wire interface circuitry 32 can be included with the input/output circuitry 26.

An energy circuit 34 may be necessary to maintain the memory circuitry 20 and/or aid in powering the other circuitry in the module 10. The energy circuit 34 could consist of a battery, capacitor, R/C circuit, photo-voltaic cell, or any other equivalent energy producing circuit or means.

The firmware architecture of a preferred embodiment of a secure transaction module and a series of sample applications using the module 10 will now be discussed. These examples are intended to illustrate a preferred feature set of the module 10 and to explain the services that the module offers. These applications by no means limit the capabilities of the invention, but instead bring to light a sampling of its capabilities.

I. OVERVIEW OF THE PREFERRED MODULE AND ITS FIRMWARE DESIGN

The module 10 preferably contains a general-purpose, 8051-compatible micro controller 12 or a reasonably similar product, a continuously running real-time clock 14, a high-speed modular exponentiation accelerator for large integers (math coprocessor) 18, input and output buffers 28, 30 with a one-wire interface 32 for sending and receiving data, 32 Kbytes of ROM memory 22 with preprogrammed firmware, 8 Kbytes of NVRAM (non-volatile RAM) 24 for storage of critical data, and control circuitry 16 that enables the micro controller 12 to be powered up to interpret and act on the data placed in an input circuitry 26. The module 10 draws its operating power from the one-wire line. The micro controller 12, clock 14, memory 20, buffers 28, 30, one-wire front-end 32, modular exponentiation accelerator 18, and control circuitry 16 are preferably integrated on a single silicon chip and packaged in a stainless steel microcan using packaging techniques which make it virtually impossible to probe the data in the NVRAM 24 without destroying the data. Initially, most of the NVRAM 24 is available for use to support applications such as those described below. One of ordinary skill will understand that there are many comparable variations of the module design. For example, volatile memory can be used, or an interface other than a one-wire could be used. The silicon chip can be packaged in credit cards, rings etc.

The module 10 is preferably intended to be used first by a Service Provider who loads the module 10 with data to enable it to perform useful functions, and second by an End User who issues commands to the module 10 to perform operations on behalf of the Service Provider for the benefit of the End User. For this reason, the module 10 offers functions to support the Service Provider in setting up the module for an intended application. It also offers functions to allow the End User to invoke the services offered by the Service Provider.

Each Service Provider can reserve a block of NVRAM memory to support its services by creating a transaction group 40 (refer to FIGS. 11 and 12). A transaction group 40 is simply a set of objects 42 that are defined by the Service Provider. These objects 42 include both data objects (encryption keys, transaction counts, money amounts, date/time stamps, etc.) and transaction scripts 44 which specify how to combine the data objects in useful ways. Each Service Provider creates his own transaction group 40, which is independent of every other transaction group 40. Hence, multiple Service Providers can offer different services in the same module 10. The number of independent Service Providers that can be supported depends on the number and complexity of the objects 42 defined in each transaction group 40. Examples of some of the objects 42 that can be defined within a transaction group 40 are the following:

RSA Modulus	Clock Offset
RSA Exponent	Random SALT
Transaction Script	Configuration Data
Transaction Counter	Input Data
Money Register	Output Data
Destructor	

Within each transaction group 40 the module 10 will initially accept certain commands which have an irreversible effect. Once any of these irreversible commands are executed in a transaction group 40, they remain in effect until the end of the module's useful life or until the transaction group 40, to which it applies, is deleted from the module 10. In addition, there are certain commands which have an irreversible effect until the end of the module's life or until a master erase command is issued to erase the entire contents of the module 10. These commands will be discussed further below. These commands are essential to give the Service Provider the necessary control over the operations that can be performed by the End User. Examples of some of the irreversible commands are:

Privatize Object	Lock Object
Lock Transaction Group	Lock Micro-In-A-Can ™

Since much of the module's utility centers on its ability to keep a secret, the Privatize command is a very important irreversible command.

Once the module 10, as a whole, is locked, the remaining NVRAM memory 24 is allocated for a circular buffer for holding an audit trail of previous transactions. Each of the transactions are identified by the number of the transaction group, the number of the transaction script 40 within the specified group, and the date/time stamp.

The fundamental concept implemented by the firmware is that the Service Provider can store transaction scripts 44 in a transaction group 40 to perform only those operations among objects that he wishes the End User to be able to perform. The Service Provider can also store and privatize RSA key or keys (encryption keys) that allow the module 10 to "sign" transactions on behalf of the Service Provider, thereby guaranteeing their authenticity. By privatizing and/or locking one or more objects 42 in the transaction group 40, the Service Provider maintains control over what the module 10 is allowed to do on his behalf. The End User cannot add new transaction scripts 44 and is therefore limited to the operations on objects 42 that can be performed with the transaction scripts 44 programmed by the Service Provider.

II. USAGE MODELS OF THE MODULE

This section presents a series of practical applications of the module 10, ranging from the simplest to the most complex. Each of these applications is described in enough detail to make it clear why the module 10 is the central enabling technology for that application.

A. Background of Secure E-Mail

In this section we provide an example of how a module 10 could be used to allow anyone to receive his or her own e-mail securely at any location.

1. Standard E-Mail

In a standard e-mail system, a user's computer is connected to a provider of Internet services, and the user's computer provides an e-mail password when polling the provider's computer for new mail. The mail resides on the provider's computer in plain text form, where it can be read by anyone working there. In addition, while traveling from its source, the mail passes through many computers and was also exposed at these locations. If the user receives his mail from his provider over a local area network, anyone else on the same network can capture and read the mail. Finally, with many e-mail systems that do not require the user to enter the password, anyone sitting at the user's computer can retrieve and read his mail, since his computer automatically provides the password when it polls the provider's computer.

It is frequently also possible to copy the password from a configuration file in the user's computer and use it to read his mail from a different computer. As a result of this broad distribution of the e-mail in plain text form and the weakness of password protection, standard e-mail is regarded as very insecure.

To counter this problem, the security system known as P.G.P. (Pretty Good Privacy) was devised. To use P.G.P., a user generates a complete RSA key set containing both a public and private component. He makes his public key widely available by putting it in the signature block of all his e-mail messages and arranging to have it posted in publicly accessible directories of P.G.P. public keys. He stores his private key on his own personal computer, perhaps in a password-protected form. When someone wishes to send private e-mail to this user, he generates a random IDEA encryption key and encrypts the entire message with the IDEA encryption algorithm. He then encrypts the IDEA key itself using the public key provided by the intended recipient. He e-mails both the message encrypted with IDEA and the IDEA key encrypted with the user's public key to the user. No one that sees this transmission can read it except the intended recipient because the message is encrypted with IDEA and the IDEA key is encrypted with the intended recipient's public key. The recipient's computer contains the corresponding private key, and hence can decrypt the IDEA key and use the decrypted IDEA key to decrypt the message. This provides security from those who might try to read the user's mail remotely, but it is less effective when the user's computer is accessible to others because the computer, itself, contains the private key. Even if the private key is password protected, it is often easy to guess the user's password or eavesdrop on him when he enters it, so the user's computer provides little security. In addition, the user can receive secure e-mail only at his own computer because his private key is stored in that computer and is not available elsewhere. Therefore, the weakness of P.G.P. is that it is tied strongly to the user's computer where the private key resides.

2. Module Protected E-Mail

With the exemplary module 10 being used to protect e-mail, a user could have his e-mail forwarded to him wherever he goes without fear that it would be read by others or that his PC would be the weak link that compromises the security of his mail. The module protected e-mail system is similar to the P.G.P. system, except that the private key used for decrypting the IDEA key is stored in a privatized object in a transaction group of the module 10 instead of in a PC. The module protected e-mail system operates as follows:

a. Referring to FIGS. 2, 11 and 12, the user creates a transaction group 40, S1, generates an RSA key set S2 and loads it into three objects 42 of the transaction group 40 (one RSA modulus object, N, and two RSA exponent objects, E and D). He then privatizes the decryption exponent S3, D. Finally, he creates a transaction script 44, S4 to take data placed in the input data object, encrypt it with the modulus N and private exponent D and place the result in the output data object. He locks the group S5 to prevent any additional transaction scripts 44 from being added. He "forgets" the value of D and publishes the values of E and N in public directories and in the signature blocks of his e-mail messages. Since he has forgotten D and since the D exponent object has been privatized, there is no way that anyone will ever find out the value of D.

b. Referring to FIG. 3, to send secure e-mail to the user, the P.G.P. system is used. When the user receives the secure e-mail A1, he transmits the encrypted IDEA key into the input data object of the transaction group 40, A2 and then calls the transaction script 44 to decrypt this key A3 and place the decrypted result in the output data object A4. He then reads the decrypted IDEA key from the output data object and uses it to decrypt his mail A5. Note that it is now impossible for anyone, including the user, to read any new mail without having physical possession of the module 10. There is therefore no way that a user's mail can be read without his knowledge, because the module 10 must be physically present on the computer where the mail is read. The user can carry his module 10 wherever he goes and use it to read his forwarded mail anywhere. His home computer is not the weak point in the security system.

Secure e-mail, as described above, is the simplest possible module application, requiring only one RSA key and one transaction script 44. It is unnecessary even to store the public key E in the module 10, but it is a good idea to do so because the public key is supposed to be publicly accessible. By storing E in an exponent object and not privatizing that object or the modulus object, N, the user insures that the public key can always be read from the module 10. There are no transaction scripts 44 involving E because the module 10 will never be required to perform an encryption.

B. Digital Notary Service

This section describes a preferred notary service using the module 10.

1. Background of a Standard Notary Service

A conventional Notary Service Provider receives and examines a document from an End User and then supplies an uncounterfeitable mark on the document signifying that the document was presented to the notary on a certain date, etc. One application of such a notary service could be to record disclosures of new inventions so that the priority of the invention can later be established in court if necessary. In this case, the most important service provided by the notary is to certify that the disclosure existed in the possession of the inventor on a certain date. (The traditional method for establishing priority is the use of a lab notebook in which inventors and witnesses sign and date disclosures of significant inventions.)

2. Electronic Notary Service Using the Module

A company, hereafter referred to as the Service Provider, decides to go into business to supply a notary service (strictly, a priority verification service) for its customers, hereafter referred to as the End Users. The Service Provider chooses to do this by using the module 10 as its "agents" and gives them the authority to authenticate (date and sign) documents on his behalf. The preferred operation of this system is as follows:

a. Referring to FIGS. 4, 11 and 12, the Service Provider creates a transaction group 40 for performing electronic notary functions in a "registered lot" of modules 10, B1.

b. The Service Provider uses a secure computing facility to generate an RSA key set and program the set into every module 10 as a set of three objects 42, a modulus object and two exponent objects B2. The public part of the key set is made known as widely as possible, and the private part is forgotten completely by the Service Provider. The private exponent object is privatized to prevent it from being read back from the modules

c. The Service Provider reads the real-time clock 14 from each module 10 and creates a clock offset object that contains the difference between the reading of the real-time clock 14 and some convenient reference time (e.g., 12:00 a.m. Jan. 1, 1970). The true time can then be obtained from any module 10 by adding the value of the clock offset object to the real-time clock B3.

d. The Service Provider creates a transaction sequence counter object initialized to zero B4.

e. The Service Provider creates a transaction script 44 which appends the contents of the input data object to the true time (sum of real-time clock 14 and the value of the clock offset object) followed by the value of the transaction counter followed by the unique lasered registration number. The transaction script 44 then specifies that all of this data be encrypted with the private key and placed in the output data object. The instructions to perform this operation are stored in the transaction group 40 as a transaction script object B5.

f. The Service Provider privatizes any other objects 42 that it does not wish to make directly readable or writable B6.

g. The Service Provider locks the transaction group 40, preventing any additional transaction scripts 44 from being added B7.

h. Referring to FIG. 5, now the Service Provider distributes the modules to paying customers (End Users) to use for notary services. Anytime an End User wishes to have a document certified, the End User performs the Secure Hash Algorithm (Specified in the Secure Hash Standard, FIPS Pub. 180) to reduce the entire document to a 20 byte message digest. The End User then transmits the 20 byte message digest to the input data object C1 and calls on the transaction script 44 to bind the message digest with the true time, transaction counter, and unique lasered serial number and to sign the resulting packet with the private key C2.

i. The End User checks the certificate by decrypting it with the public key and checking the message digest, true time stamp, etc. to make sure they are correct C3. The End User then stores this digital certificate along with the original copy of the document in digital form C4. The Service Provider will attest to the authenticity of the certificates produced by its modules.

j. After a period of time specified by the Service Provider, the user returns his module 10, pays a fee, and gets a new module containing a new private key. The old modules can be recycled by erasing the entire transaction group and reprogramming them. The Service Provider maintains an archive of all the public keys it has ever used so that it can testify as needed to the authenticity of old certificates.

C. Digital Cash Dispenser

This exemplary usage model focuses on the module 10 as a cash reservoir from which payments can be made for goods or services. (To simplify the discussion, the subject of refilling the module 10 with cash is postponed until later). In this case the Service Provider is a bank or other financial institution, the End User is the bank's customer who wishes to use the module 10 to make purchases, and the Merchant is the provider of the purchased goods or services. The roles of the Service Provider, the Merchant, and the End User in these transactions are explained in detail below.

The fundamental concept of the digital cash purse as implemented in the module 10 is that the module 10 initially contains a locked money object containing a given cash value, and the module 10 can generate, on demand, certificates which are essentially signed documents attesting to the fact that the amount of money requested was subtracted from the value of the money object. These signed documents are equivalent to cash, since they attest to the fact that the internal money object was decreased in value by an amount corresponding to the value of the certificate. The merchant can redeem these certificates for cash by returning them to the Service Provider.

When dealing with digital certificates representing cash, "replay" or duplication is a fundamental problem. Since digital data can be copied and retransmitted easily, it differs from ordinary coins or paper money which are difficult to reproduce because of the special technology that is used in their manufacture. For this reason, the receiver of the payment must take special steps to insure that the digital certificate he receives is not a replay of some previously issued certificate. This problem can be solved by having the payee generate a random "SALT", a challenge number, and provide it to the payer.

SALT is a method of preventing replay. A random number is sent and used in a challenge/response mode. The other party is challenged to return the random number as part of their response.

The payer constructs a signed certificate which includes both the money amount and the payee's SALT. When the payee receives this certificate, he decrypts it with the public key, checks the money amount, and then confirms that the SALT is the same as the one he provided. By personalizing the certificate to the payee, the payer proves to the payee that the certificate is not a duplicate or replay and is therefore authentic. This method can be used regardless of whether the module 10 is the payer or the payee.

Another problem that must be addressed is irrepudiability. This means that none of the parties to the transaction should be able to argue that he did not actually participate in the transaction. The transaction record (money certificate) should contain elements to prove that each party to the transaction was a willing participant.

1. Background Conventional Cash Transactions

In a conventional cash transaction, the End User first receives Federal Reserve Notes from a bank and the bank subtracts the equivalent amount of money from the balance in his account. The End User can verify the authenticity of the Federal Reserve Notes by means of the "public key", which includes:

a. Magnetic ink attracted by a magnet.

b. Red and blue threads imbedded in the paper.

c. Microfine printing surrounding the engraved portrait.

d. Embedded stripe printed with USA and denomination of the note.

The "private key" to this system is the details of how the raw materials for printing money are obtained and how the money is actually printed. This information is retained by the government and not revealed.

These notes are carried by the End User to the Merchant, where they are exchanged for goods or services. The Merchant also uses the "public key" of the notes to verify that they are legitimate.

Finally, the Merchant carries the notes to a Bank, where the "public key" is again examined by the teller. If the notes are legitimate, the Merchant's bank account balance is increased by the face value of the notes.

The end result of this transaction is that the End User's bank balance is reduced, the Merchant's bank balance is increased by the same amount, the goods or services are transferred from the Merchant to the End User, and the Federal Reserve Notes are ready to be reused for some other transaction.

2. Exemplary Monetary Transactions Using the Module

Monetary transactions using the module **10** and digital certificates are somewhat more complicated because digital data, unlike Federal Reserve Notes, can be copied and duplicated easily. Nevertheless, the use of "SALTs" and transaction sequence numbers can guarantee the authenticity of digital certificates. (In the following discussion, it is assumed that every party to the transaction has its own RSA key set with a private key that it is able to keep secret.)

a. Referring to FIG. **6**, the Service Provider (bank) prepares the module **10** by creating a transaction group **40** containing a money object representing the monetary value stored in the module **10**. The Service Provider also creates a transaction count object, a modulus object, and an exponent object and stores the provider's private key in the exponent object D1. He privatizes the key so that it cannot be read D2. Next, he stores a transaction script **44** in the transaction group **40** to perform the monetary transaction and locks the group so that no further objects can be made D3, D4. (The details of what this transaction script does are described further below.) Finally, he publishes the corresponding public key widely so that anyone can obtain it D5.

b. The End User receives the module **10** from the Service Provider, and the End User's bank account is debited by the amount stored in the module **10**. Using a PC or handheld computer, the End User can interrogate the module **10** to verify that the balance is correct.

C. Referring to FIG. **7**, when the End User wishes to purchase some goods or services from a Merchant E1, the Merchant reads the unique lasered registration number of the module and places it in a packet along with a random SALT E2, E3. The merchant then signs this packet with the merchant's own private key E4 and transmits the resulting encrypted packet along with the amount of the purchase to the input data object of the transaction group **40**, E5.

d. The Merchant then invokes the transaction script **44** programmed into the module **10** by the Service Provider. This transaction script **44** subtracts the amount of the purchase from the money object E6, appends the value of the transaction counter object to the contents of the input data object E7, signs the resulting packet with the private key, and places the result in the output data object E8.

e. The Merchant then reads the result from the output data object and decrypts it with the Service Provider's public key E9. He then confirms that the amount of the purchase is correct and that the remaining data is identical to the packet he signed in step c., E10.

f. Having confirmed that the certificate provided by the module **10** is both authentic and original (not a duplicate), the Merchant delivers the goods or services E11. Later the Merchant sends the digital certificate to a bank.

g. The bank decrypts the certificate with the Service Provider's public key E12, extracts the amount of the purchase and the transaction count, and decrypts the remaining data with the Merchant's public key to reveal the unique lasered registration number of the module E14. The bank then looks up the module **10** by the unique lasered registration number in a database to confirm that the transaction count for this transaction has not been submitted before. When this test is passed, the bank adds the transaction count value to the database, and then increases the Merchant's bank balance by the amount of the purchase E15. The fact that portions of the certificate were signed by both the module **10** and the Merchant confirms that the transaction was freely agreed to by both the Merchant and the module **10**.

Note that there are many different ways of combining data combinations of the transaction counter value, the unique lasered registration number, the random SALT provided by payee, and the amount of purchase, encrypted by the module's private key, the Merchant's private key, or both. Many of these combinations can also provide satisfactory guarantees of uniqueness, authenticity, and irrepudiability, and the design of the firmware allows the Service Provider flexibility in writing the transaction script **44** to serve his particular needs.

D. Digital Cash Replenishment

The discussion of a digital cash purse is section II.C., above, did not address the issue of cash replenishment. The Service Provider can add cash replenishment capability to the module **10**, as discussed in section II.C., simply by adding another modulus object and exponent object containing the Service Provider's public key, a random SALT object, and a transaction script **44** for adding money to the balance. The Service Provider can add money to a module **10** either in person or remotely over a network. The process of adding money is as follows:

1. Referring to FIG. **8**, the Service Provider reads the unique lasered registration number (ID number) of the module F1, F2 and calls on a transaction script **44** to return the value of a random SALT object. The module **10** calculates a new random SALT value from the previous value and the random number generator and returns it to the Service Provider F3.

2. The Service Provider places the random SALT returned by the module **10** in a packet along with the amount of money to be added and the unique lasered registration number of the module **10** and then encrypts the resulting packet with the Service Provider's private key F4. This encrypted packet is then written back into the input data object of the transaction group **40**.

3. The Service Provider invokes a transaction script **44** which decrypts the contents of the input data object with the Service Provider's public key and then checks the unique lasered registration number and the value of the random SALT against the one that it originally provided. If the SALT matches, the money amount is extracted from the packet and added to the value of the money object in the module F5.

Note that the inclusion of the unique lasered registration number is not strictly necessary, but it is included to insure that the Service Provider knows exactly which module is receiving the funds.

E. Exemplary Description of Direct Transfer of Funds Between Modules

Section II.C.2.g. above reveals a problem that occurs when the Merchant returns the digital certificates to his bank for crediting to his account. The Merchant's bank must either send the certificates back to the Service Provider for redemption, or have access to the Service Provider's records in a database so that it can determine whether the value of the transaction count object is unique. This is inconvenient and requires infrastructure. It also prevents any of the transactions from being anonymous (as they would have been if cash had been used), because the Merchant's bank must log used certificate numbers into a database to prevent them from being reused. These problems can all be eliminated by making use of fund transfers between modules. In addition, the steps required to accomplish a fund transfer between modules are considerably simpler than those described in section II.C.2.

In the discussion which follows, it is assumed that the Merchant also has a module which he uses to collect the funds received from End Users (customers). The module in the possession of the End User will be called the Payer, and the module in the possession of the Merchant will be called the Payee. The steps to accomplish the funds transfer are as follows:

1. Referring to FIGS. 9, 11 and 12, using his computer, the Merchant calls on a transaction script 44 in the Payee to provide a random SALT. He reads this SALT from the output object of the transaction group 40.

2. The Merchant copies the SALT and the amount of the End User's purchase to the input data object of the Payer G1, then calls on a transaction script 44 in the Payer to subtract the amount of the purchase from the balance, combine the Payee's SALT in a packet with the amount of the purchase, encrypt the resulting package with the Service Provider's private key, and return it in the output data object G2.

3. The Merchant then reads this packet and copies it to the input data object of the Payee, then calls on a transaction script 44 in the Payee to decrypt the packet with the Service Provider's public key G3 and check the SALT against the one originally generated by the Payee. If they agree, the Payee adds the amount of the purchase to its balance G4.

This completes the funds transfer. Note that this transaction effectively transferred the amount of the purchase from the Payer to the Payee, and the steps of the transaction were much simpler than the three-way transaction described in II.C.2. The Merchant can transfer the balance to his bank account by a similar transaction in which the bank provides a SALT to Merchant's module and the Merchant's module prepares a certificate for the balance which it delivers to the bank. Use of a module by the Merchant to collect funds simplifies the transaction, eliminates the need for a database to confirm uniqueness, and preserves the anonymity of the End User that would normally result from a cash transaction.

F. Exemplary Transactions With a Module Over a Network

The transactions described in section II.C.2., II.D. and II.E. above could also be performed over a network, allowing a physical separation between the Merchant, End User, and modules. However, this could produce a potential problem because one of the communications to the module 10 is unencrypted and therefore subject to falsification. To avoid this problem, both parties must produce a SALT so that the other can demonstrate its ability to encrypt the SALT with the Service Provider's private key and therefore prove authenticity. The operation of this protocol is described as follows as it relates to the transfer of funds between modules (section II.E. above). This method can be employed to allow any of the transactions described above to take place over a network. This clearly enables secure electronic commerce over the Internet.

1. Referring to FIG. 10, 11 and 12, the Payer generates a random SALT and transmits it over the network to the Payee H1.

2. The Payee appends the amount of the purchase to the Payer's SALT, followed by a SALT randomly generated by the Payee. The Payee then encrypts this packet with the Service Provider's private key and sends it back to the Payer H2.

3. The Payer decrypts the packet with the Service Provider's public key H3, extracts the Payer SALT, and compares it with the SALT that the Payer provided in step 1. If they agree, the Payer subtracts the amount of the purchaser from its balance H4 and generates a certificate consisting of the amount of the purchase and the Payee's SALT, which it encrypts with the Service Provider's private key and returns to the Payee H5.

4. The Payee decrypts the packet with the Service Provider's public key H6, extracts the Payee SALT, and compares it with the SALT that the Payee provided in step 2. If they agree, the Payee adds the amount of the purchase to its balance H7.

The exchange of SALTs allows each module to confirm that it is communicating with another module, and that the funds transfer requested is therefore legitimate. The SALT comparison described in step 3 allows the Payer to confirm that the Payee is a legitimate module 10 before the funds are withdrawn, and the comparison described in step 4 allows the Payee to confirm that the Payer is a legitimate module 10 before the funds are deposited. The transactions described above provide the minimum necessary information in the encrypted packets to confirm that the funds are being transferred from one module 10 to another. Other information, such as the unique lasered registration number, could be included (at the cost of anonymity) to provide additional information and greater control over the transaction.

G. An Exemplary Technique for Software Authorization and Usage Metering

The module 10 is well-suited for the tasks of enabling specific software features in a comprehensive software system and for metering usage of those features. (This usage model parallels the previously described model for withdrawing money from a module 10.)

1. Preparation

Referring to FIGS. 11 and 12, the Service Provider creates a transaction group 40 and stores a configuration object in the group detailing which software within the module 10 the End User is allowed to use. The Service Provider also creates a money object containing the allowed usage credit (which could be in units of time rather than the actual dollar amount), and stores and privatizes a private RSA key pair to use for authentication. A transaction script 44 is stored to receive a SALT and the amount to withdraw from the End User, decrement the balance by the amount withdrawn, and output an RSA signed certificate containing the amount withdrawn, the sale, and the value of the configuration object.

2. Usage

At periodic intervals during the use of the software within the module 10, the PC program generates a random SALT and an amount to charge for the use of the module 10 and transmits this information to the module 10. The module 10 decrements the balance and returns the certificate. The PC decrypts the certificate and confirms that the SALT is the same, the amount withdrawn is correct, and the use of the software within the module 10 is authorized by the information stored in the configuration object. If all of these tests are successful, the module 10 executes for a specified period of time or for a given number of operations before asking the module 10 for another certificate.

There are many possible variations on this usage model. For example, the transaction script 44 could also bind up the

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | **sales@docketalarm.com** | **1-866-77-FASTCASE**

fastcase®
Smarter legal research.