US007383442B2

(12) **United States Patent**
Schell et al.

(10) **Patent No.: US 7,383,442 B2**
(45) **Date of Patent: *Jun. 3, 2008**

(54) **NESTED STRONG LOADER APPARATUS AND METHOD**

(75) Inventors: **Roger R. Schell**, Orem, UT (US); **Kevin W. Kingdon**, San Mateo, CA (US); **Thomas A. Berson**, Palo Alto, CA (US)

(73) Assignee: **Novell, Inc.**, Provo, UT (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 941 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **10/279,517**

(22) Filed: **Oct. 24, 2002**

(65) **Prior Publication Data**

US 2003/0061483 A1 Mar. 27, 2003

Related U.S. Application Data

(63) Continuation of application No. 09/274,532, filed on Mar. 23, 1999, now Pat. No. 6,532,451.

(60) Provisional application No. 60/079,133, filed on Mar. 23, 1998.

(51) **Int. Cl.**

G06F 11/30	(2006.01)
G06F 12/14	(2006.01)
H04K 9/32	(2006.01)
H04L 9/14	(2006.01)
H04L 9/00	(2006.01)
H04K 9/00	(2006.01)

(52) **U.S. Cl.** **713/188**; 713/180; 713/164; 380/3; 380/4; 380/25; 380/277

(58) **Field of Classification Search** ..... None
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,458,315 A 7/1984 Uchenick

(Continued)

FOREIGN PATENT DOCUMENTS

WO WO-00/42730 7/2000

(Continued)

OTHER PUBLICATIONS

"API's and Toolkits", *IBM*, (1996),1-2.

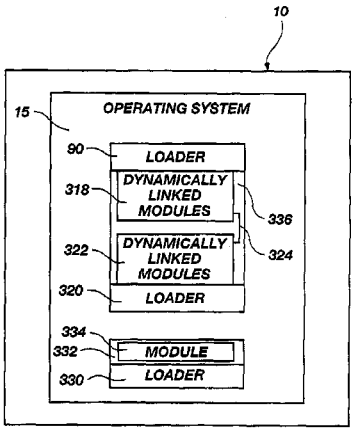
(Continued)

Primary Examiner—Ayaz Sheikh
Assistant Examiner—Arezoo Sherkat
(74) *Attorney, Agent, or Firm*—Schwegman, Lundberg & Woessner, P.A.

(57) **ABSTRACT**

An apparatus and method provides one or more controlled, dynamically loaded, modular, cryptographic fillers. Fillers may be loaded by a single loader, multiple independent loaders, or nested loaders. Loaders may be adapted to load other loaders, within cryptographic controls extant and applicable thereto. Integration into a base executable having one or more slots, minimizes, controls, and links the interface between the fillers and base executables. The filler may itself operate recursively to load another filler in nested operations, whether or not the fillers are in nested relation to one another. An ability of any filler to be loaded may be controlled by the base executable verifying the integrity, authorization, or both for any filler. The base executable may rely on an integrated loader to control loading and linking of fillers and submodules. A policy may limit each module function, access, and potential for modification or substitution. Dynamically loaded modules (loaders, other fillers, and submodules thereof), typically represent a relatively small portion of the overall coding required by the base executable, and may provide strong controls limiting integration by providing access that is nested, layered, or both between modules excluding direct access to or by them from the base executable or supported applications.

20 Claims, 7 Drawing Sheets



U.S. PATENT DOCUMENTS

4,723,284	A		2/1988	Munck et al.
4,870,681	A		9/1989	Sedlak
4,918,728	A		4/1990	Matyas et al.
4,937,863	A		6/1990	Robert et al.
4,941,176	A		7/1990	Matyas et al.
5,007,089	A		4/1991	Matyas et al.
5,142,578	A		8/1992	Matyas et al.
5,164,988	A		11/1992	Matyas et al.
5,200,999	A		4/1993	Matyas et al.
5,201,000	A		4/1993	Matyas et al.
5,249,230	A		9/1993	Mihm, Jr.
5,265,164	A		11/1993	Matyas et al.
5,280,529	A		1/1994	Nost
5,299,263	A		3/1994	Beller et al.
5,337,360	A		8/1994	Fischer
5,386,471	A		1/1995	Bianco
5,390,247	A		2/1995	Fischer
5,406,628	A		4/1995	Beller et al.
5,412,717	A		5/1995	Fischer
5,432,849	A		7/1995	Johnson et al.
5,495,533	A		2/1996	Linehan et al.
5,689,565	A		11/1997	Spies et al.
5,721,777	A		2/1998	Blaze
5,724,423	A		3/1998	Khello
5,933,503	A		8/1999	Schell et al.
5,968,174	A	*	10/1999	Hughes 713/2
5,970,145	A	*	10/1999	McManis 713/187
5,970,252	A	*	10/1999	Buxton et al. 717/166
6,195,794	B1	*	2/2001	Buxton 717/108
2001/0005885	A1	*	6/2001	Elgamal et al. 713/164

OTHER PUBLICATIONS

"Applications", *IBM*, (1996),1-2.
"Conclusion", *IBM*, (1996),1-3.
"Cryptographic Engines", *IBM*, (1996),1-2.
"Gore to Unveil Encryption Policy", *The Net*, (1996),1-2.
"High-Tech Leaders Join Forces to Enable International Strong Encryption", *IBM*, (1996),1-3.
"IBM (International Business Machines Corp) to Form Consortium for Data Encryption", *Yahoo*, (Oct. 1, 1996),1-2.
"Making PC Interaction Trustworthy for Communications, Commerce and Content", *Intel Security Program*, (Jul. 1998),1-6.
"Method for Ensuring Integrity of Public Key Algorithm Public and Private Keys and for Coupling the usage of a Key to the Correct Specification of the key's Associated Control Vector", *SPI Database of Software Technologies*, (Jun. 1994),1 page.
"Secure Way Cytographic Infrastructure", *IBM*, (1996),1-3.
"Security Prevails in e-mail coding", (before Jul. 8, 1999),1-2.
"Solaris Manpage for Intro(1M)(maintenance Commands)", *SPI Database of Software Technologies*, (1993),4-7.
"SunOS Manpage for INTRO(4)", *SPI Database of Software Technologies*, (Apr. 29, 1992),14-15.
"SunOS Manpage for INTRO(8)", *SPI Database of Software Technologies*, (May 22, 1991),7-11.
"Supporting Services and Sub-Systems", *IBM*, (1996),1-4.
"The IBM Secure Way Cryptographic Infrastructure", *IBM*, (1996),1-2.
"Two-Level Data Security System for an IBM Personal Computer", *SPI Database of Software Technologies*, (Mar. 1987),11-12.
"White House Encryption Initiative-2: IBM, Digital Support", *Dow Jones International News*, (Oct. 1, 1996),1-3.
Lipschutz, Robert.P. ,"Netscape servers fill the communication gag", *PC Magazine Network Edition* vol. 15, No. 21, pNE19-pNE24,(Dec. 3, 1996),4 pages.
Rajan,et al. ,"Mechanics of the Common Security Services Manager (CSSM)", *Intel Corporation*, (1999),1-17.

* cited by examiner

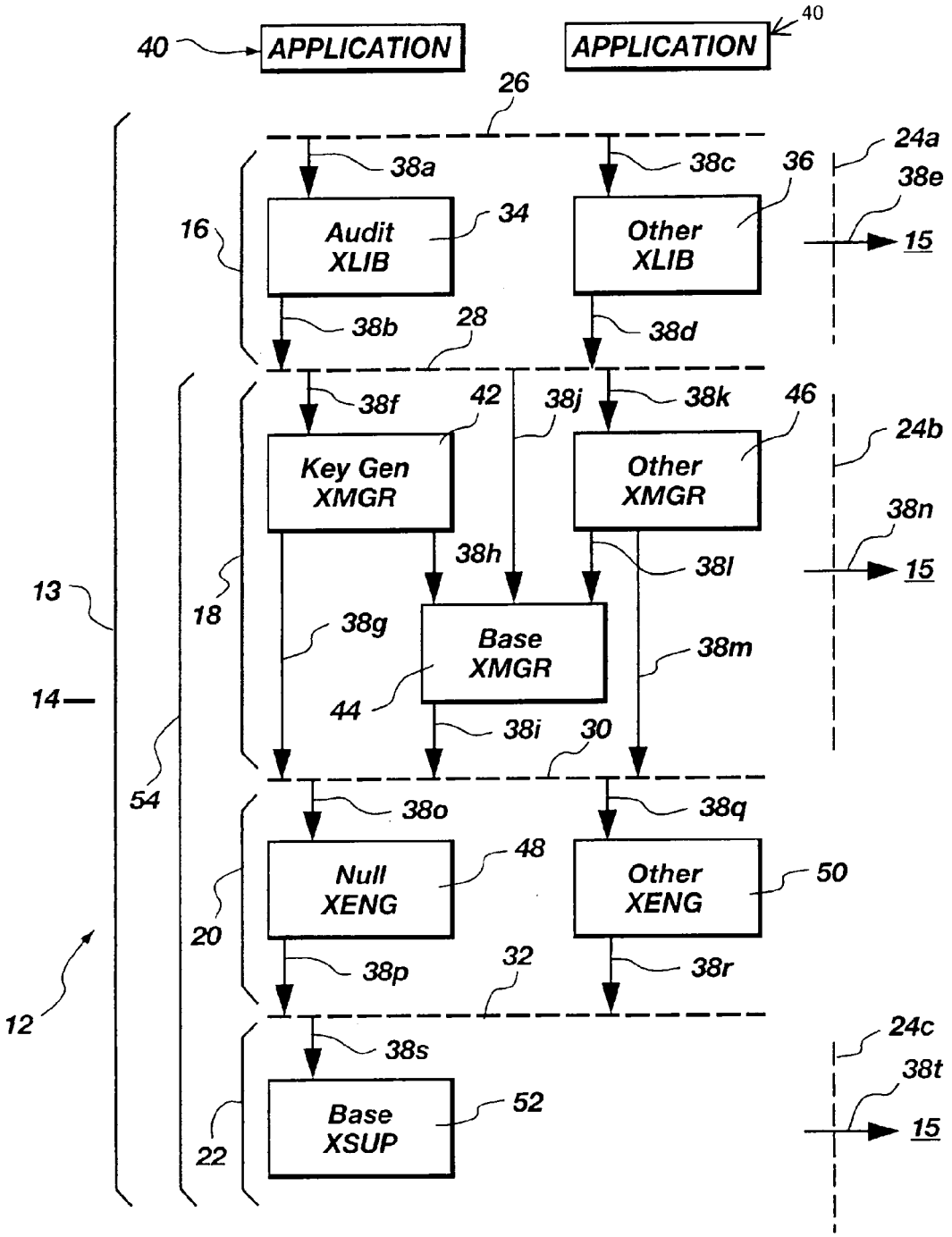

40
APPLICATION
APPLICATION
40
26
24a
38a
38c
36
38e
16
Audit XLIB
34
Other XLIB
15
38b
28
38d
38f
42
38j
38k
46
24b
Key Gen XMGR
Other XMGR
38n
38h
38l
15
13
18
38g
Base XMGR
38m
14
44
38i
30
54
38o
38q
48
50
Null XENG
Other XENG
20
38p
32
38r
12
24c
38s
38t
Base XSUP
52
15
22

Fig. 1

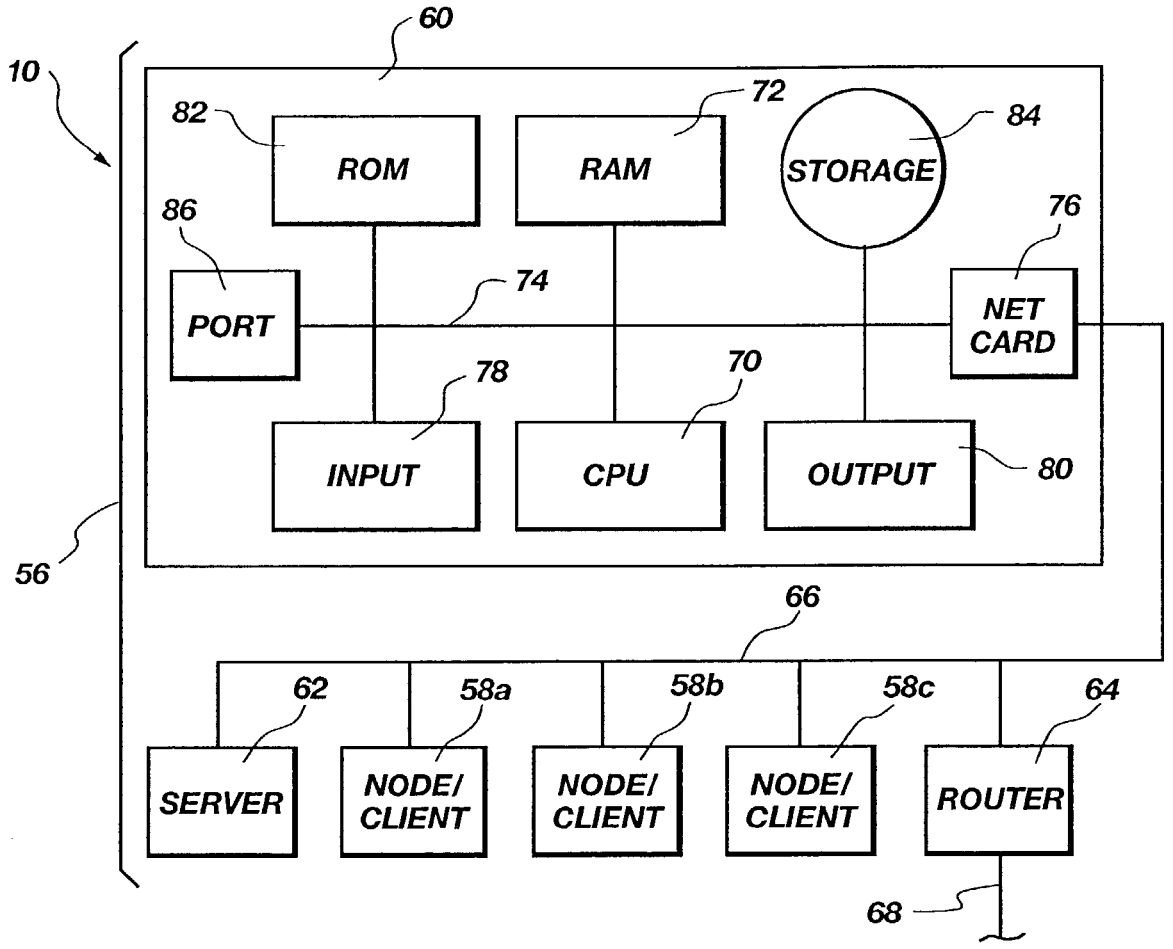

10
60
82
ROM
72
RAM
84
STORAGE
86
PORT
74
76
NET CARD
78
INPUT
70
CPU
OUTPUT
80
56
66
62
SERVER
58a
NODE/ CLIENT
58b
NODE/ CLIENT
58c
NODE/ CLIENT
64
ROUTER
68

Fig. 2

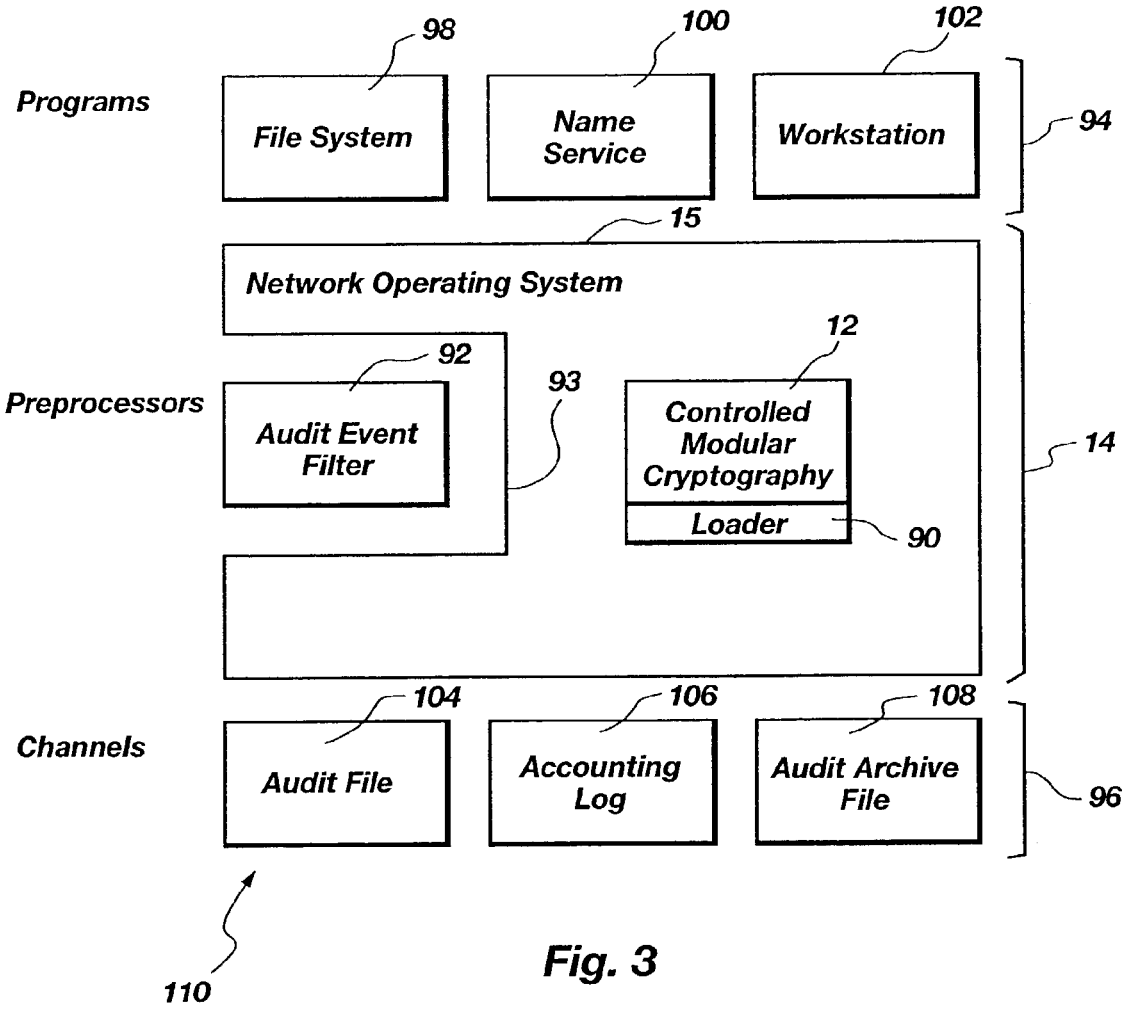

Programs
98
File System
100
Name Service
102
Workstation
94
15
Network Operating System
12
92
Preprocessors
Audit Event Filter
93
Controlled Modular Cryptography
Loader
90
14
104
Channels
Audit File
106
Accounting Log
108
Audit Archive File
96
110

Fig. 3

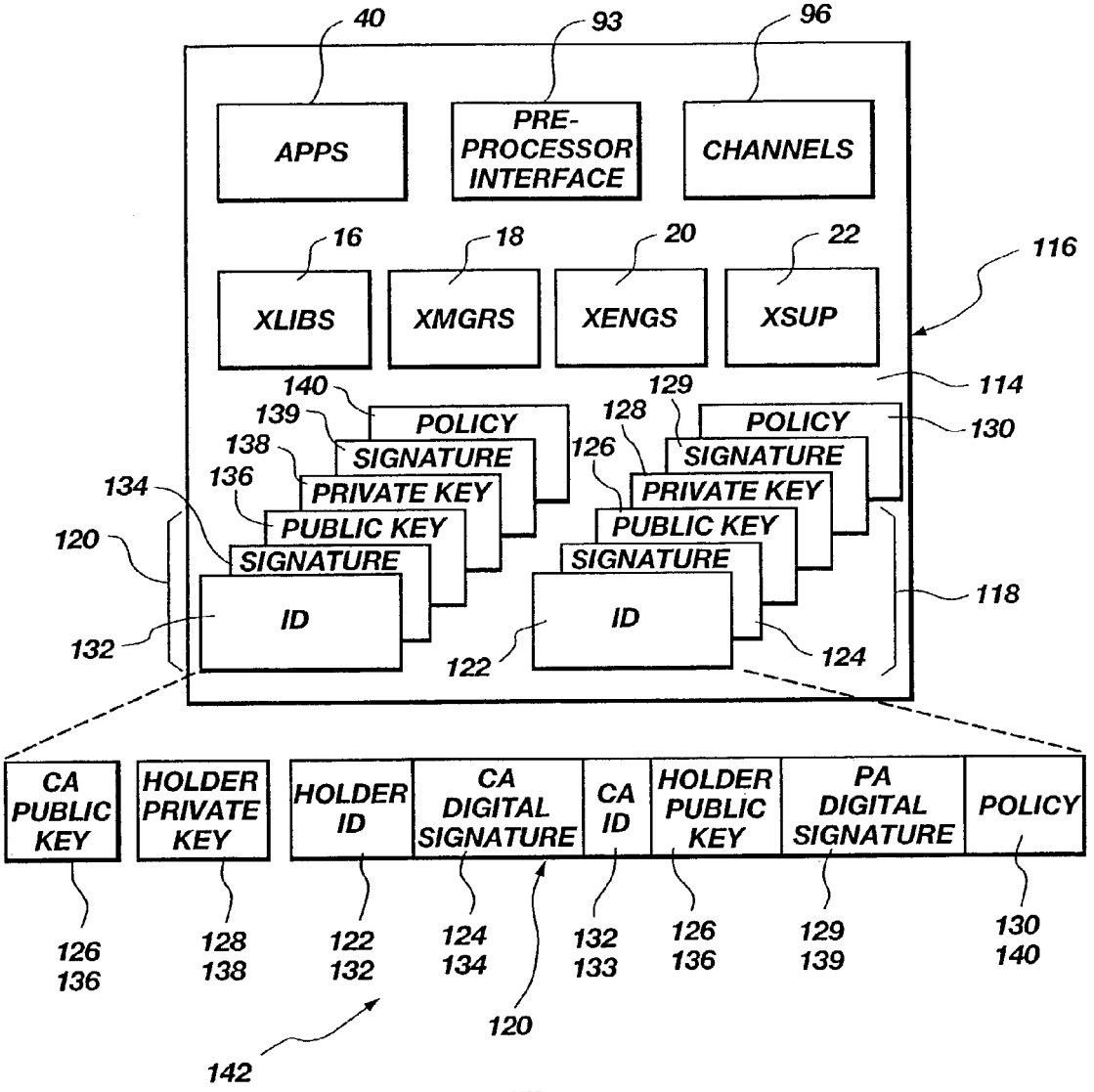

40
93
96
APPS
PRE-PROCESSOR INTERFACE
CHANNELS
16
18
20
22
116
XLIBS
XMGRS
XENGS
XSUP
114
140
139
138
136
134
120
132
POLICY
SIGNATURE
PRIVATE KEY
PUBLIC KEY
SIGNATURE
ID
129
128
126
POLICY
SIGNATURE
PRIVATE KEY
PUBLIC KEY
SIGNATURE
ID
130
118
122
124
CA PUBLIC KEY
HOLDER PRIVATE KEY
HOLDER ID
CA DIGITAL SIGNATURE
CA ID
HOLDER PUBLIC KEY
PA DIGITAL SIGNATURE
POLICY
126 136
128 138
122 132
124 134
120
132 133
126 136
129 139
130 140
142

Fig. 4

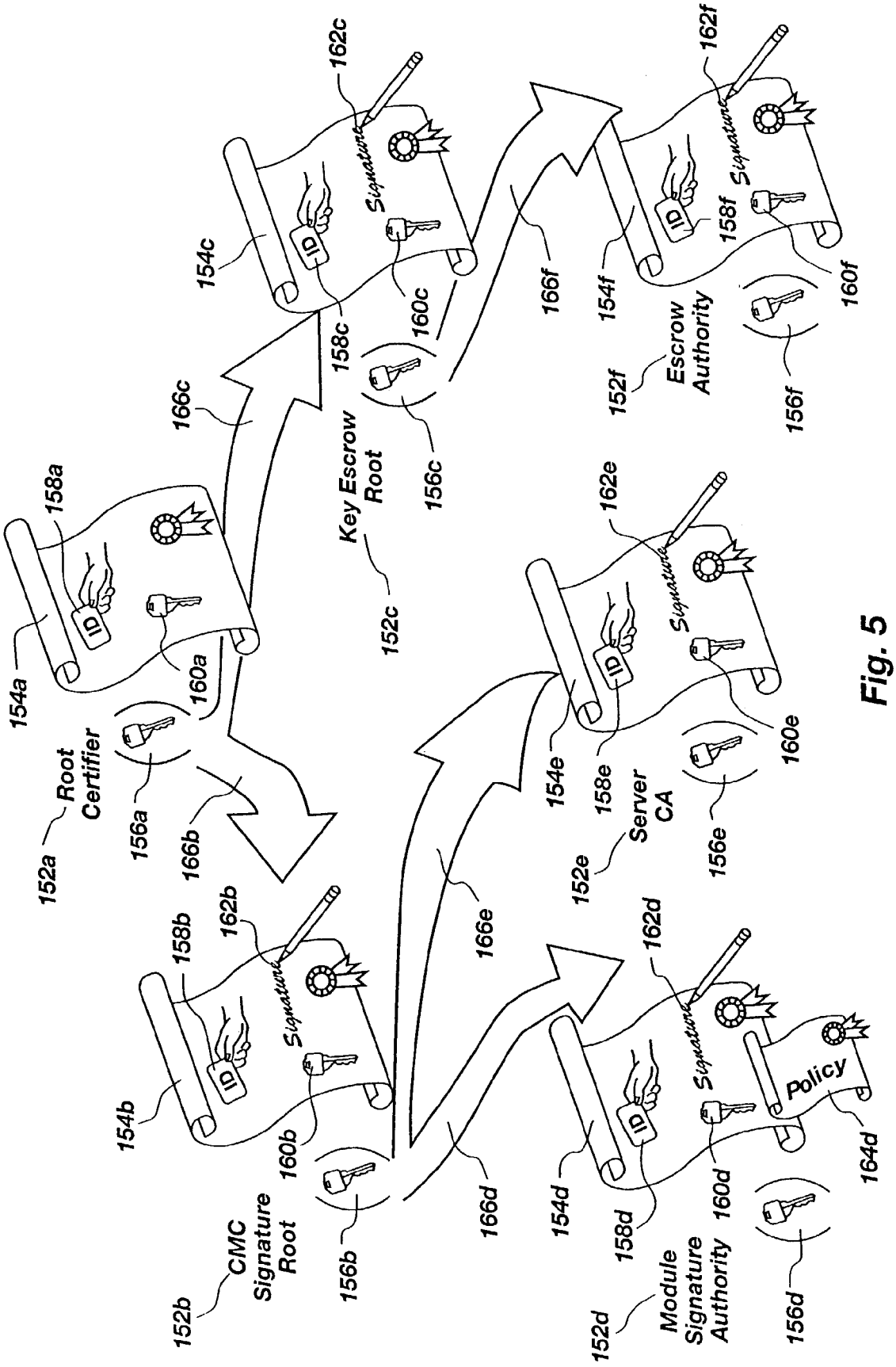
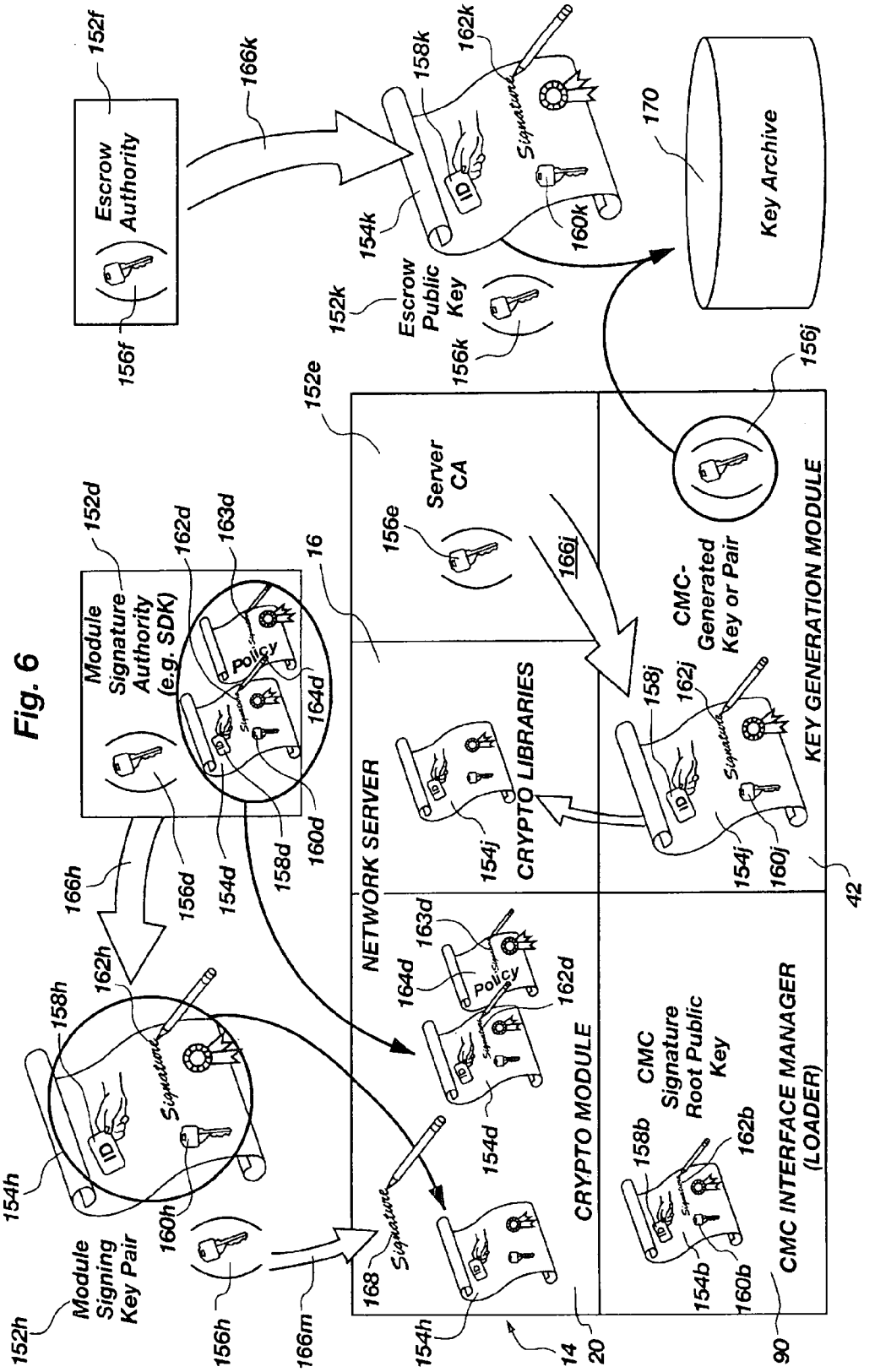

Root Certifier
Key Escrow Root
Escrow Authority
Server CA
CMC Signature Root
Module Signature Authority
Policy
ID
Signature

Fig. 5

Fig. 6


Escrow Authority
152f
156f
166k
158k
162k
154k
160k
Signature
ID
Policy
Escrow Public Key
152k
156k
170
Key Archive
152e
Server CA
156e
166j
156j
CMC-Generated Key or Pair
158j
162j
154j
160j
KEY GENERATION MODULE
42
Module Signature Authority (e.g. SDK)
152d
162d
163d
164d
16
156d
154d
158d
160d
NETWORK SERVER
CRYPTO LIBRARIES
166h
162h
158h
154h
160h
Module Signing Key Pair
152h
156h
166m
168
CRYPTO MODULE
14
20
CMC Signature Root Public Key
158b
162b
154b
160b
CMC INTERFACE MANAGER (LOADER)
90

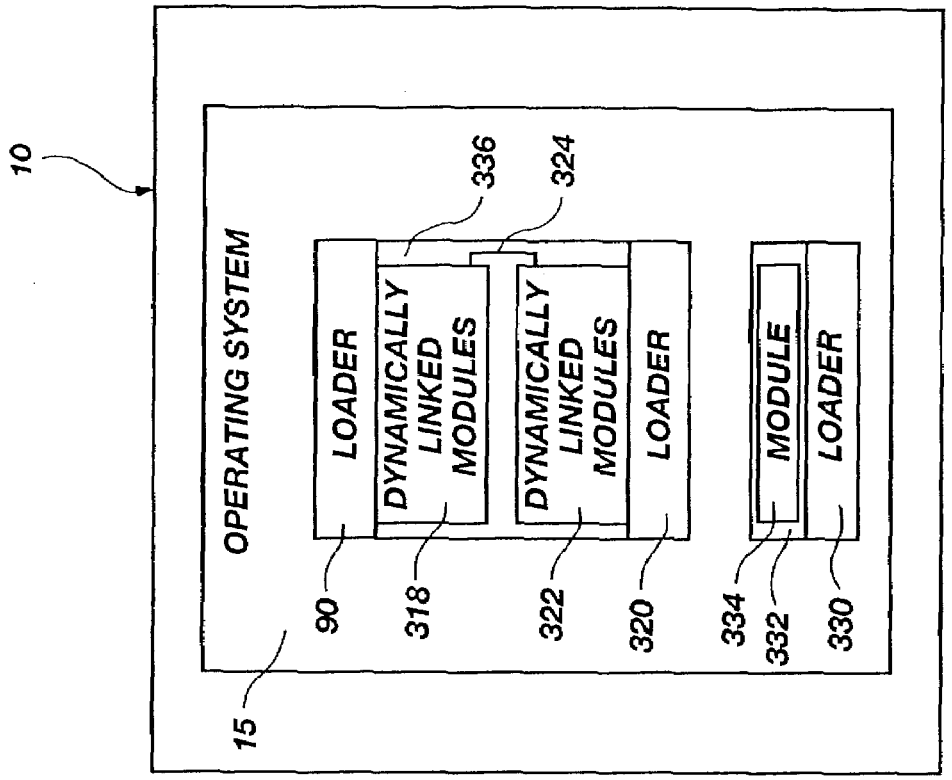

10
OPERATING SYSTEM
LOADER
DYNAMICALLY LINKED MODULES
DYNAMICALLY LINKED MODULES
LOADER
MODULE
LOADER
336
324
90
318
322
320
334
332
330
15

Fig. 8

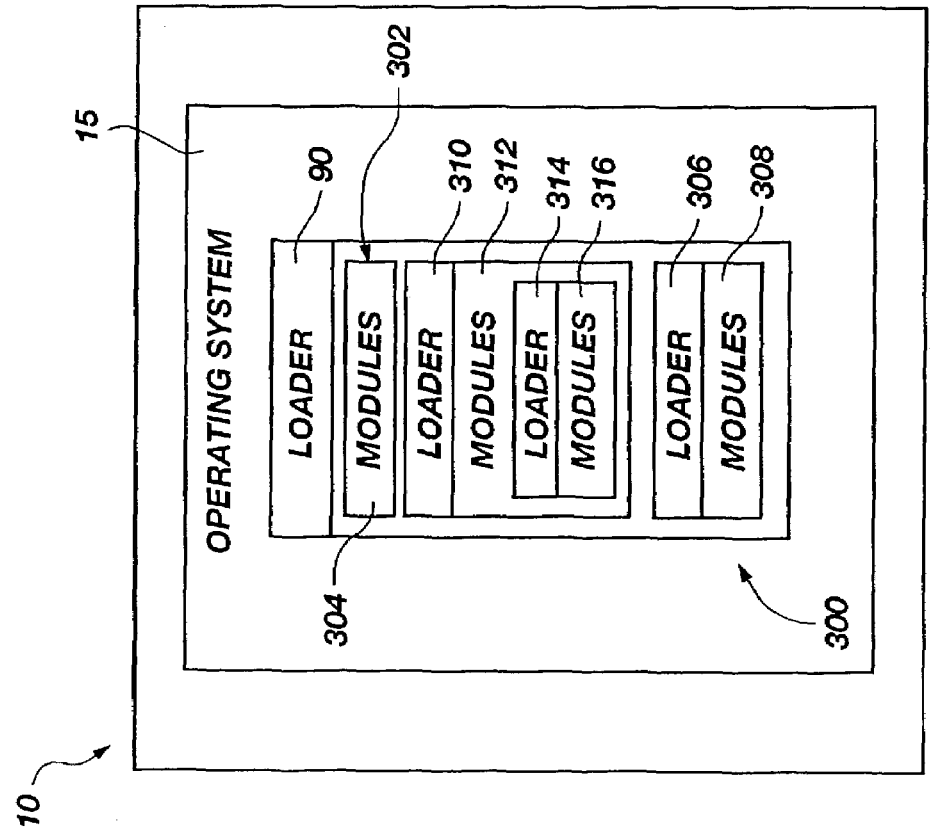

10
15
OPERATING SYSTEM
LOADER
MODULES
LOADER
MODULES
LOADER
MODULES
LOADER
MODULES
302
90
310
312
314
316
306
308
304
300

Fig. 7

NESTED STRONG LOADER APPARATUS AND METHOD

RELATED APPLICATIONS

This application is a continuation of U.S. Ser. No. 09/274,532 filed Mar. 23, 1999 now U.S. Pat. No. 6,532,451 which is a Continuation-In-Part of, and claims priority to, U.S. Provisional Patent Application Ser. No. 60/079,133, filed on Mar. 23, 1998, the specifications of which are incorporated herein by reference.

BACKGROUND

1. The Field of the Invention

The present invention relates to cryptographic systems and methodology. More particularly, the present invention relates to novel apparatus systems and the capability of hosting a plurality of individual modularized methods for allowing encrypting applications within a single use application operating on a computer and to separately control the access, use, and authorization of each of the individual encrypting applications.

2. The Background Art

Encryption is a technology dating from ancient times. In modern times, encryption of military communications has been common. However, since the famous "ENIGMA" machine of World War II, cryptography has been used in numerous functions. One of those functions is special purpose software or applications that may be hosted on computers. Hiding underlying algorithms, limiting access, inhibiting reverse engineering, limiting unauthorized use, controlling licensure, and the like may be legitimate uses of cryptography.

Cryptographic Processes

Modern cryptography protects data transmitted over high-speed electronic lines or stored in computer systems. There are two principal objectives: secrecy, to prevent the unauthorized disclosure of data, and integrity (or authenticity), to prevent the unauthorized modification of data. The process of disguising plaintext data in such a way as to hide its substance is encryption, and the encrypted result is cyphertext. The process of turning cyphertext back into plaintext is decryption.

A cryptographic algorithm, also called a cipher, is the computational function used to perform encryption and/or decryption. Both encryption and decryption are controlled by a cryptographic key or keys. In modern cryptography, all of the security of cryptographic algorithms is based in the key or keys and depends not at all on keeping any details of the algorithms secret.

There are two general types of key-based cryptographic algorithms: symmetric and public-key. Symmetric algorithms (also called secret-key algorithms) are algorithms where the encryption key can be calculated from the decryption key and vice versa (and in fact these keys are usually the same). These require that a sender and receiver agree on these keys before they can protect their communications using encryption. The security of these algorithms rests in the key, and divulging the key allows anyone to encrypt and decrypt data or messages with it.

In public-key algorithms (also called asymmetric algorithms), the keys used for encryption and decryption different from each other in such a way that at least one key computationally infeasible to determine from the other. To ensure secrecy of data or communications, only the decryption key need be kept private, and the encryption key can thus be made public without danger of encrypted data being decipherable by anyone other than the holder of the private decryption key. Conversely, to ensure integrity of data or communications, only the encryption key need be kept private, and a holder of a publicly-exposed decryption key can be assured that any ciphertext that decrypts into meaningful plaintext using this key could only have been encrypted by the holder of the corresponding private key, thus precluding any tampering or corruption of the ciphertext after its encryption.

Most public-key cryptographic algorithms can be used to provide only one of secrecy or integrity but not the other; some algorithms can provide either one but not both. Only the RSA (Rivest, Shamir, and Adleman) public-key algorithm (U.S. Pat. No. 4,405,829), whose security is based on the difficulty of factoring large numbers, has been able to be used to provide both secrecy and integrity.

A private key and a public key may be thought of as functionally reciprocal. Thus, whatever a possessor of one key of a key pair can do, a possessor of the other key of the key pair can undo. The result is that pairwise, secret, protected communication may be available without an exchange of keys. Thus, in general, a receiver, in possession of its own private key may be enabled to use its own copy of a sender's public key, to decrypt data encrypted with a sender s private key corresponding to the sender s public key.

An asymmetric algorithm assumes that public keys are well publicized in an integrity-secure manner. A sender (user of a public key associated with a receiver) can then know that the public key is valid, effective, and untampered with. One way to ensure integrity of data packets is to run data through a cryptographic algorithm. A cryptographic hash algorithm may encrypt and compress selected data. Such hash algorithms are commercially available. For example, the message digest **5** (MD **5**), and the message digest **4** (MD **4**) are commercially available software packages or applications for such functions.

A certificate may be thought of as a data structure containing information or data representing information, associated with assurance of integrity and/or privacy of encrypted data. A certificate binds an identity of a holder to a public key of that holder, and may be signed by a certifying authority. A signature is sometimes spoken of as binding an identity of a holder to a public key in a certificate. As a practical matter, a certificate may be very valuable in determining some level of confidence in keys associated with encryption. That is, just how "good" is an encryption in terms of privacy and integrity? That confidence level may be established by means of a certificate hierarchy. By certificate hierarchy is meant a certification process or series of processes for providing certificates from a trusted authority to another creator of keys.

A certificate, being a data structure, may contain, for example, data regarding the identity of the entity being certified as the holder of the key associated with the certificate, the key held (typically it is a public key), the identity (typically self-authenticating) of the certifying authority issuing the certificate to the holder, and a digital signature, protecting the integrity of the contents of the certificate. A digital signature may typically be based on the private key of the certifying authority issuing the certificate to the holder. Thus, any entity to whom the certificate is asserted may verify the signature corresponding to the private key of the certifying authority.

In general, a signature of a certifying authority is a digital signature. The digital signature associated with a certificate enables a holder of the certificate, and one to whom the certificate is asserted as authority of the holder, to use the signature of the certifying authority to verify that nothing in the certificate has been modified. This verification is accomplished using the certificate authority s public key. This is a means to verify the integrity and authenticity of the certificate and of the public key in the certificate.

Cryptographic Policies

Government authorities throughout the world have interests in controlling the use of cryptographic algorithms and keys. Many nations have specific policies directed to creation use, import, and export of cryptographic devices and software. Numerous policies may exist within a single government. Moreover, these policies are undergoing constant change periodically.

Cryptographic policies may limit markets. For example, a cryptographic algorithm may not be included in software shipped to a country having laws restricting its importation. On the other hand, such a cryptographic device may be desired, highly marketable, and demanded by the market in another country. Thus, generalized software development standardization of software, and the like may become difficult for software vendors. Moreover, users have difficulties attendant with supporting limited installed bases of specialized software. That is, a sufficient installed base is required to assure adequate software.

In short, cryptographic use policies sometimes constrain the set of cryptographic algorithms that may be used in a software system. In addition to restrictions on allowable algorithms, cryptographic use policies may also place constraints on the use and strength keys associated with those algorithms. Software shipped or used in any country must be in compliance with the policies extant.

Another common aspect of certain cryptographic use policies is a requirement that a copy of cryptographic keys be stored or "escrowed" with an appropriate authority. However the mechanisms necessary to satisfy different policies can vary greatly.

Cryptography, especially public key cryptography, provides certain benefits to software designers. U.S. Pat. No. 4,200,700, U.S. Pat. No. 4,218,582, and U.S. Pat. No. 4,405,829 are directed to such technology and are incorporated herein by reference. These benefits are available in situations where data may be shared. Many modem software packages (applications, operating systems, executables) are used in businesses or in other networks where multiple "clients" may share a network, data, applications, and the like. Most modern software packages employ cryptography in some form.

One application for cryptography in network management or network operating systems includes authentication. Also, integrity of data packets transferred, encryption of files, encoding associated with licenses for software or servers, and license distribution or serving are some of the applications for cryptography.

Users may be identified and their rights to access may be authenticated by means of passwords on a network. Cryptography is typically used to transfer some authentication, integrity, verification, or the like in a secure manner across a network that may be open to channel tapping. Public key cryptography is typically used in such a circumstance. Another application of cryptography for authentication involves a single sign-on. For example, a user may need to enter a single password at the beginning of a session. This may remain true regardless of the number of servers that may eventually be called into service by the individual user (client) during this single session. Historically, scripts have been used to provide a single sign-on, but public key mechanisms are now being provided for this function.

Users have previously demonstrated that networks may be subject to attack by spoofing of network control packets. This procedure may be demonstrated in playback and in man-in-the-middle scenarios. By such spoofing, users may obtain unauthorized privileges on a network server. Adding packet signatures, keyed on a per-session basis may provide improved packet integrity.

File encryption is becoming more available. Such encryption has particular use in the special circumstance of audit files. For example, a need exists to protect an audit trail from inspection or modification, or both, by a system administrator, even though the audit trail remains under the system administrator's physical control.

Certain licensing schemes may use various encryption modes to protect software against piracy by end users and others throughout a distribution chain. Data structures cryptography methodologies, checks, and other protection mechanisms may be proprietary to a software developer. Nevertheless, license server mechanisms are being developed to support control of the use of application software in conformity with licenses. Licenses may be provided by an application software provider. The license server may use public key cryptography to create and verify signed data structures. Secret key cryptography may be used to support authentication and file encryption.

Certain applications may provide file confidentiality using proprietary, exportable, secret key algorithms. Users in large numbers make use of such algorithms. Nevertheless, considerable interest in breaking such proprietary algorithms has been successful with certain software. Proprietary encryption methodologies have been consistently broken, given enough time and attention by interested hackers.

Certain applications use public key cryptography for digital signatures. Market leaders in software have provided relatively weak secret key algorithms adopted by others. Thus, files written in different applications from different vendors, even encrypted files, may be opened by an application from any of the vendors using the market leader s secret key algorithm. Within a single product line, a vendor of software applications may use multiple algorithms. Several, if not a plethora of, algorithms exist, including both secret key algorithms and public key algorithms. Stream and block ciphers, as well as hash functions are available and well documented in the computer programming art. Also, certain algorithms are the subject of patent applications which may cloud their broadly based use.

What is needed is a standardized cryptography methodology for distribution across entire product lines. Moreover, encryption technologies are needed for permitting a licensee of a principal software manufacturer to become a third party vendor or value-added distributor capable of producing its own proprietary software, software additions, or preplanned software modules. Currently, software-with-a-hole may provide an operating system with a cryptographic module that fits in the "hole" in an operating system. However, software manufacturers using this technology typically require that a third-party vendor send its product to the principal software manufacturer for integration. The manufacturer may then provide all interfacing and wrapping of the third-party s filler (such as an encryption engine) to fit within the "hole" in the software of the manufacturer.

Also, export restrictions exist for encryption technology. Limiting the strength of exported cryptography is established by statute. To be exportable, such products must meet certain criteria (primarily limitations on key size) that effectively prevent the exportation of strong cryptographic mechanisms usable for data confidentiality. Moreover creating "cryptography with a hole" is undesirable for several reasons, including export and import restrictions. Cryptography with a hole is the presence of components specifically designed or modified to allow introduction of arbitrary cryptographic mechanisms by end users. A great escalation of the difficulty of such introduction, without creating numerous, individually customized software packages, is a major need today, although not necessarily well recognized.

Certain foreign countries have more stringent regulation of the importation of encryption technology by non-government entities. A government may require that any imported encryption technology be subject to certain governmental policies as well as key escrow by some governmental agency. Key escrow systems may be easily provided in software, but integrity and assurance remain difficult. Using only software, reliable key escrow may be impossible, in the absence of very high assurance. For example, Class B3 or A1 may be required of a "trusted computing base" in order to protect keys against disclosure or modification. Likewise, protection of algorithms against disclosure or modification, and escrow against bypass, are also often required. Under any circumstances, software offers few protections when compared with hardware solutions.

Customers, whether third-party vendors, distributors, or end users, need information security. International commercial markets need products that may be marketed internationally without a host of special revisions that must be tracked, updated, and maintained with forward and backward compatibility as to upgrades and the like. Meanwhile such solutions as key escrow do not receive ready customer acceptance in U.S. markets, particularly where a government is an escrow agent. Furthermore, individual customers or vendors may wish to operate individualized cryptography systems or other authenticable applications in conjunction with a single operating system or host application. Currently, no provision exists for allowing such customization in a controlled, authenticable environment.

Therefore, what is needed is a cryptography apparatus and method that may be mass produced in a variety of products across an entire product line. A technology, or product that can be sold in identical form both domestically and abroad is highly desirable. An encryption method and apparatus are needed that may provide improved methods for security and integrity of data provided by cryptographic algorithms and keys themselves, without requiring "trust" between sender and receiver.

Also needed is a key escrow mechanism for corporate environments. For example, file encryption by an employee will usually be required to be subject to an escrow key in the possession of the employer. Also, in conjunction with signature authorities, delegation of such authority may be useful in a corporate environment. Nevertheless, each corporate user may be viewed as a secondary (vendor) level desiring to have its own encryption and escrow control of all copies of all keys.

What is needed is a method for producing cryptographic applications that may be customized individually, from individual modules. That is, what is needed are modules that may be used to limit the capabilities of cryptographic applications without proliferating individual customized software products that may become very difficult to maintain, upgrade, support, and the like. What is needed is an apparatus and method that can separate a cryptography application or a cryptography filler for an operating system "slot" into modules. Modules need to be configured to minimize the extent of interfaces and the amount of code that must be interfaced. Modules should minimize the number of exclusions from a system that must be patched or replaced in order to enable the software system to satisfy relevant cryptography usage policies.

What is needed also are an apparatus and method effective to enable a manufacturer of a cryptographic engine to produce a single implementation of a modular package embodying particular cryptographic algorithms. The manufacturer should remain able to include that implementation in all versions of a software product. This should be true regardless of different key usage policies mandated by various regulatory authorities. It should be true regardless of a requirement for disabling of certain of the included algorithms.

Also needed is an alternative to prior art systems that require both a "policy" and an algorithm implementation to be supplied (even lastly shipped) from the manufacturer of a cryptography engine as the wrapping and certifying entity. Instead, what is needed is an apparatus having an architecture and implementation by which a manufacturer of a cryptographic engine need not be the same entity as a supplier/generator of a policy (e.g., government) to which the cryptographic engine s algorithms are constrained to conform.

Beneficial, and therefore desirable or needed, is an apparatus and method having distinct executable modules and policy data structures sufficiently separable to reduce the cost of customizing an entire software system. Thus, a system is needed that is adaptable to inexpensive customization without implementing an embedded policy.

Also needed is an apparatus and method for separating a policy from an algorithm to enable flexibility in the management and delivery of cryptographic capabilities in conformance with the local regulations. For example, some method is needed by which a manufacturer can produce a cryptographic engine, but exclude a policy certificate permitting use of the algorithms implemented by that engine. That is, a method is needed by which a manufacturer or one or more other policy certificate authorities may separately offer key and policy authorization, certification, and verification conforming to local regulations.

Further need exists for an apparatus and method whereby a plurality of dynamically linked cryptographic modules might be hosted within a single base application. Accordingly, the dynamically inked cryptographic modules might be separately accessed, authorized, and used, and might operate independently of each other. Such a need exists whereby the individual dynamically linked cryptographic modules, singly, or as a whole might be absent from the base application without significantly limiting the operation of the base application.

BRIEF SUMMARY AND OBJECTS OF THE INVENTION

In view of the foregoing, it is a primary object of the present invention to provide an apparatus and method by which a plurality of individual and distinct modular cryptography modules might be hosted by a single base application operating on a computer.

It is another object of the invention to provide such an apparatus and method by which the plurality of individual

and distinct modular cryptography modules might be separately loaded into individual slots within the host program.

It is another object of the invention to provide an apparatus and method for dynamic loading of the plurality of individual and distinct modular cryptography modules into the base executable in a manner to prevent substitution, modification, extension, or misuse of algorithms implementable by the modules and whereby each of the independent slots may have independent mechanisms for authorization of the modular cryptography modules.

It is another object of the invention to provide such an apparatus and method whereby the modules act as fillers for the slots and whereby modules filling separate slots might act independently and transparently from each other.

It is another object of the invention to provide such an apparatus and method whereby the presence or absence of a particular module within any of the slots does not prevent the proper operation of the base program.

It is another object of the invention to provide such an apparatus and method whereby the plurality of slots might be serviced by a single loader or individual loaders, and whereby the slots might be recursively constrained with inner slots located inside of outer slots and each slot being provided with an independent loader capable of authorization and linking.

Consistent with the foregoing objects, and in accordance with the invention as embodied and broadly described herein, an apparatus and method are disclosed in certain embodiments of the present invention as including a controlled modular cryptography system.

A principle feature provided by an apparatus and method in accordance with the invention includes limitation of software integration. For example, a software integration limiter may provide a cryptographic operating system with a "slot." That is, an operating system may be thought of as a block of executable instructions storable in a memory device, and loadable on a processor. An additional feature under the invention is that a plurality of such slots might exist within a given base application, such as network operating systems for example. The individual slots may be filled by different applications or components of applications, such as distinct types of cryptography systems or cryptography systems having different functions, or cryptography systems by different customers or vendors.

Furthermore, the various slots may have differing mechanisms for authenticating and linking the different applications loadable therein. For instance, each slot may use a separate public or private key for authentication purposes.

The individual slots may be serviced by a single loader, or each may have its own loader. Furthermore, the individual slots may be recursively loadable, with certain of the slots operating within others of the slots.

The software integration limiter in accordance with the invention may provide an architecture and coding to limit the integration of modular application allowed to fill these slots. Thus, the operating system or other base application cannot operate at all, or may be configured to not operate with cryptographic capability, absent an authorized, added software "filler" filling each of the slots.

Another feature available in an apparatus and method in accordance with the invention may be a vendor-constrained, remotely sealed, software integration limiter. For example, prior art systems may require that a manufacturer receive, license for import, and wrap the code of value-added resellers and vendors, incorporating the codes into a cryptographically enabled software product.

By contrast, an apparatus and method in accordance with the invention may provide for a universal "base executable" comprising a software system for operating on a computer. A software development kit may be provided with certain authorizations to an agent, vendor, distributor, or partner. The authorizations may be provided as certificates permitting the agent to create software modules and wrap them without their contents being known, even to the original manufacturer of the "base executable" or the software development kit. Such a system may then include a constrained policy obtained by the agent, vendor, etc., in cooperation with a government, to meet the import laws of the country of sale of the entire package, the software "base executable" modules from vendors, and an authorizing policy. Such a system may allow an agent (development partner, third party value-added seller, module vendor, distributor) to provide sealed encryption algorithms. The algorithms may remain known only to the agent, (partner, distributor, etc.) although accessed for linking using keys authorized by the manufacturer of the base executable.

The software development kit may provide for an authorization mechanism, such as a certificate of authority immediately identified with the software development kit and the agent. Any "base executable" may then verify any module from any vendor to determine that the vendor has produced a module in accordance with policy constraints imposed on the software development kit and verified by the "base executable" produced by the manufacturer.

Thus, a universal "base executable" may be exportable by a manufacturer and importable by a distributor or reseller. A distributor may be responsible to obtain the proper licensure for cryptographic equipment and functionality. The "base executable" can verify that all modules operating within a slot to which the particular authorization module is assigned come from a software development kit assigned to a vendor operating within prescribed bounds of policy authorization and other permitted functionality.

In short, the "base executable" knows how to recognize a valid signature provided from a module created on a proper software development kit. A software development kit may produce or generate proper digital signatures. The agent's, distributor's, or partner's module product may then carry the proper signature. Therefore, the "base executable" may recognize and run only those modules having valid signatures corresponding to software development kit "toolboxes" of known, authorized agents or distributors, and in accordance with authorized policies. The base executables may be different for each slot and each base executable may recognize only certain digital signatures.

Another feature of an apparatus and method in accordance with the invention may be a null engine. A null engine may be provided by a manufacturer with any "base executable" (principal software product, operating system), having no enabled cryptographic capability. Nevertheless, the null engine may support all interfaces required by a "slot" in the base executable, and all functionalities except cryptographic capabilities required by a "filler." Thus, for example, an operable software system may be delivered having no cryptographic capability, simply by providing a filler including a null engine to fill the "slot" within the software product (operating system, base executable) provided by the manufacturer. Thus, where only certain of the slots are available for use by a particular customer, the slots that are not available may be filled by null engines.

Another feature of the apparatus and method in accordance with the invention may be flexible key escrow capability. This feature may be thought of as a modular key

escrow method. Escrow capability may escalate from a self escrow. For example, an individual company, individual user, or the like, may hold and control all keys. At an opposite extreme an escrow of a key may reside with some other independently authorized escrow agent. A key escrow may reside with a governmental agency itself as required in some countries.

Another feature of an apparatus and method in accordance with the invention may include cryptographic wrapping of keys. That is, wrapping may be thought of as tamper proofing (authentication) and encrypting (secrecy protection) a key. Prior art systems' keys may be simply bit strings of sufficient length and complexity to reduce the probability of their being deciphered.

Here, a holder's identification and a certification authority's identification may be applied to a key itself. The digital signature of the certifying authority may enable verification of such certification. The keys may be centrally managed, such as by a management module in the "base executable" from a manufacturer. Such a module can therefore restrict creation, distribution, and use of keys, especially within a network or internetwork.

Another feature of an apparatus and method in accordance with the invention may include quality-graded certificates. The certificates may be generated by distributors (value-added resellers, module vendors, agents, partners). However, the certificates may provide a "pedigree" indicating an integrity level of the cryptography provided by a certificated software product. Thus, a purchaser of software who will become a user or owner (holder) may know the cryptographic strength (algorithm, key length) or quality (integrity, value limit of assurance) of the systems used or created, with a verification that cannot be forged.

Another feature of an apparatus and method in accordance with the invention may be provision of cryptographic engines that are not independently usable. For example, cryptographic engines may be comprised of, or included with, wrapped, non-linkable modules that can only be used in a filler to fill a "slot" in a base executable (principal software application) from a specified manufacturer.

Thus, unlike the prior art where a cryptographic engine obtained by a vendor or third party may be used with any software, cryptographic engines made in accordance with the invention may not be enabled absent verification of their integrity, applicability, policy, or the like by a base executable (principal software product). A separate slot or software instruction limiter operating within a loader within a slot may verify any and all modules attempting to link with the base executable through the slot and vice versa. Different types of modular applications and cryptography systems may be enabled through different slots.

Another feature of an apparatus and method in accordance with the invention may be constraining the linking of modules to a specific class of module, or within a specific class of module, through the use of cryptography. Thus, for example, a hierarchy of linking may be created within individual software modules, so that all modules may link only to peers (associated modules in one filler) and may not necessarily be able to link directly with selected modules of the total group of peer modules with the same filler. For example, an application or library module may not bypass a limiting manager module to interface with a cryptographic engine module.

Another feature of an apparatus and method in accordance with the invention may include a restriction of the use of cryptography, by use of cryptographic methods. Thus, for example, a manufacturer may produce a library of cryptographic functions that is much stronger than what would ordinarily be exportable under the laws of the United States, or importable under the laws of receiving countries.

For example, if only a 40-bit engine may be exported under normal circumstances, a library may be created in which a digital encryption standard using 64 bits exists within the library. Nevertheless, the more powerful standard may not be implemented, absent the proper keys which are themselves encrypted and maintained under the control of the manufacturers and the policies provided.

Thus, the above objects may be met by one or more embodiments of an apparatus and method in accordance with the invention. Likewise, one or more embodiments of an apparatus and method in accordance with the invention may provide the desirable features as described.

BRIEF DESCRIPTION OF THE DRAWINGS

The foregoing and other objects and features of the present invention will become more fully apparent from the following description and appended claims, taken in conjunction with the accompanying drawings. Understanding that these drawings depict only typical embodiments of the invention and are, therefore, not to be considered limiting of its scope, the invention will be described with additional specificity and detail through use of the accompanying drawings in which:

FIG. **1** is a schematic block diagram of modules arranged in one embodiment of an architecture for an apparatus and method in accordance with the invention;

FIG. **2** is a schematic block diagram of an apparatus in a network for hosting and implementing the embodiment of FIG. **1**;

FIG. **3** is a schematic block diagram of an example of executables operable in a processor for implementing the embodiment of the invention illustrated in FIG. **1**;

FIG. **4** is a schematic block diagram illustrating examples of data structures in memory device corresponding to the apparatus of FIGS. **1-3**;

FIG. **5** is a schematic block diagram illustrating certificate hierarchies for implementing one embodiment of an apparatus and method in accordance with the invention; and

FIG. **6** is a schematic block diagram of certain operational processes for one embodiment of a controlled modular cryptography system implemented in accordance with the invention.

FIG. **7** is a schematic block diagram illustrating a recursive arrangement of slots in accordance with the invention.

FIG. **8** is a schematic block diagram illustrating alternative arrangements for including a plurality of slots in a base application.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

It will be readily understood that the components of the present invention, as generally described and illustrated in the Figures herein, could be arranged and designed in a wide variety of different configurations. Thus, the following more detailed description of the embodiments of the system and method of the present invention, as represented in FIGS. **1** through **6**, is not intended to limit the scope of the invention, as claimed, but it is merely representative of certain presently preferred embodiments of the invention.

The presently preferred embodiments of the invention will be best understood by reference to the drawings, wherein like parts are designated by like numerals through-

out. Reference numerals having trailing letters may be used to represent specific individual items (e.g., instantiations) of a generic item associated with the reference numeral. Thus, a number **156***a*, for example, may be the same generic item as number **156***f*, but may result from a different version, instantiation, or the like. Any or all such items may be referred to by the reference numeral **156**.

Referring to FIGS. **1-3**, a method and apparatus for providing controlled modular cryptography is illustrated in software modules supporting hardware, and as executables distributed through a processor. Reference is next made to FIGS. **4-6**, which illustrate in more detail, schematic block diagrams of certain preferred embodiments of an apparatus and method for implementing controlled modular cryptography in accordance with the invention.

Those of ordinary skill in the art will, of course, appreciate that various modifications to the detailed schematic diagram of FIGS. **1-6** may easily be made without departing from the essential characteristics of the invention as described. Thus, the following description of the detailed schematic diagrams of FIGS. **1-6** is intended only as an example, and it simply illustrates certain presently preferred embodiments of an apparatus and method consistent with the invention as claimed herein.

Referring now to FIGS. **1-6**, a controlled modular cryptography (CMC) process **12** or method **12** may be implemented using a plurality of modules **13**. The CMC process **12**, alternately referred to herein as CMC **12**, may be embedded within another executable **14**, such as a network operating system **14**. The network operating system **14** may include an operating system proper **15**, what would conventionally be known in a generic operating system as the operating system **15**.

The operating system **14** may also have provision for insertion of a pre-processor **92** in a conventional hole **93**.

By contrast, the CMC **12** is not accessible by third parties at a pre-processor slot **93**. Third parties may create pre-processors **92** having direct access to the operating system **15**. Prior art cryptographic engines are often mere pre-processors interposed between applications **40** and the operating system **15**. Likewise, the unauthorized installation by a third party of a cryptographic engine in a pre-processor slot **93** may be rendered virtually impossible by the CMC **12** and operating system **15**. Instead, the CMC **12** may be loaded into the base executable **14** such as the operating system **14** in a manner that embeds the CMC **12** into the operating system **14** and prevents interfacing by any third party to the cryptographic capability of the CMC **12**. (See FIG. **3**.)

Referring now to FIGS. **1-6**, and more particularly to FIG. **1**, the controlled modular cryptography **12** may include library modules **16** for interfacing with applications **40**. Each library module **16** (X library **16**, or simply library **16**) may be application-specific.

The loader **90** (see FIG. **3**) provides layering (hierarchical linking) or nesting (recursive loading or linking) of interfaces. The layering may effectively prevent applications **40** and an operating system proper **15**, for example, from interfacing directly with controlling modules **13** (e.g., manager modules **18**) or with engines **20** (e.g., cryptographic engine **50**). The loader **90** may do this directly by dynamic loading of modules **13**, enforcing restrictions on access to interfaces between levels **16**, **18**, **20**, **22**, **15** illustrated in FIG. **1**.

Manager modules **18** as well as the original loader **90** loading a filler **12** (CMC **12**) may assure that a layering hierarchy is enforced between modules to limit interfaces. Manager modules **18** may interface with library modules **16**. The manager modules **18** may also interface with the cryptographic engines **20**, or engine modules **20**. Support modules **22** may interface with engine modules **20** also.

Library modules **16**, manager modules **18**, and support modules **22** may interface with the operating system **15** in one preferred embodiment of an apparatus and method in accordance with the invention. The engines **20** may be enabled to interface only with other modules **13** , and not with the operating system **15**.

The subdivision of modules **13** in a layering architecture (e.g., layer **18** is manager modules **18** including modules **42**, **44**, **46**) great flexibility may be obtained. Since modules **13** are dynamically bound by the loader **90**, and managed by a manager module **18**, the modules **13** may be modified or exchanged as needed and as authorized. Management modules **18** not yet envisioned may be added in the future, without revising the base executable **14**, or even the filler **12**, outside the module **13** in question.

For example, a management module **18** may support cryptographic-token PIN (personal identification number) management, not available in an initially purchased product **14**. Another example may be added support for policy management enhancements, such as providing separate APIs (application programming interfaces) for encrypting and decrypting ubiquitous financial data used by banks.

New functionality, not typically used or required in current practice by banks, may include a separate key type or size specifically for financial data. Such keys **156**, **160** may be relatively stronger than general keys **156**, **160**, while use, holders, data types, and the like may be restricted by policies **164** crafted for special financial purposes. Thus financial keys **156**, **160** may be managed differently from other general types of keys **156**, **160**.

Referring now to FIG. **2**, a network **56** may comprise a plurality of nodes **58** (e.g., clients **58**). Although the clients **58***a*, **58***b*, **58***c* are illustrated, the computers of the clients **58**, server **62**, and router **64** may also be thought of as being hosted, programmed to run on any computer **58**, **60**, **62**, **64** on a network **56**. Likewise, the CMC **12** may be programmed into any of those computers **58**, **60**, **62**, **64**. By node is meant a computer on a network **56** in its broadest sense.

Likewise, the host **60** or server **60** may actually be programmed to function as one of several servers **62** or routers **64**. As a practical matter, the server **62** may be replaced by the server **60**. A node **58** may include some or all of the structural contents illustrated for the server **60**. For example, if every node **58** comprises a computer, every node **58** may have any or all of the components **70-86**.

A network **56** may include a backbone **66** for interconnecting all the nodes **58** or clients **58**. The router **64** may also connect to one or more other networks **68**. The network **68** may be a local area network (LAN), wide area network (WAN) or any size of internetwork.

The server **60** may include a CPU **70** or processor **70** for hosting the operating system **14** and CMC **12**. As a practical matter, a random access memory **72**, or RAM **72**, may temporarily store, in part or in total, any or all codes and data associated with executables within the CMC **12**. For example, during operation of the CMC **12**, individual modules **13** might be stored, or a portion thereof might be stored in the RAM **72**.

The CPU **70** and RAM **72** may be connected by a bus **74**. Also on the bus may be operably connected a network card **76** or network interface circuit **76** (net card **76**), one or more input devices **78**, and output devices **80**, or the like. Additional memory devices such as a read-only memory **82**

(ROM **82**) and a storage device **84** (such as a hard disk **84**), may be operably connected to the bus **74** to communicate data with the processor **70**.

Additional ports **86** may be provided as appropriate. As a practical matter, the input device **78** and output device **80** may merely represent ports for accessing one or more available input or output devices **78**, **80**. Similarly, with the distributed nature of hardware and software in a modern computing environment, other devices may be accessed, through the net card **76**, elsewhere on the network **56**.

Referring to FIG. **1** once more, the interfacing between modules **13** may be restricted. Such a restriction may provide additional assurance that the CMC **12** may not be misused, modified, or replaced improperly. Therefore, certain of the modules **13** may have operating system interfaces **24**. For example, the interfaces **24***a*, **24***b*, **24***c* represent the interfaces between the libraries **16**, managers **18**, base **22**, respectively, shared with the operating system **15**.

In the illustrated embodiment of FIG. **1**, the engines **20** share no interface with the operating system **15**. Instead, the engines **20** may interface through the base support **22**. Library interface **26** represents the interface between library **16** and applications **40**. The library interface **26** may be considered to be an interface between the CMC **12** and applications **40**.

The libraries **16** may be structured to interface directly with applications **40**. The foundation **54** or the CMC foundation **54** may be thought of as the core of the CMC **12**. The managers **18** provide cryptographic facility as well as controlling access to and between modules **13**, especially in the core **12**. The interface between the CMC enforcement by the foundation **54** and applications outside the base executable **14** is moved away from the manager interface **28** by the library interface **26** and interposed libraries **16**. Thus applications **40** are not permitted to interface directly with the (controlling) management modules **18**. This further avoids creation of cryptography with a hole.

The manager interface **28** represents the interface between the manager modules and the library modules **16**. The engine interface **30** represents the interface between engines **20** and the manager modules **18**. The support interface **32** represents the interface between the engines **20** and the support modules **22**.

In general, communications **38** may be calls from upper layers **40**, **16**, **18**, **20** shown to lower layers **16**, **18**, **20**, **22**, respectively, in FIG. **1**. Each layer **16**, **18**, **20**, **22** may properly execute without requiring anything from a layer **18**, **20**, **22**, **15**, respectively, below.

For example, in one embodiment of an apparatus and method in accordance with the invention, one library **16** may be an audit library **34**. For example, the audit library **34** may have functional responsibility for encrypting security audit data for storage. The underlying data may correspond to events of significance to audit executables. The network **56** itself may be managed by an individual acting as a system manager, yet the audit data encrypted by the audit library **34** may be inaccessible to the system manager.

Other libraries **36** may be provided. Each of the libraries **36** may be application specific. In one presently preferred embodiment, each of the applications **40** interfacing at the library interface **26** may have an associated, unique, library module **36** provided.

The key generation manager **42** may create symmetric keys or asymmetric key pairs provided to cryptographic engines **20**. The key generation manager **42** may also perform the escrow function of the escrow archive **170** (see FIG. **6**). A base manager **44** may provide enforcement of policies **164**.

Access to modules **13**, such as the engines **20**, and access to cryptographic algorithms within engine modules **20**, and the like, may be enforced by the manager modules **18**. In one embodiment of an apparatus and method in accordance with the invention, the base manager **44** may provide an enforcement function with respect to all functions and all modules **13**. Other managers **46** may also be provided. For example, manager modules **46** may alter methods of policy enforcement for the escrow of keys **156**.

In one embodiment, the CMC **12** may be provided with a null engine **48**. A null engine **48** may be operated to interface at the engine interface **30** and the support interface **32** while providing no enablement of cryptographic capability. Thus a base executable **14** may be fully functional otherwise, including all necessary interfaces to the filler **12** (CMC **12**), while having no enabled cryptographic capability. The interfaces **26**, **24** to the filler **12** may be as secure as if the dynamically loaded modules were manufactured as integrated portions of the base executable **14**.

Thus, an apparatus **10** may be provided as a base executable **14**, having fully imbedded support for a cryptographic engine **20**. However, the presence of a null engine **48** accommodates all the proper interfaces **30**, **32** while actually providing no cryptographic capability.

Thus, a CMC **12** (filler **12**) may be provided with a base executable **14**, including a null engine **48**, exhibiting minimal differences with respect to the operating system **15** as compared to another cryptographically-enabled product. Meanwhile, other engines **50** may be provided by a manufacturer or a third party vendor authorized to create cryptographic engines **20** according to some policy and authorization.

A base support module **52** may provide some minimal set of operating system instructions for the engines **20**. That is, in general, the engines **20** need some access to the operating system. Nevertheless, for providing the assurance that engines **20** may not be created, modified, extended, circumvented, inserted, or the like, in an unauthorized fashion, the support module **52** may intervene. Thus, the base module **52** may provide access to some limited number of functions from the operating system **15** for the engine **20**.

Referring now to FIG. **3**, an operating system **14** may be implemented in one embodiment of an apparatus and method in accordance with the invention to include a loader **90**. The loader **90** may be associated with the operating system proper **15**. The functional responsibility of the loader **90** may be to dynamically load and properly link all modules into the CMC **12** (filler **12**), for example, installing (e.g., embedding) them into the operating system **14**.

More specifically, the loader **90** may be tasked with the functional responsibility to provide all proper linking between modules **13**. Linking may be enabled on a layer-to-layer (or interface **28**, **30**, **32**) basis rather than on a moduleby-module basis. For example, a binding may exist between any two modules **13** in a layer (e.g., layer **18**, or layer of manager modules **18**). Binding may also exist between any module (e.g., modules **42**, **44**, **46**) in that layer (e.g., layer of managers **18**) and another module (e.g., modules **48**, **50**) in a layer (e.g., layer **20**, or layer of engines **20**) sharing an interface (e.g., interface **30**) with that layer (e.g., layer **18**).

Specific modules **13** need not be individually limited and controlled by the loader **90**. In one embodiment, individual modules **13** may be bound (linked). Thus, for example, only

those functional features authorized for a key generation manager **42**, or a cryptographic engine **50**, might be enabled by being properly bound.

In one example, a cryptographic engine **50** may be manufactured to contain numerous algorithms. However, according to some policy **164** (see e.g., FIGS. **5**, **6**) incorporated into a certificate **154**, a manager **46** and the loader **90** may limit linking (binding) to an enablement of algorithms and engines **20**. A manager module **46** may also control key usage, including length and function. Function may be distinguished between, for example, encryption versus authentication. Use may be controlled based upon, for example, a manufacturer's (of the module **13**) signature **162** and key type.

The operating system **15** may support a selection of pre-processors **92** such as the audit event filter **92**. Preprocessors may be adaptable to fit in a hole **93** readily available for the purpose. In one currently preferred embodiment of an apparatus and method in accordance with the invention, a CMC **12** is not adaptable to be implemented as a preprocessor **92**. Instead, the CMC **12** may be limited to interfacing only with the operating system proper **15** as illustrated in FIG. **1**, and only after proper loading by a loader **90**. Even within the operating system **15**, the CMC **12** may be limited to interfacing with the operating system **15** through a limited number of interfaces **24**.

As a practical matter, certain applications **94** or programs **94** have resident portions within the server **60** hosting the operating system **14**. For example, a file system **98**, a name service **100**, a work station **102** and the like may have resident portions operating in the processor **70**. Even if, for example, a server **62** is operating as a file server, the file system **98** may be a portion of a file server executable that needs to be resident within the processor **70** of the server **60** in order for the server **60** to communicate with the server **62** over the network **66**.

Generally, certain data may need to flow into and out of the operating system **14**. Accordingly, a number of channels **96** or data flow paths **96** may need to exist. As a practical matter, the channels **96** may be comprised of either paths, data itself, or as executables hosted on the processor **70** for the purpose of providing communication. Thus, an audit file **104**, an accounting log **106**, an archive file **108**, and the like may be provided as channels **96** for communication.

Thus, the overall operating system **14** along with the applications **94** and channels **96** may be thought of as a local system **110** or the local processes **110**. These local processes **110** operate within the CPU **70**. The CPU **70** is a processor within the server **60** or host **60**. As a practical matter, the processor **70** may be more than a single processor **70**. The processor **70** may also be a single processor operating multiple threads under some multitasking operating system **15**.

Data representing executables or information may be stored in a memory device **72**, **82**, **84**. Referring now to FIG. **4**, one may think of a dynamic data structure **114** or an operating system data structure **114** storable in an operable memory **116**. That is, for example, the operating memory **116** may be within the RAM **72** of the host **60**. All or part of the data structure **114** may be moved in and out of the processor **70** for support of execution of executables.

The data structure **114** may be dynamic. The modules **13** for example, may be dynamically loadable, such as network loadable modules. Thus, for example, a host **60** may operate without having any fixed, storable, data structure **114**. That is, no static data structure need be assembled and stored in a manner that may make it vulnerable to being copied or otherwise inappropriately accessed. The data structure **114** may only exist dynamically during operation of the processor **70**, and even then need not all exist in the memory device **116** (e.g., RAM **72**) simultaneously at any time. Thus, additional assurance is provided against misuse, and abuse of data and executables in a CMC **12** associated with an operating system **14**.

The data structure **114** may contain a certificate **118** and certificate **120**. A certificate **118**, for the purposes of FIG. **4**, may be thought of as an instantiation of a certificate **154** associated with the operating system **14** and its included CMC **12**. The certificate **118** may be thought of as the data certifying the holder of a certificate operating and using the data structure **116**. By certificate **120** is meant data provided in a certificate issued to the holder.

A certificate **118**, **120** may also be thought of as a binding of a holder ID **122**, **132** to a public key **126**, **136**, certified by a digital signature **124**, **134** of a certifying authority. An issuer (e.g., **152***b*) or authority and a holder (e.g., **152***d*) may each be a holder (e.g., **152***b*) to a higher authority (e.g., **152***a*), and issuer (e.g., **152***d*) to a lower holder (e.g., **152***h*), respectively.

When discussing authorities, holders, receivers, and the like, it is important to realize that such an authority, holder, sender, receiver, or the like may actually be a hardware device or a software operation being executed by a hardware device. Any hardware device, operating software, or data structure in a memory device may be owned, controlled, operated, or otherwise associated with an individual or an entity. Nevertheless, insofar as the invention is concerned, names of such entities may be used to represent the hardware, software, data structures, and the like controlled or otherwise associated with such entities.

As a practical matter, a certificate **118** authenticating the rights of the CMC **12** may contain an identification record **122** identifying the holder (the specific instance of the CMC **12**), a signature record **124** verifying the higher certification authority upon which the holder depends, and a public key record **126** representing the public key of the holder. The private key **128** may be very carefully controlled within the CMC foundation **54** using encryption for wrapping. The private key **128** may be associated with the holder (CMC **12**) and is the private half **128** of a key pair including the public key **126**. Thus, by means of the private key **128**, the holder may create the signature **134** in the certificate **120** for another use of the key pair **136**, **138**.

Meanwhile, a certification authority **152** (see FIGS. **5-6**) may provide to a holder or sign **166**, the certificate **118** (one of the certificates **154**). The certificate **120** may reside in another computer or simply be allocated to a different thread or process than that of the certificate **118**.

As a practical matter, a private key **128**, **138** may be protected by physical security. Therefore, a private key **128**, **138** may typically be controlled and be cryptographically wrapped except when dynamically loaded into a dynamic data structure **114**.

The private key **128** may be used to certify an identification record **132** identifying a new holder. A signature **134** created by use of the private key **128** may verify the authenticity and quality of the certificate **120** and public key **136**. The public key **136** may be thought as the matching key **136** to a key pair including the private key **138** created by the new holder of the certificate **120**. That is, one may think of a new holder, as a process, or an individual issuing a public key **136** certified by the signature **134** of the private key **128** as duly authorized to create software which functions within the limits of a policy **140**. The certificate **118**, an instance of

a certificate **154** held by the CMC **12**, may have a signature **124** by a higher certifying authority **152**.

A policy **130**, **140** may limit the authorization of the holder identified by the ID **122**, **132** and certified by the digital signature **124**, **134**. A policy **130**, **140** may incorporate the limitations governing the use of algorithms in engines **20**, for example. Thus, a policy **130**, **140** may be thought of, for example, as the rules enforced by a manager module **18** controlling access to and from a module **13**, such as an engine (e.g., cryptographic engine **50**).

Each policy **164** (e.g., **164***d*, see FIG. **5**) may contain a digital signature **163** (e.g., **163***d*) of the certifying authority **152** (e.g., **152***b*) above the holder **152** (e.g., **152***d*) of the certificate **154** (e.g., **154***d*) and policy **164** (e.g., **164***d*). The policy **164** (e.g., **164***d*) may thus be bound to the corresponding certificate **154** (e.g., **154***d*) by the digital signature **163***d*.

In one embodiment, policies **164** may be generated by a separate policy authority using a policy authority digital signature **129**, **139** (see FIG. **4**). A policy authority signature **129**, **139** binding a policy **130**, **140** to a certificate **118**, **120** need not be different from a certificate authority signature **124**, **134**, but may be. This is analogous to the certification authorities **152** for certificates **154**. Thus, the policies **164** may be provided and signed **166** by a certifying signature **163** binding the policy **164** to a corresponding certificate **154**. Nevertheless, the policy **164** may be certified by a policy authority **129**, **139** other than the certificate authority **152** creating the corresponding certificate **154**.

Referring to FIGS. **4-6**, the certificate **118** may include identification records **122**. The identification records **122** may contain information recursively identifying the higher certifying authority (e.g., **152***a*, **152***b*), as well as the holder (CMC **12**) certified. However, the signature **124** may be verified by using the public key of the higher authority **152**. For example, the signature records **124** may comprise a signature of a signature root authority **152***b* or higher authority **152***a* certifying, which authority is known by the identification **133**. The private key **128** may be thought of as a key by which the holder (e.g., the CMC **12**) creates signatures **134** for certificates **120** associated with, for example the key generation module **42** of the base executable **14** (see FIG. **6**).

The identification records **132** may typically identify the holder of the certificate **154** associated with the certificate **120**. Although the signature **134** is associated with the certifying authority providing the certificate **120**, and itself holding the certificate **118** identification records **133** may identify the certifying authority **152** (e.g., associated with the ID **122**). The signature **134** may be used by entities or processes needing to verify the authorization of the holder (entity identified by the ID **132**) of the certificate **120**.

As a practical matter, a private key **128**, **138** is typically not stored in the clear in any non-volatile storage generally available. That is, a private key **128**, **138** may typically be unwrapped or loaded only dynamically to minimize the chance of any unauthorized access. The private key **128**, **138** may optionally be stored within a cryptographic co-processor, for example an additional processor **70**. The cryptographic co-processor may be embodied as a chip, a token, a PCMCIA card, a smart card, or the like. The private key **128** may be unwrapped in the co-processor for use only within the co-processor.

The applications **40**, the preprocessor **93**, and the channels **96** may be stored in the data structure **114**. Nevertheless, the data structure **114** may be distributed.

The library modules **16**, the manager modules **18**, the engines **20**, and the support modules **22** may be stored in the data structure **114**. In one embodiment, the data structure **114** may all be resident in the RAM **72** in some dynamic fashion during operation of the operating system **14** functioning in the processor **70**.

The certificate **120** may be embodied as illustrated in the frames **142**. The identification record **132** may be thought of as a data segment **132** associated with a holder. The segment **133** may be provided to identify a certifying authority **152**. Each public key **136**, **126** may be represented as bits of a segment **136**, **126** in the frame **142**. The signature **134**, **124** of a certifying authority **152** may be represented as another set of bits of a segment **134**, **124** in the frame **142**. The policy **140**, **130** may be represented by another segment **140**, **130**. The certificates **118**, **120** may have corresponding (e.g., even identical) policies **130**, **140** under which to operate.

The public key **136**, **126** is identified with the holder ID **132**, **122**. A public key **136** **126** is typically published to other functions or to other entities, just as a certification authority s **152***a*, **152***b*, **152***c* public key **160***a*, **160***b* **160***c* is published. Thus, a certifying authority's public key **136**, **126** is illustrated in FIG. **4** as being separate from the frame **142**. The public key **136**, **126** may be embedded in another certificate held by a certifying authority. Similarly, a holder's private key **138**, **128** may be maintained with utmost security. Therefore a holder's private key **138**, **128** is not available with the holder's published public key **136**, **126**, except to the holder. Thus, a holder's private key **138**, **128** may not actually be generally available or associated with the certificate **120**, or certificate **118**, respectively, in the frame **142**.

Referring now to FIGS. **4-6**, the certificate hierarchy is illustrated, as is the implementation of operational keys **156**, **160**. Reference numerals having trailing letters, may be thought of as specific examples of a generic structure or function associated with the reference numeral alone. Thus, a certifier or certification authority **152** is a general expression, whereas the root certifier **152***a* and the CMC signature root **152***b* are specific examples of a certification authority **152**.

In general, an authority **152** (e.g., root certifier **152***a*), may issue a certificate **154** (e.g., **154***b*, **154***c*). A certificate **154** (e.g., **154***b*, **154***c*) may be associated with authorization of a certificate holder (e.g., **152***b*, **152***c*) by a certification authority **152** (or just authority **152**). Associated with a certificate **154** may be certain data **120**, **118**. For example, in one embodiment, a certificate **154** may actually be embodied as a frame **142** as illustrated in FIG. **4**.

In general, a certificate **154** (e.g., **154***b*) may be prepared by an authority **152** (e.g., **152***a*) using a private key **156** (e.g., **156***a*) held securely in the possession of the authority **152** (e.g., **152***a*). A certificate **154** (e.g., **154***b*), itself, may contain information such as the holder identification **158** identifying the holder to whom the authority **152** has issued the certificate **154**. Note that the holder **152** (e.g., **152***b*) may itself be another authority **152** (e.g., **152***b*) to a lower level holder **152** (e.g., **152***d*).

The certificate **154** may also include the authority's **152** signature **162**. By signature **162** is meant, a digital signature as known in the cryptographic art. Also included in the certificate **154**, or linked by the signature **162** with the corresponding certificate **154**, may be a policy **164**. A policy **164** represents the extent of the authorization provided by the certificate **154** (e.g., **154***a*) to the holder (e.g., **154***b*) of the certificate from the authority (e.g., **152***a*) in order to produce cryptographic functionality.

For example, a holder **152***d* may have a certificate **154***d* and private key **156***d* authorizing the holder **152***d* to produce modules, such as cryptographic engines **20**, manager modules **18**, library modules **16**, or symmetric or asymmetric keys **156**. The policy **164***d* may embody the restrictions, limitations, and authorizations extended to the holder **152***d* of the certificate **154***d*.

In one embodiment, the enforcement of policies **164** may be managed in one or more of several, relatively sophisticated ways. For example, a policy **164** might permit a private key of a relatively long length, such as 1024 bits, to be used for digital signatures **162** only. On the other hand, a private key **156** used to wrap symmetric keys may be permitted to extend only to 768 bits, and only on condition that the key **156** be escrowed.

Also, rules for "composition" of policies **164** (certificated features or functions), or perhaps more descriptively, "superposition" of policies **164**, may be embodied in manager modules **18**. For example, more than a single policy may be loaded within a filler **12**, for one of several reasons. For example, modules **13** from different vendors may be manufactured under different authorities **152**. Also by way of example, as in FIG. **4**, a policy authority digital signature **129**, **139**, certifying a respective policy **130**, **140**, need not be from the same source as a certificate authority digital signature **124**, **134**, but may be.

Meanwhile, a manager module **18** may be programmed to enforce the most restrictive intersection of all features (e.g., certificated features or functions such as quality, cryptographic strength, etc.). For example, one policy **164** (a certificated feature) may require that key-wrapping keys may be 1024 bits long and must be escrowed. Another policy **164** in another module **13** in the same filler **12** may require that keys be only 512 bits long, but need not be escrowed. The cryptographic manager module **18** may require a key length limit of 512 bits, and require escrow also. Thus a superposition of policies **164** may use the most restrictive intersection of policy limitations.

An authority **152**, thus certifies **166** or provides a signing operation **166** for a certificate **154** for a holder. Referring to FIG. **5**, the certification authority **152***a* (the root certifier **152***a*) is an authority **152**, to the CMC signature root **152***b* as a holder, both with respect to the certificate **154***b*.

Each certificate **154** is signed using a private key **156** of a certifying authority **152**. For example, the certifiers **152***a*, **152***b*, **152***e* use private keys **156***a*, **156***b*, **156***e*, respectively, to sign the certificates **154***b* and **154***e* delivered to the CMC signature root **152***b* and server CA **152***e*, and certificate **154***j* forwarded by the key generation module **42**.

The certificate **154***b* also includes a public key **160***b*. A public key **160**, in general, is one half of a key pair including a private key **156**. For example, the private **156***a*, **156***b*, **156***c*, **156***d*, **156***e*, **156***f*, **156***g*, **156***h* is the matched half associated with the public **160***a*, **160***b*, **160***c*, **160***d*, **160***e*, **160***f*, **160***g*, **160***h*. The key pair **156***a*, **160***a*, is associated with the root certifier **152***a*. Similarly, the private key **156***b* may be used by the CMC signature root **152***b* to certify **166***d*, **166***e* the certificates **154***d*, **154***e* with the signatures **162***d*, **162***e*. Thus, in turn, each of the public keys **160***d*, **160***e*, respectively, is the public key half of the pair that includes the private key **156***d*, **156***e*, respectively.

A holder, such as the module signature authority **152***d* or the server certification authority **152***e* may verify the validity of the public key **160***b* using the signature **162***b* and the public key **160***a*. Similarly, a processor entity may verify the validity of the certificates **154***d*, **154***e*, respectively, by recourse to the signature **162***d*, **162***e*, respectively and the publicly available public key **160***b* responsible.

Referring to FIGS. **5** and **6**, generation of private/public key pairs **156**, **160** and subsequent certification **166** may be represented by cascading certificates **154**. For example at the top or root of all certification authorities **152** may be a root certifier **152***a*. The certifier **152***a* may generate a private **156***a*, and a public key **160***a*, as a key pair **156**, **160**.

The root certifier **152***a* need have no signature **162**. The root certifier **152***a* in such circumstance must be "trusted." Another method, other than a digital signature **162** of a higher certifying authority **152**, may typically be required for verifying the public key **160***a* of the root certifier **152***a*.

Only one root certifier **152***a* (RC **152***a*) is needed for the entire world. In one embodiment, the root certifier **152***a* may be an entity willing and able to credibly assume liability for the integrity of public keys **160**, and the integrity of associated certificates **154**. For example, an insurer, or a company known and trusted by the entire business world, may serve as a root certifier **152***a*. Such companies may include large, multinational insurance companies and banks. The root certifier **152***a* is functionally responsible to physically protect the secret key **156***a*. The root certifier **152***a* is also responsible to distribute the public key **160***a*.

The root certifier **152***a* may authorize private/public key pairs **156***b*, **160***b* to be created by the CMC signature root **152***b*. The integrity of the public key **160***b*, and the identity **158***b* of the CMC signature root may be certified by a digital signature **162***b* created by the root certifier **152***a* using the private key **156***a*.

Any subsequent entity, receiving a certificate **154** cascading from the CMC signature root **152***b* as a certifying authority **152**, may verify the certificate **154**. For example, the certificate **154***b*,and its contents (public key **160***b*, ID **158***b*, and signature **162***b*) may be verified using the signature **162***b*. The signature **162***b* may be created using the private key **156***a*. Therefore, the signature **162***b* can be verified using the public key **160***a*, available to the entity to whom the authority of a certificate **154***b* is asserted as authentication.

The root certifier **152***a* may have its public key **160***a* embedded in the base executable **14**. Alternatively, any method making the public key **160***a* securely available may be used. In this example, the base executable **14** or principal software product **14** may typically, be an operating system **14**. The base executable **14**, operating system **14** or base executable **14** may be thought of as including everything that arrives in the base executable associated with newly purchased, generic, software package **14**. This may sometimes be referred to as "the base executable **14**."

As a practical approach, the CMC signature root **152***b* may be associated with, and the private key **156***b* be in the possession of, the "manufacturer." For example, the manufacturer of a base executable **14**, such as a network operating system **14**, may be the holder of the private key **156***b* used to certify all public keys **160***d* and associated certificates **154***d* of the module signature authority **152**.

As a practical matter, the highest level of public key **160** embedded in (or otherwise securely available to) a base executable **14** may be the signature root key **160***b* associated with the certificate **154***b*. An instantiation of the certificate **154***b* may be embedded in, or otherwise securely available to, the CMC loader **90**. Thus, the loader **90** may verify against the manufacturer s public key **160***b* (available to the loader) the signature **162***d* in the certificate **154***d* effectively presented by the module **13**. That is, one may think of the

certificate **154***d* as being included in the cryptographic module **13** (engine **20**) of FIG. **6** by a module vendor.

Thus, the loader **90** may verify that a vendor is authorized to produce the modules under the policy **164***d* bound to the certificate **154***d*. However, the foregoing starts at the wrong end of the process. The signature **168** on the module **13** is present for verification of the module by the loader **90**. The signature **168**, encrypted using the private key **156***h*, may be verified by recourse to (e.g., decryption using) the public key **160***h*. The key **160***h* presented in the certificate **154***h*, also available with the module **13**.

In turn, the signature **162***h* on the certificate **154***h*, may be verified using the public key **160***d*. The key **160***d* corresponds to the private key **156***d* used to encrypt the signature **162***h*. The key **160***d* is available in the certificate **154***d* with the module **13**. The certificate **154***d* and key **160***d* are verified by the signature **162***d* on the certificate **154***d* with the module. The signature **162***d* may be verified (e.g., such as by decryption or other means) using the public key **160***b* of the CMC signature root **152***b*. An instantiation of this key **160***b* is available to the loader **90** with the certificate **154***d*, as discussed above. By having the certificate **154***d* independently of the modules **13**, the loader may thus verify each module before loading into the filler **12** (CMC **12**).

As an example, the CMC signature root **152***b* may be associated with the manufacturer of the base executable **14**. The base executable **14** may be thought of as the principal software product **14**, such as an operating system **14**. By contrast, the CMC **12** may be thought of as a filler **12**, a modularized segment that is required to be present within the base executable **14**, but which may be modified, customized, limited, authorized, or the like by a manufacturer for a class of customers or by a suitably authorized, third-party vendor of modules.

In the case of a base executable **14** that serves as a network operating system **14**, such as Novell Netware™, the manufacturer (Novell, in this example) may be the CMC signature root **152***b*. Another example may be a third-party vendor of modules **13**. A third party vendor of modules **13** may produce, for example, engine modules **20** for insertion into the CMC **12**, but may be a value-added reseller of the base executable **14** adapted with such a cryptographic engine module **20** or other module **13**.

For purposes of discussion, a manufacturer may be thought of as the maker of the base executable **14**. A vendor or third party vendor may be thought of as the maker of modules **13** for inclusion in the CMC **12** (filler **12**) portion of the base executable **14**. A distributor, reseller, or third party reseller may be thought of as a seller of base executables **14** purchased from a manufacturer. The manufacturer may distribute and create modules **13**. A vendor of modules **13** may be a distributor of the base executable **14**, also.

Thus, a situation of great interest involves a manufacturer desiring to provide the base executable **14**, while certifying a vendor s module products **13**. The modules **13** may be integrated as part of the CMC **12** of the base executable **14** after the base executable **14** is shipped from the manufacturer. As discussed above, shipment of a base executable **14** in some standard configuration is desirable. In a preferred embodiment a base executable **14** shipped into a foreign country having import restrictions on cryptography, may provide a reliable method for enabling authorized cryptography exactly, while disabling all other potential uses of cryptography. Minimum modification, interfacing, and cost may be provided by an apparatus and method in accordance with the invention, with maximum assurance of authorization and control, all at a reasonable processing speed.

The CMC signature root **152***b* may be responsible for manufacturing and exporting the base executable **14** to customers (users) and third party resellers, and supporting software development kits (SDKs) to third party vendors. The manufacturer may be a maker modules **13** also. Typically, the manufacturer may produce the null engine **48**, at least.

The module signature authority **152***d* associated with the ID **158***d* may be that of the holder of a software development kit for modules **13**. A policy **164***d* bound to the certificate **154***d* may be certified by the signature **162***d* of the CMC signature root's **152***b* private key **156***b*.

The policy **164***d* may be enforced by the manager module **42** and embodies the limits on the use and strength of keys **156***d*. For example, the length (strength) of keys **156** useable under the policy **164***d* and the types of modules **13** may be controlled by statute in each country of importation for the base executable **14**.

A loader **90** from the manufacturer may control linking of modules **13**. Thus, a third party, including a module vendor cannot change the limitations inherent in a key, the policy, or the like.

A policy **164**, in general, may define the maximum strength of the key. A module signature authority **152***d*, holding a particular authorized software development kit may create different types of keys **156** as long as each is within the bounds of the policy **164***d*. The module signature authority **152***d* may also then certify a module-signing key pair **152***h* authority for each module type produced and sold. Certificate **154***h*, so signed using the private key **156***d*, may provide a key **156***h* to sign each module **13**, such as the cryptographic modules **13** exemplified by the engine **20** of FIG. **6**. Meanwhile, a module signature authority **152***d* may certify embedded keys **160***h* and associated certificates **154***h* automatically by using the software development kit.

Note that a chain or cascade of certificates **154***d*, **154***h* may be used in a module in order to have the signatures **162** for the loader **90** to verify. The loader **90** may then verify the keys **160***d*, **160***h* using signatures **162***d*, **162***h* of the certificates **154***d*, **154***h* to authorize the loading of the module **20** (see FIG. **6**).

Verification may be necessary in order for the loader to have the certified keys **160***d*, **160***h*, **160***b* necessary for verifying the module signature **168**. That is, a vendor may use a software development kit containing a module signature authority **152***d* to create some number of module signing key pairs **152***h*.

The private keys **156***h* may be used to sign **166***m* with a signature **168** every module **13** created. Note that the modules **16**, **10**, **42** in the base executable **14** of FIG. **6** may all be thought of generically as modules **13** as in FIG. **1**. The certificate hierarchy **154***h*, **154***d*, **154***b* of the module **13** may all be verified by the loader **90** using the appropriate public keys **160***d*, **160***b*, to verify the respective signatures **162***h*, **162***d* from the certificates **154***h*, **154***d*.

The server certifying authority **152***e* (CA **152***e*) may be produced by the manufacturer based on a CMC signature root **152**. The server certificate authority **152***e* may be embodied in the server **60** (see FIGS. **2** **6**) on a server-by-server basis. Thus, a server **60** may generate keys **156***j* or pairs as shown in FIG. **6**. Thus, the server **60** is able to certify by a key generation manager **42** keys **160** generated by that server **60**.

A private key **156** may preferably be unique to an individual server **60** so that there is no need to provide a globally

exposed private key **156**. The private key **156***e* of the server certificate authority **152***e* of FIG. **6** may be the only private key **156** embedded in a base executable **14** or operating system **14** hosted by a server **60**. This may be very important for providing signatures **162***j* for certifying **166***j* other keys **160***j* and IDs **158***j* signatures **162**.

As a practical matter, by embedding is meant alternate methods that may be implemented in the server **60** in another manner well adapted to dynamic loading. For example, the private key **156***e* may not necessarily need to be embedded, as in the illustrated example. Rather, the key **156***e* may simply be "securely available" such as by reading from a secure hardware device. Thus, a key **156***e* may be securely available to the CMC **12** in the server **60** and function as well as if actually embedded. The expression embedded should be interpreted broadly enough to include this "securely available" approach. This is particularly true since dynamic loading in combination with cryptographic techniques herein for verification make such methods readily tractable.

In general, a private key **156** may be used to produce certifying signatures **162**. A key **156** may also be used to decrypt data received when it has been encrypted using a corresponding public key **160** to ensure privacy.

Both keys **156**, **160** may be necessary for both privacy and integrity, but they are used at opposite ends of a communication. That is, for example, the CMC signature root **152***b* may use the public key **160** of the module signature authority to assure privacy communication to the module signature authority **152***d*. The module signature authority **152***d*, may use the public key **160***b* of the CMC signature root **152***b*. Each **152***b*, **152***d* may use its own private key **156***b*, **156***d* to decrypt received messages. Integrity may be verified by a signature **162** authored using an appropriate private key **156***b*, **156***d*. Meanwhile authenticity of communications, such as a signature **162***d*, created using a private key **156***b* may be verified by an entity using the corresponding, published, public key **160***b*.

As a matter of good cryptographic practice, integrity and confidentiality (privacy) may rely on separate keys. A module **13** may employ a plurality of private/public key pairs **156**/**160**. One pair may be used for channel confidentiality. A separate and distinct pair may be used for channel integrity.

The certificates **154** in the base executable **14**, for example in the module **13**, and loader **90** illustrate authentication of the cascade of certificates. Initially, the modules **13** of FIG. **6** are signed by the signature **168** created with the private key **156***h*.

The public key **160***h* may be used to verify the signature **168**. References to decryption of signatures **168** mean verification, which requires some amount of decryption.

The authenticity of the public key **160***h* is assured by the signature **162***h* on the certificate **154***h*. The signature **162***h* is verified using the public key **160***d* in the certificate **154***d* available.

The authenticity of the public key **160***d* is assured by the signature **162***d* on the certificate **154***d*. The signature **162***d* is verified using the public key **160***b* in the certificate **154***b* available.

This illustrates the practical limit to authentication. The following is not separately illustrated in the architecture, but could be implemented. The authenticity of the public key **160***b* could be assured by the signature **162***b* by obtaining the certificate **154***b*. The signature **162***b* would have to be verified using the public key **160***a* in the certificate **154***a* available. Note that some other mechanism must be used to verify the certificate **154***a*.

A server may generate keys for cryptographic operations. For example, a separate set of keys **156***j* may exist for each client **58** on the network **56**.

Asymmetric systems are more computationally expensive than symmetric systems. The key length used in asymmetric systems is typically much longer than that for symmetric systems (e.g., asymmetric keys may be 1-2 k bits long, versus 40, 64, or 128 bits for typical symmetric keys). In cryptographic protection schemes, an asymmetric algorithm may used to protect a symmetric key that will be distributed to a client **50** encrypted using the client's public key **160** and decrypted by the client's corresponding private key **156**. A shared secret key may be used for shared symmetric key communication in a network **56**. Thus, the server CA private key **156***e* may be used to generate a signature certifying other public/private key pairs **160**, **156**. That pair **156**, **160** may be used to certify another pair or to distribute a symmetric key pair.

A certificate **154** is needed for a public key **160**, and must be signed (**162**) using the corresponding private key **156**. A private key **156**, for example, is used to certify any public key **160** created in the key generation module **42** of FIG. **6**. That is, the key generation module **42** may generate a key pair **156**, **160**; in which the server CA private key **156***e* is used to sign the certificate **154***j* created by the key generation module for the cryptographic libraries. The server CA private key **156** may be used to sign all certificates **154** (with included public key **160**) generated by the CMC filler **12** in the base executable **14** of operating system **14** hosted on the server **60**.

A server key (not shown), which may be symmetric, may be generated by the key generation module **42** and used for key wrapping. All keys that should be kept secret may be wrapped for being transmitted or stored secretly outside of the CMC **12**, such as in a cryptographic library **36**.

Certain of the attributes of a key **156** (algorithm, archive, type, etc.) may be wrapped along with the key **156** before being passed outside of the CMC **12**. Thus a private half of an asymmetric key pair, or a symmetric, secret key should be wrapped preceding any export or output from the CMC **12**.

The libraries **16** may be (typically must be) application-specific, and anything transmitted to them may be considered to be outside of the control of the CMC **12** once it is transmitted to the library **16**.

Escrow is controlled by a manager **14** such as the key generation manager **42**, a cryptographic manager **18**. In any case, every key **156***j* generated should be saved throughout its useful life. A key **156***j* may be saved, typically, in an encrypted format in a secure environment called a key archive **170**. The archived key **156***j* may first be encrypted, and the key **160** to that encryption is the escrow public key **160***k*. The corresponding public key **160***j* is also archived, although it may be publicly available.

The escrow authority **152***f* may be an entity generating a public/private key pair **160**, **156** for each server **60** in order to encrypt (privacy protect) private keys **156** before archival. Thus, the escrow authority **152***f* may have a private key **156***f* unique to itself, which is used to sign **162***k* the certificates **154***k* for all of those public/private key pairs **156***k*, **160***k*. The escrow authority **152***f* may receive its private/public key pair **156***f*, **160***f* from a key escrow root **152***c*. The key escrow root key **156***c* may certify the key **160***f* held by the escrow authority **152***f*. The manufacturer of the base executable **14**, (Novell, in the example above), may be (i.e., control) the key escrow root **152***c*.

The certificate **154***c* held by the key escrow root **152***c* may itself be signed by the root certifier **152***a* certifying the

public key pair **160***c* of the key escrow root **152***c*. Thus, the key escrow path (certifications **166**, cascade) of certificates **154** and keys **156**, **160** may have its source in the root certifier **152***a*, just as the CMC signature root **152***b* does.

An escrow authority **152***f* may hold the private key **156***f* to the archive holding the encrypted, escrowed keys **156**. The archive **170** may actually be inside the server **60**. Thus, the holder of the base executable **14** has all the encrypted keys **156**.

However, a government or some such agency may require certain keys of the escrow authority **152***f*. A manufacturer such as Novell, the operating system manufacturer, in the example above, could also serve this function as well as being the key escrow root **152***c*. This may be advantageous for the same reasons that a manufacturer would be the signature root **152***b*. The escrow authority **152***f* may give to the agent the escrow private key **156***f* for the specific server. This may be the private half **156***k* of an escrow key **156** that the keys **156** in question were encrypted in for archiving. The government may then go to the user of the server **60** to get access to the archive **170** in the server **60** of the owner of all the keys **156***j*.

Some governments may want to be the escrow authority **152***f* for all escrow keys. The government may unlock the key archive **170** whenever desired. In certain countries, the key archive **170** may be in possession of a trusted third party or the government. For example, the key generation module **42** may need to create keys **156***j*, encrypt them, and send them as data to a trusted third party acting for the government to control the archive **170**.

From the above discussion, it will be appreciated that the present invention provides controlled modular cryptography in an executable designed to be embedded within another executable such as a network operating system, or the like. Cryptographic capability is controlled by a manager module operating according to a policy limiting the capability and access of other modules, particularly that of the cryptographic engine. Thus, a system **14** (a base executable **14**) may be provided having nearly all of the capabilities of the "filler" **12** intact. A very limited interface between a filler **12** and its internal engine selection **20** provides for examination of engines **20** by regulatory authority. Moreover, the restricted interfaces **30**, **32** between the engines **20** and the remaining modules **13** of the filler **12** present great difficulty to those who would modify, circumvent, or replace any portion of the filler **12** (CMC **12**) in an attempt to alter its capabilities. Meanwhile, asymmetric key technology provides for enforcement of all controls, thus providing privacy and integrity for all communications, operations, exchanging of keys, and the like.

Referring to FIG. **7**, an operating system **15** may include a loader **90** in an apparatus **10**. The loader **90** may be one of several nested loaders **300**. For example, the loader **90** controls access to a location **302** as described above. Modules **304** may be loaded dynamically by the loader **90**. If no cryptographic capability is to be included in any module **304**, a null engine **48** may be installed in the place of the module **304**. Additionally, any of the modules **13** (see FIG. **1**) may be loaded as the module **304**, as permitted in any of the architectural options described above. Thus, a module **304** may actually include the entire base executable base **14** of a filler **12** in the entire hierarchical arrangement provided.

In the embodiment of FIG. **7**, a nested loader **306** is preferably loaded by the loader **90**. Accordingly, a third party, properly authorized, as described above, may provide a loader **306** to be loaded into the location **302** by the loader **90**. The loader **306** may be authorized by proper certificates **154**, keys **156**, and policies **164** to be properly loaded and linked by the loader **90**. Thus, any module **304**, **308**, or the like may also be an entire hierarchical system **14** of modules **13**.

Also, a loader **310**, properly verified, loaded, and dynamically linked into the location **302**, may include therein modules **312**, which might in turn comprise another loader **314**. Thus, not only may multiple loaders **306**, **310** be loaded into the location **302** by a principal loader **90**, but the loader **310**, may also be adapted to load other loaders **314**. Thus, the loader **314** may recursively implement the properly authorized, verified and controlled operations corresponding to those performed by the loader **90**, within the domain defined and allocated to the loader **314**. That is, the loader **314** may not supplant nor circumvent the loader **90**. Rather, the loader **314** is loaded, linked, and controlled in accordance with the authorizations verified by the loader **90** and implemented thereby. Likewise, the loader **314** may simply be a module **304** of an entire hierarchy **14** of modules **13** (See FIG. **1**). Thus a more restrictive environment provided by an extant policy **130**, **140** may be implemented in a module **316** than in higher-level modules such as the module **304**.

Thus, FIG. **7** illustrates nested loading of multiple modules **304**, **308** which are independent from one another, and concurrently, loading of other modules **312**, **316** recursively. Since the controls implemented by the filler **12** in the operating system **15** may be recursively applied, differing levels of security may be imposed between modules **302**, **308** **312**, and **316** within the apparatus **10**. Typically, greatest security imposed will by within the recursion **90**, **310**, **314**.

Referring to FIG. **8**, an operating system **15** may have multiple loaders **90**, **320** integrated with the operating system **15**. In one embodiment, the loaders **90**, **320** may be "visible" to one another. Alternatively, the loaders **90**, **320** may be completely independent and transparent of one another.

One embodiment, dynamically linked molecules **318**, **322** may not be "aware" of each other s existence. In an alternative embodiment, a link **324** may be established dynamically between the modules **318**, **322**. Because the loaders **90**, **320** operate dynamically according to the infrastructure provided as described above for FIGS. **1-6**, security and control may be assured. As a practical matter, multiple loaders **90**, **320**, aware of one another or not, create no new risk not already accommodated. One may see that the modules **318**, **322** represent a multiplicity of linked filler modules **12** of the basic embodiment **14**.

Thus, relative interdependence or independence of the modules **318**, **322** is a matter of design choice. Similarly, what may be done with an individual module **13**, **304**, **308**, **312**, **316**, **318**, **322** may be done with any number of modules as described above. Accordingly, multiple sets of hierarchies **14** of modules **13**, recursive loading of loaders **306**, **314** by loaders, **90**, **310**, respectively, and loading by plurality of loaders **90**, **320**, may all be encompassed within selected embodiments of an apparatus **10** in accordance with the invention.

Moreover, the loader **90** may be independent of another loader **330** having a separate slot **332** associated therewith. Accordingly, the loader **330** may load a module **334** into the slot **332**. As discussed above, the module **334** may actually implement multiple modules **13**. Additionally, the module **334** may itself include nesting, recursion, and other processes. In the depicted embodiment of the FIG. **8**, the loaders **90**, **330** are typically transparent to one another. Thus, the slot **332**, serviced by the loader **330**, and the slot **336**, serviced by the loader **90** are not available or visible to the

opposing loaders **90**, **330**, respectively. Furthermore, each of the loaders **330** may have separate authorization mechanisms, for example, each may have a separate public or private key or digital signature associated therewith the public keys associated with the different loaders **90**, **330** can be the subject of different distribution channels.

Thus, multiple slots **302**, **330**, **336** serviced by one or more loaders **90**, **320**, **330**, may be controlled as described above to provide a broad spectrum of flexible, yet highly controllable, dynamically integrated executables and data.

The present invention may be embodied in other specific forms without departing from its spirit or essential characteristics. The described embodiments are to be considered in all respects only as illustrative, and not restrictive. The scope of the invention is, therefore indicated by the appended claims, rather than by the foregoing description. All changes which come within the meaning and range of equivalency of the claims are to be embraced within their scope.

What is claimed and desired to be secured by United States Letters Patent is:

1. A method to limit software module integration, comprising:

receiving a first filler module having a first unique property;

dynamically loading the first filler module into a base module using the first loader after the first unique property is verified by the first loader, wherein the base module when loaded with the first filler module permits the first filler module to be executed;

processing a second loader of the first filler module; and

dynamically loading a second filler module after the second loader verifies a second unique property of a second filler module, wherein the second filler module when loaded is capable of being executed, and wherein the first and second loaders are independent loaders from one another.

2. The method of claim **1** wherein in receiving, the base module includes a plurality of slots for integrating the first and second filler modules with the base module after the first unique property and the second unique property are verified.

3. The method of claim **1** wherein in receiving, the base module is an operating system processing on one or more processors.

4. The method of claim **1** wherein in dynamically loading the first and second filler modules, the filler modules are modules wrapped with cryptographic information that identifies the first and second unique properties.

5. The method of claim **4** wherein in dynamically loading the first and second filler modules, the cryptographic information defines access rights to and/or interaction rights between the first and second filler modules with the first and second unique properties.

6. The method of claim **1** wherein in dynamically loading the second filler module, the second loader dynamically links a plurality of addition modules of the second filler module to one another in a layered hierarchy.

7. The method of claim **1** wherein in dynamically loading the first and second filler modules, the first and second unique properties are associated with access policies.

8. A method to verify software module integration, comprising:

identifying a first unique property of a first filler module;

verifying the first unique property before permitting the first filler module to load before subsequent processing;

authorizing the first filler module to process when verified and loaded, and wherein the first filler module acquires a second unique property of a second filler module;

verifying the second unique property by the first filler module before permitting the second filler module to load before subsequent processing; and

authorizing the second filler module for processing when it is verified and loaded, and wherein the first filler module is loaded with an independent loader that is different from a loader that loads the second filler module.

9. The method of claim **8**, wherein in verifying the first and second unique properties, the first unique property is verified by a base loader module and the second unique property is verified by a first filler loader module.

10. The method of claim **8** wherein in identifying the first unique property, the first unique property is verified by a base module integrated into and/or with an operating system.

11. The method of claim **8** wherein in verifying the first and second unique properties, the unique properties are verified by authenticating digital signatures and/or digital certificates associated with the first and second filler modules.

12. The method of claim **8** wherein in authorizing the first and second filler modules for processing, the first and second filler modules are dynamically linked with one another and an operating system before processing.

13. The method of claim **8** wherein in verifying the first and second unique properties, the first and second unique properties are wrapped around the first and second filler modules, respectively, and wherein the unique properties and filler modules are electronically distributed as one or more data structures.

14. The method of claim **13** wherein in verifying the first and second unique properties, the data structures are generated by a software development kit after the filler modules are manufactured.

15. A data structure used for dynamically integrating software modules residing in a computer readable medium, the data structure comprising:

one or more certificates for dynamically authenticating accesses desired by one or more holders of the certificates that desire use of the data structure;

one or more policies for defining access rights and/or interaction rights of the one or more holders to one or more modules;

a policy manager for dynamically enforcing the one or more policies; and

the one or more modules being accessed by the one or more holders, wherein each module includes a loader module for dynamically authenticating and dynamically loading one or more of the remaining modules, and wherein each module is available for execution when loaded, and wherein each loader is independent from other remaining ones of the loaders.

16. The data structure of claim **15**, wherein the data structure is dynamically provided to an operating system for use in authenticating holders and interactions between the one or more modules.

17. The data structure of claim **15** further comprising, one or more cryptographic engines for dynamically authenticating the one or more certificates.

18. The data structure of claim 17 further comprising, a base module associated with each of the one or more modules for plugging into and interfacing with one or more slots of an operating system.

19. The data structure of claim 15, wherein each certificate includes a signature, a public key, and an identifier for one of the holders.

20. The data structure of claim 15, wherein data structure resides in volatile memory of an operating system and is used by the operating system to manage interactions and accesses of the one or more holders to the one or more modules.

* * * * *