しないので、システム全体の性能にほとんど影響を与えずに実行できる。その結果、かなりの数のこのような資格検証トリガを目的コード中に配置し、誰かがこのトリガを「パッチング」することによってコードを改変することを事実上不可能にす

ることができる。別の実施例では、資格検証トリガが、ソフトウェア・モジュールの適正な実行に必要な何らかの有用な仕事をも行う。これによって、ソフトウェアを「パッチング」するのが一層難しくなり、このような検証トリガの性能に対する影響がさらに軽減される。

【0022】ソフトウェア自身はどのような資格付与も含んでいないので、配布の制限は必要でない。好ましい実施例では、ソフトウェアの配布者は、複数のソフトウェア・モジュールを単一の総称媒体に記録し、記録された同じ1組のモジュールをその全ての顧客に配布することができる。各顧客は、自分がライセンスを与えられているソフトウェア・モジュールだけを実行できる、独自の資格付与キーを受け取る。顧客に、ライセンスが与えられていない何らかのモジュールが与えられても、適切なキーなしにはそれが実行できないので、大きな問題にはならない。顧客は、自由にソフトウェアを自分のシステムのその他の記憶装置にロードし、あるいはソフトウェアのバックアップ用コピーをいくらかでも作成することができる。

【0023】

【実施例】本発明の好ましい実施例にもとづくソフトウェア保護機構の主要要素の説明図を図1に示す。顧客のコンピュータ・システム101は、制御記憶機構103に緊密に結合された中央演算処理装置(CPU)102、ランダム・アクセス・システム・メモリ104、消去不能で電子的に読取り可能な独自の識別子105、複数の記憶装置106、107、108を含んでいる。好ましい実施例では、独自の識別子105は機械の通し番号である。好ましい実施例では、記憶装置106~108は、回転式磁気ディスク記憶装置であるが、他の記憶技術を使用することも可能である。コンピュータ・システム101はまた、オペレータからの入力を受け取る操作卓109及びソフトウェア媒体読取装置110をも含んでいる。図1には1台の操作卓、1個のソフトウェア媒体読取装置、3個の記憶装置が示されているが、システム101に取り付けられるこのような装置の実際の数は可変であることを理解されたい。また、追加の装置をシステム101に取り付けられることを理解されたい。好ましい実施例では、コンピュータ・システム101は、IBM社のAS/400コンピュータ・システムであるが、他のコンピュータ・システムも使用できる。

【0024】ソフトウェア・モジュールは、それを動作させるのに必要な資格付与キー(entitlement key)とは別に配布される。本来、ソフトウェア・モジュールは、開発用コンピュータ・システム125上で作成される。開発用コンピュータ・システム125は、コンパイラ126及びトランスレータ127を含んでいる。ソフトウェア・モジュールはソフトウェア記録媒体112上に記録され、倉庫120に格納され、そこから顧客に配布される。配布者の販売事務所121か

ら、暗号化された資格付与キー111が配布される。販売事務所121は、マーケティング用コンピュータ・システム124にアクセスすることができる。マーケティング用コンピュータ・システム124は、資格付与キーの生成/暗号化プログラム122と、顧客情報を含むデータ・ベース123とを含んでいる。具体的には、データ・ベース123は、暗号化された資格付与キーの生成に必要な機械の通し番号とプロセッサの種類の情報を含んでいる。好ましい実施例では、マーケティング用コンピュータ・システム124は、中央に位置して複数の販売事務所と通信するコンピュータである。ただし、マーケティング用コンピュータ・システム124は、販売事務所自体に位置し、局所または中央のデータ・ベースにアクセスすることもできる。図1には、配布者の制御下にある別々の2つのコンピュータ・システム124、125が示されているが、両方の機能を実行する物理的に単一のコンピュータ・システムでもよいことを理解されたい。

【0025】ソフトウェアの配布者から顧客に、暗号化された資格付与キー111が郵送、電話またはその他の適当な手段で送られる。資格付与キーを電子的にまたはフロッピー・ディスクなどの磁気媒体上で伝送することが可能であるが、キーは、オペレータが操作卓109上でタイプして、これをシステム101中に入力できるほど十分簡潔なものである。

【0026】好ましい実施例では、多数のソフトウェア・モジュールが同一媒体112上で配布される。配布者は、資格付与キーを介して各モジュールを使用する資格を独立に与えることができる。倉庫120は、媒体112上に、総称的に1組のソフトウェア・モジュールを格納する。各顧客に、どのモジュールを使用するライセンスが与えられているかにかかわらず、同じ総称的な1組のソフトウェア・モジュールが発送される。別途に配布される資格付与キーは、どのソフトウェア・モジュールをその上で実行する資格があるかをシステム101が決定するための情報を含んでいる。

【0027】好ましい実施例では、ソフトウェア媒体112は、一枚または複数の読取り専用光ディスクから構成され、媒体読取装置110は光ディスク読取装置である。ただし、電子式配布媒体やその他の配布媒体も使用できることを理解されたい。ソフトウェア媒体112を受け取ると、通常、顧客は、所望のソフトウェア・モジュールを媒体読取装置110からシステム101にロードし、そのソフトウェア・モジュールを記憶装置106～108に記憶する。顧客は、ソフトウェア・モジュールの一つまたは複数のバックアップ用コピーを任意の適当な媒体上に生成し、これらをアーカイブに記憶することができる。ソフトウェア媒体112は、システムへの複写やロードに対する制限を含んでおらず、自由に複写可能かつロード可能である。

【0028】好ましい実施例による暗号化前の資格付与キー200の内容が図2に示されている。このキーは、料金グループ・フィールド201、ソフトウェア・バージョン・フィールド202、キー形式フィールド203、機械通し番号フィールド204及び製品資格付与フラグ205を含んでいる。料金グループ・フィールド201は、16通りの機械段階値から、1つを指定し、ソフトウェアの段階的価格設定をサポートするのに使用される。ソフトウェア・バージョン・フィールド202は、資格が与えられるソフトウェアのバージョン・レベルを指定する。ソフトウェアのグレードアップを維持するための別途料金が課せられることが予想されている。資格付与キー200中で指定されるバージョンは、そのバージョン・レベルより以前（下位）の全レベルのソフトウェアに資格を与える。キー形式フィールド203はキー・フォーマット、キーの関連性の将来変更のため、あるいはサポートされる異なる製品の番号拡張のために保留された領域である。機械通し番号フィールド204は、資格付与キーが対象としている機械の通し番号を含んでいる。製品資格付与フラグ205は、それぞれが製品番号に対応する80本の別々の製品フラグを含む、80ビットのフィールドである。対応する製品番号に資格が与えられている場合、このビットは“1”にセットされ、そうでない場合は、“0”にセットされる。

【0029】好ましい実施例では、ソフトウェア・モジュールは、コンパイルされた目的コードとして配布される。典型的なソフトウェア・モジュール300を図3に示す。このソフトウェア・モジュールは、コンピュータ・システム101上で実行できる複数の目的コード命令を含んでいる。本発明によれば、いくつかの資格検証トリガ命令（以下、「資格検証トリガ」と表記）301が目的コードに組み込まれている。ソフトウェア・モジュール内に含まれる全ての資格検証トリガ301は、同一である。資格検証トリガ301は、命令コード・フィールド302、バージョン・フィールド303、製品番号フィールド304を含んでいる。フィールド305は使用されていない。命令コード・フィールド302は、実行すべき操作を識別する、目的コード命令の動詞部分である。バージョン・フィールド303は、ソフトウェア・モジュールのバージョン・レベルを識別する。製品番号フィールド304は、そのソフトウェア・モジュールに関連する製品番号を識別する。バージョン及び製品番号の情報はモジュールごとに変わるが、全ての資格検証トリガには単一の命令コードが使用される。別の実施例では、資格検証トリガは、他の何らかの有用な仕事を実行する直接命令（とりわけ、その処理用のオペランドを命令中で指定しておく必要のない命令）でもある。このような別の実施例では、トリガ命令が実行されると、システム101は資格検証と同時に他の何らかの操作を実行する。

【0030】コンピュータ・システム101は、暗号化された資格付与キー111を受け取ってデコードする手段、ならびに資格検証トリガにตอบสนองしてシステムがソフトウェア・モジュールを実行する資格をもつことを検証する手段を含んでいる。好ましい実施例では、図4に示されているように、これらの機能は、異なるレベルのシステム・ハードウェア及びシステム・ソフトウェアの間で分割されている。この実施例では、システム101は、ハードウェア・レベル401、水平マイクロコード・レベル402、実行可能コード・レベル403、仮想計算機レベル404という、4レベルのハードウェア／ソフトウェア機能を含んでいる。機械インターフェース405が、仮想計算機レベルを下位の全てのレベルから分離している。機械インターフェース405は、顧客に対して定義されている最も下位レベルにあるインターフェースである。すなわち、仮想計算機レベルの命令ビットは顧客に対して定義されるが、それより下のレベルの操作は定義されない。したがって、顧客は、機械インターフェース・レベルより下のレベルの命令を直接変更する能力がない。永久的な、変更不能な機械通し番号410が、ハードウェア・レベル401に格納されている。

【0031】水平マイクロコード402は、実行可能な命令セットを解釈するマイクロコード・エントリを含んでいる。これは、制御記憶機構103、すなわち好ましい実施例では、顧客による変更が不可能な読み専用メモリ（ROM）に、物理的に記憶されている。水平マイクロコード中のエントリは、機械キー取得機能420、ロック設定機能421、ロック検査機能422をサポートしている。機械キー取得機能420は、好ましい実施例ではシステム通し番号に基づく、独自の識別子を、ある永久的なハードウェア位置から取り出す。ロック設定機能421は、製品ロック・テーブル460にアクセスし、テーブル中のエントリを変更する。ロック設定機能は、製品ロック・テーブル460を変更できる唯一のマイクロコード機能である。ロック検査機能422は、製品ロック・テーブル460にアクセスし、エントリの一つを読み取って資格が付与されているかどうかを検証する。

【0032】実行可能コード・レベル403は、下位レベルの監視サポート、及び機械インターフェースで定義された命令を実施するサポートを含んでいる。これはメモリに物理的に記憶されるが、その内部仕様が顧客に対して定義されていないので、実際上は顧客によって変更できない。下位レベルのサポートには、ロック解除ルーチン430、初期プログラムロード（IPL）ロック再検証ルーチン431、例外処理ルーチン432が含まれる。アンロック・ルーチン430は、固有の機械キーを用いて資格付与キー111をデコードし、暗号化された資格付与キー111を製品キー・テーブル450に記憶する。初期プログラムロードロック再検証ルーチン43

1は、システムが再び初期設定されると、コード化された製品キー・テーブル450の内容を取り出して、製品ロック・テーブル460を再構築する。例外処理ルーチン432は、水平マイクロコード・レベル402の機能によって生成される例外条件にตอบสนองする。実行可能コード・レベル403は、目的コード形式のソフトウェア・モジュール300を含んでいる。

【0033】仮想計算機レベル404は、機械語命令によって表されているものとしてソフトウェアを参照する。このレベルの計算機は、機械語命令が直接実行可能ではないという意味で、仮想計算機として動作する。仮想計算機レベル404は顧客に利用できる最も下位レベルのインターフェースなので、システム上で実行可能な目的コード形式のモジュールを作成するのに、非従来型のコンパイル経路が必要である。このコンパイル処理については、後に説明する。厳密には、この非従来型のコンパイル経路を介して作成されるソフトウェアは、実行しようとする実行可能な目的コードの形式をとらなければならないが、機械語命令が直接実行可能であるかのように、仮想計算機レベルの機械語命令で表して議論するのが普通であり、よりわかりやすい。これが確立されると、仮想計算機レベル404がコンパイルしたユーザのソフトウェアを含んでいるとすることができる。これはまた、システム101に対するより上位レベルのオペレーティング・システムのサポートも含んでいる。仮想計算機レベル404でこのオペレーション・システムのサポートは、資格付与キーの入力をサポートするのに必要な2個のユーザ・インターフェース・ルーチンを含んでいる。一般入力ルーチン441は、通常の操作に入力を処理するのに使用する。さらに、オペレーティング・システムの初期導入中に資格付与キーを入力するには特別なインストール入力ルーチン440が必要である。これが必要なのは、機械インターフェース・レベル405より上位のオペレーティング・システムの部分が、本発明では他のプログラム製品として扱われるためである。すなわち、このような部分は製品番号をもち、その目的コードは資格検証トリガの影響を受ける。インストール入力ルーチン440は、オペレーティング・システム、資格検証トリガをもたない唯一の部分であり、従って、システムを最初に導入する際に資格付与キーが入力できるようになる。

【0034】ソフトウェア・モジュール300は、システム101上で実行されるコンパイルされた目的コード形式のプログラム製品の一部である。これは、仮想計算機レベルの他の対象にアクセス可能であるという意味で、仮想計算機レベル404のエンティティとして存在する。ただし、実際に実行可能なコードは、破線の枠で示されているように、実行可能コード・レベル403で動作する。実行可能コードは、水平マイクロコードのロック検査機能422によって実行される、資格検証トリ

が301(一つだけが図に示されている)を含んでいる。

【0035】コード化された製品キー・テーブル450が図5に示されている。製品キー・テーブル450はランダム・アクセス・メモリ104に入っており、不揮発性記憶装置に複製されているため、システムの電力を遮断したり、その他の理由で再初期設定しなければならない場合に、回復することが可能である。テーブル450は、それぞれが各製品番号に対応し得る80個のエントリ501を含んでいる。各エントリ501は、そのエントリの製品番号に適用される暗号化された資格付与キー502と、そのキーが最初にいつ使用されたかを示す日時スタンプ503と、そのキーのバージョン番号504と、そのキーの料金グループ505と、そのキーが製品をアンロックするかどうかを示す資格付与ビット506との完全なコピーを含んでいる。日時スタンプ503は暗号化することができる。バージョン番号504、料金グループ505及び資格付与ビット506は、暗号化された資格付与キー502に含まれる情報を繰り返す。それらは、照合をサポートするためにこのテーブルに含まれている。ただし、最初に暗号化された資格付与キー502中の情報を検証してからでない、製品ロック・テーブル460のエントリを改変して、プログラム実行の資格を与えることはできない。製品キー・テーブル450中の各資格付与キー502は暗号化された形なので、ユーザーのこのテーブルへのアクセスを防止するために特別な措置は必要でない。

【0036】図6に、製品ロック・テーブル460が示されている。このテーブルは、水平マイクロコードの完全な制御下にある、ランダム・アクセス・メモリ104の特別な下位アドレス範囲に入っている。製品ロック・テーブル460は、それぞれが各製品番号に対応し得る80個のエントリ601を含んでいる。各エントリは、資格付与の最大バージョンのレベルを示すバージョン番号を含んでいる。0のバージョン番号は、製品のどのバージョンにも資格付与がないことを示している。製品ロック・テーブル460の改変に対する保護の度をさらに強化するために、バージョン番号をスクランブルすることができる。

【0037】次に、本発明の好ましい実施例によるコンピュータ・システム101上でのソフトウェア・モジュールの動作について述べる。この動作には4つの部分がある。第1の動作は、複数の資格検証トリガを実行可能な目的コード形式のソフトウェア・モジュール中に配置する。第2の動作は、ソフトウェア・モジュールへのアクセスを許可する暗号化された資格付与キー111を生成する。第3の動作は、コンピュータ・システム101が資格付与キー111を受け取り、デコードして、記憶し、製品ロック・テーブル460を設定する。第4の動作は、ソフトウェア・モジュールをシステム101上で

実行して、資格検証トリガに出会ったときには、システム101に資格を検証させる。始めの2つの動作は、ソフトウェア配布者の管理下で実施される。後の2つの動作は、顧客のシステム101上で実施される。

【0038】ソフトウェア配布者は、ソフトウェア・モジュールをコンパイルするとき、資格検証トリガを目的コード中に配置しなければならない。開発用コンピュータ・システム125で起こる典型的なこの過程を図7に示す。原始コードは、資格検証トリガを含めず、通常通りプログラマによって生成される。ステップ701で、原始コードがコンパイラ126に入力され、ステップ702で、プログラム・テンプレートを作成する。プログラム・テンプレートは、仮想計算機レベル404(すなわち機械インターフェース405より上のレベル)の機械語命令を含んでいる。ステップ704で、プログラム・テンプレートは、その製品番号及びバージョン番号を識別すると共に、トランスレータ127への入力として働く。トランスレータ127は、自動的に、かなりの数の資格検証トリガを生成し、これを目的コード中のランダムな位置に挿入し、資格検証トリガの挿入後に参照を解決する。ステップ705で出力された結果得られる実行可能な目的コード形式のソフトウェア・モジュールには、資格検証トリガが含まれている。この実行可能な形式のソフトウェア・モジュールは、実行可能コード・レベル403の目的コード命令を含んでいる。

【0039】暗号化された資格付与キーを生成する過程を図8に示す。ステップ801で、販売担当員によって生成されたあるバージョン・レベルの1個または複数のプログラム製品に対するライセンスの注文が、マーケティング用コンピュータ・システム124に入力される。理論上、顧客が多数のバージョン・レベルの製品を注文することは可能であるが、一般的にはそうする理由はほとんどない。ただし、各資格付与キーは指定されたバージョン・レベルでしか動作しないので、顧客が異なるバージョン・レベルを注文すると、別々の資格付与キーを生成しなければならない。顧客の注文を受け取ると、ステップ802で、システム124で実行中のキー生成/暗号化プログラム122が、顧客に関する情報、具体的には機械の通し番号及びプロセッサ形式の情報を含むデータベース123にアクセスする。この情報を使って、暗号化されていない資格付与キー200の料金グループ・フィールド201及び機械通し番号フィールド204が生成される。ステップ803で、顧客の注文及び可能な製品番号提供のデータベースへの参照によって残りのフィールドが生成されて、完全な非暗号形式の資格付与キーが構築される。次いで、ステップ804で、キー生成/暗号化プログラム122が、当技術分野で既知のいくつかの暗号化技法のうちのどれかを使って、資格付与キーを暗号化する。次いで、ステップ805で、その結果得られた暗号化された資格付与キー111が、顧客に

伝送される。図1ではキー111は複数の2進ビットとして示されているが、資格付与キーを鍵盤から入力する仕事を簡単にするため、16進数字や英数字による等価物など2進ビットをグループ化した何らかの形式で顧客に提示してもよい。

【0040】コンピュータ・システム101上で資格付与キー111を受け取り、製品ロック・テーブル460を維持する過程を図9a及び図9bに示す。ステップ901で、顧客は、操作卓109を介して、資格付与キー111をコンピュータ・システム101に入力する。これが初期導入である場合には、インストール入力ルーチン440がオペレータと対話して入力を受け取る。そうでない場合は、一般入力ルーチン441が入力を受け取る。資格付与キーは、デコード過程を処理するロック解除ルーチン430に渡される。ステップ902で、ロック解除ルーチン430が、機械キー取得機能420に、機械通し番号を検索させ、機械キーを生成させる。次いで、ステップ903で、ロック解除ルーチン430が、機械キーを用いて資格付与キー111をデコードする。次いで、ステップ904で、以下に述べるように、ロック解除ルーチン430がコード化された製品キー・テーブル450を再構築する。デコードされた資格付与キーは、図2に示した形をとる。これは、各製品番号ごとに、その製品が新しい資格付与キーの下でロックを解除されているかどうかを示す、80ビットの製品資格付与フラグ・アレイ205を含んでいる。新しい資格付与キーは、それがロックを解除する全ての製品に対する置換されたキーと見なされる。ロック解除ルーチン430は、デコードされた資格付与キー中の各製品資格付与フラグ205を走査する(ステップ904)。ステップ905で、製品資格付与フラグが“1”(資格付与ありを指す)にセットされている場合、ステップ908で、製品キー・テーブル450中の対応するエントリが、新しい資格付与キー、バージョン番号及び料金グループ値で置換される。資格付与ビット・フィールド506が“1”にセットされ、日/時スタンプ・ビット・フィールド503が、資格付与キーがまだ使用されていないことを示すゼロの値にセットされる。新しいキーの製品資格付与フラグが“0”の場合には、バージョン番号と料金グループ番号が製品キー・テーブル450に記憶されているものと同じでない限り、新しい資格付与キーは効果がない。バージョン番号及び料金グループ番号が同じ場合(ステップ906)、資格付与キーは製品をロックする効果がある。したがって、ロック解除ルーチン430は、ステップ907で、製品ロック・テーブル460中のバージョン番号を“0”にセットするために、ロック設定機能421を呼び出し、ステップ908で、製品キー・テーブル450中の対応するエントリを新しい資格付与キーの値で置換する。製品キー・テーブル450が再構築されると、ステップ909で、その内容が記憶

装置にセーブされる。

【0041】前に資格を付与された製品が資格を失わない限り、製品キー・テーブル450の再構築は製品ロック・テーブル460に直接影響を与えない。製品は、要求に応じて、ロックを解除される。前に資格を付与されていないソフトウェア製品を最初に実行するとき、資格検証トリガに出会うと、システムによって例外が生成される。次いで、ステップ920で、例外処理ルーチン432が、製品のロック解除を試みるため、ロック解除ルーチン430を呼び出す。次に、ステップ921で、ロック解除ルーチン430が、コード化された製品キー・テーブル450中の適当なエントリから暗号化された資格付与キーを取り出し、ステップ922で機械キーを得て、ステップ923で資格付与キーを解読する。資格が付与されている(ステップ924)場合は、ステップ926で、ソフトウェア製品の製品番号に対応する、製品ロック・テーブル460のエントリ601中でバージョン番号を設定するために、ロック設定機能421が呼び出される。同時に、ステップ927で、ロック解除ルーチン430が、第1回使用の日時を日/時スタンプ・フィールド503に記録する。次いで、ステップ928で、資格検証トリガが再試行され、プログラムの実行が続行される。ステップ924で資格付与が示されない場合は、ステップ925で、プログラムの実行が打ち切られる。

【0042】製品ロック・テーブル460は、RAMに記憶されており、システムの再初期設定後は残らない。再初期設定(「IPL」)中に、製品ロック・テーブル430を再構築するため、IPLロック再検証ルーチン431が呼び出される。このルーチンは、上記のように、資格を検証し、製品ロック・テーブル460中の対応するエントリを再構築するために機械キーを得て、コード化された製品キー・テーブル450中の各資格付与キー・エントリを系統的にデコードする。

【0043】好ましい実施例による、ソフトウェア・モジュールを実行する過程を図10に示す。システム101によるソフトウェア・モジュールの実行は、モジュールの目的コード命令が終了するまで(ステップ1003)、これを取り出して(ステップ1001)実行する(ステップ1002)ことによってなされる。命令が資格検証トリガ301である(ステップ1004)場合は、ロック検査機能422が呼び出される。ステップ1005で、ロック検査機能422が、資格検証トリガに含まれる製品番号に対応する、製品ロック・テーブル・エントリ601にアクセスする。製品ロック・テーブル460中のバージョン番号が資格検証トリガ301に含まれるバージョン番号303に等しいかまたはそれより大きい場合には、ソフトウェアは実行する資格を付与される(ステップ1006)。この場合、ロック検査機能422はそれ以上の処置を行わず、システムはソフトウ

ウェア・モジュール内の次の目的コード命令の実行に進む。ソフトウェアが資格を付与されていない場合は、ロック検査機能が例外条件を生成して、制御を例外処理ルーチン432に渡し、例外処理ルーチン432がプログラムの実行を終了させる（ステップ1007）。システムは、ソフトウェアが資格を付与されていることを示す資格検査の結果をセーブしない。したがって、ソフトウェア・モジュール中で資格検証トリガに再び出会うと、上記のように、システムは再度資格を検証する。

【0044】別の実施例では、資格検証トリガを、従来型のコンパイル経路を介して目的コード中に注入できるように、追加の定義をすることが可能である。これは、顧客及びコンパイラ作成者が利用できる機械インターフェースが、目的コード形式のモジュールがシステムによって実行されるのと同じ実行可能な命令のビットとなっているものとなるはずである。目的コードのフォーマットが顧客に知られている従来型のコンパイル経路をサポートするため、資格検証トリガを無効にするかまたは変更するような目的コードの「パッチング」に対する障壁を追加する必要がある場合がある。このような追加の障壁の一つは、同時に他の何らかの機能を果たすように、資格検証トリガを定義することである。この場合、資格検証トリガによって実施される代替機能が、他の単純命令で実施できないことが肝要である。この代替機能は、コンパイルされたどんなソフトウェア・モジュールもかなり確実にその機能を果たすいくつかの命令を含むように、選定しなければならない。これらの判定規準に合致している場合、コンパイラは、自動的に、その通常のコンパイル手順の一部として、その代替機能を（それと同時に資格検証トリガも）果たす目的コードを生成することができる。この定義は、資格検証トリガを無効にするような目的コードの「パッチング」に対する重要な障壁をもたらすはずである。他の実施例は、資格検証トリガを、目的コード中で、それが識別する製品番号に対して単純な関係をもつアドレス可能位置に、配置しなければならないと定義することである。コンパイラが、通常のコンパイル過程の一部としてこれらの命令を適切なコード位置に注入するのは比較的簡単ではなく、命令実施態様での追加的検証を行うのは簡単ではなく、この定義は、資格検証トリガ中で供給される製品番号の識別を変更する、目的コードのパッチングに対する重要な障壁となるはずである。

【0045】好ましい実施例は、80個の独立な製品番号に対応している。本発明によってサポートされる製品番号の実際の数は可変であることを理解されたい。好ましい実施例では、配布者から配布される、別々にコンパイル可能なソフトウェア・モジュールの数が80を大幅に越えることも予想されている。製品番号の数は、配布者のマーケティング組織によって提供される別々に価格設定されたソフトウェア・パッケージの数に対応する。

各ソフトウェア・モジュールは一つの製品番号しかもないが、この製品番号を共有するソフトウェア・モジュールは多数存在し得る。たとえば、配布者は、画面編集、スペル・チェック、文書フォーマット化などを扱う別々のソフトウェア・モジュールを含む、ワード・プロセッシング・パッケージを提供する。こうしたソフトウェア・モジュールが常にワード・プロセッシング・パッケージの一部としてライセンスを与えられる場合には、それらのモジュールは共通の製品番号をもつことになる。別の実施例では、各ソフトウェア・モジュールごとに、別々の番号をもつことも可能である。

【0046】好ましい実施例では、ソフトウェアは一括払い料金でライセンスが与えられるが、グレードアップを維持するための追加料金が請求されることが可能である。別の実施例では、ある期間の間ソフトウェアのライセンスを許諾することができる。このような別の実施例においては、資格付与キーが、ソフトウェアのライセンスが与えられる期間の長さを示す追加のフィールドを含むことになる。コード化された製品キー・テーブル450中の製品番号に関連する日/時スタンプをライセンス許諾の期間の長さと比較して、ライセンスが満了したかどうかを判定するために、IPLロック再検証ルーチン431が、定期的に呼び出される。このとき、ライセンスが満了した場合には、製品ロック・テーブル460中の対応するエントリにロック解除ビット（図示せず）を設けておきこれを“0”にセットすることによって、新たな資格付与が得られるまで、ソフトウェア・モジュールのそれ以上の実行を防止するようにすることもできる。

【発明の効果】本発明によって、コンピュータ・システムにおけるソフトウェアの使用を管理する、改善された方法及び装置が提供される。

【図面の簡単な説明】

【図1】本発明の好ましい実施例によるソフトウェア保護機構の主要要素を示す図である。

【図2】本発明の好ましい実施例による資格付与キーの暗号化されていない内容を示す図である。

【図3】好ましい実施例による典型的な実行可能ソフトウェア・モジュールの内容を示す図である。

【図4】好ましい実施例によるソフトウェア保護機構をサポートするために顧客のコンピュータ・システム上で必要なハードウェア及びソフトウェアの構造を示す図である。

【図5】好ましい実施例によるコード化された製品キー・テーブルのフォーマットを示す図である。

【図6】好ましい実施例による製品ロック・テーブルのフォーマットを示す図である。

【図7】好ましい実施例による、資格検証トリガをソフトウェア・モジュールに配置するのに必要なステップのブロック図である。

【図8】好ましい実施例により、暗号化された資格付与キーを生成するのに必要なステップのブロック図である。

【図9】好ましい実施例による、顧客のコンピュータ・システム上で資格付与キーをデコードし、資格付与状況の記録を維持するのに必要なステップのブロック図である。

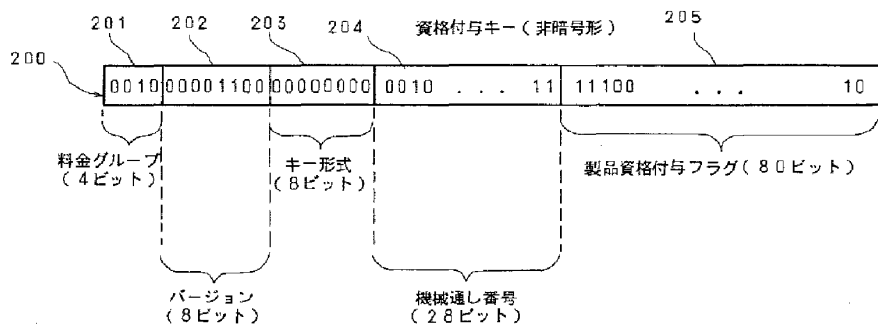
【図10】好ましい実施例による、ソフトウェア・モジュールの実行中に資格を検証するのに必要なステップのブロック図である。

【符号の説明】

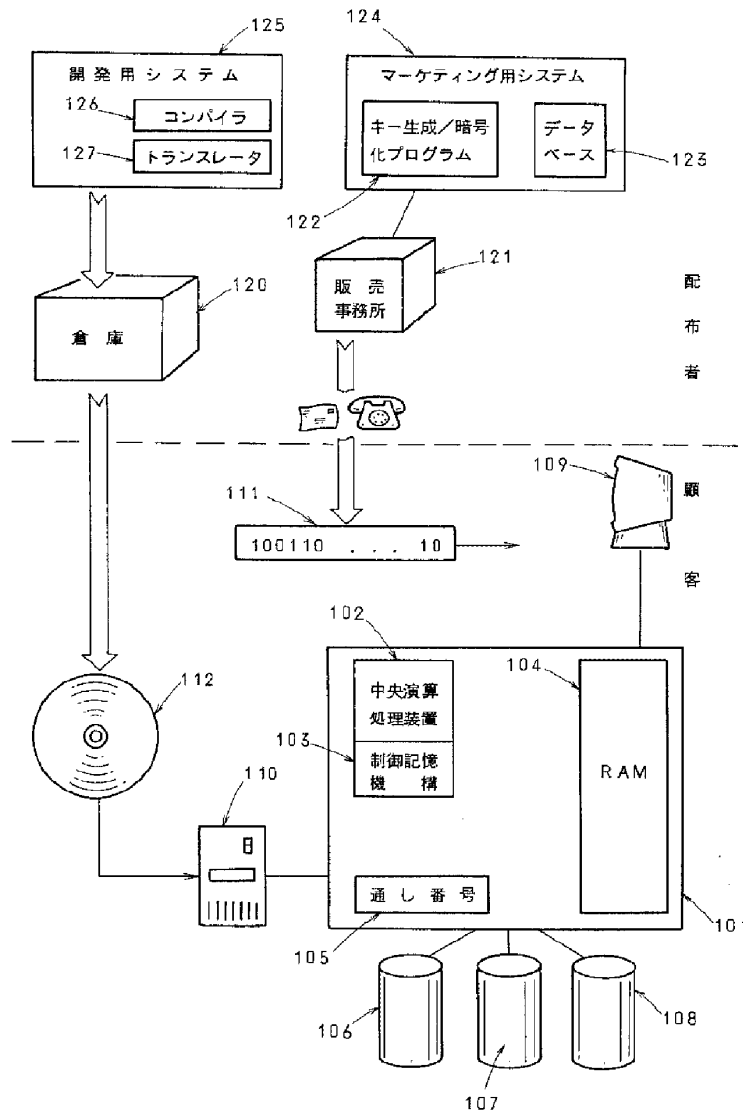
- 101 顧客側コンピュータ・システム
- 102 中央演算処理装置(CPU)
- 103 制御記憶機構

- 104 ランダム・アクセス・メモリ(RAM)
- 106 記憶装置
- 107 記憶装置
- 108 記憶装置
- 109 操作卓
- 110 媒体読取装置
- 112 ソフトウェア記録媒体
- 120 倉庫
- 121 販売事務所
- 123 データ・ベース
- 124 マーケティング用コンピュータ・システム
- 125 開発用コンピュータ・システム
- 126 コンバイラ
- 127 トランスレータ

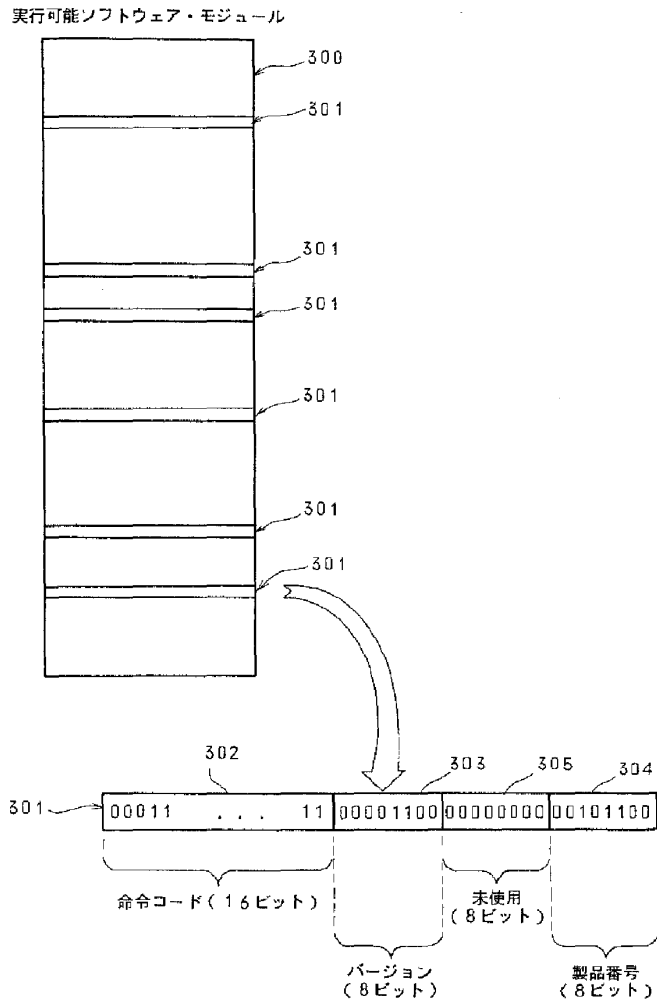
【図2】



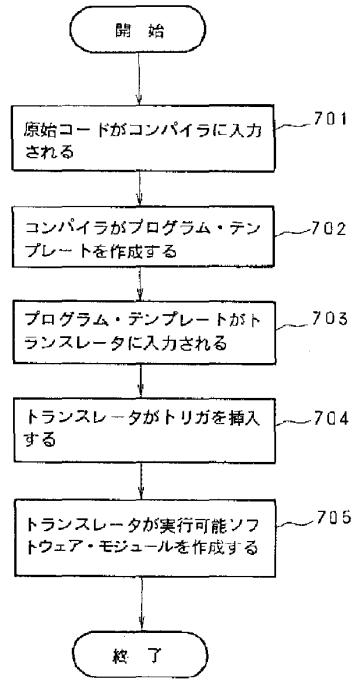
【図1】



【図3】



【図7】



【図4】

