(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
2 November 2006 (02.11.2006)

PCT

(10) International Publication Number
WO 2006/115533 A2

(51) International Patent Classification:
H04L 9/00 (2006.01)

(21) International Application Number:
PCT/US2005/030490

(22) International Filing Date: 26 August 2005 (26.08.2005)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/673,979 22 April 2005 (22.04.2005) US
11/116,598 27 April 2005 (27.04.2005) US

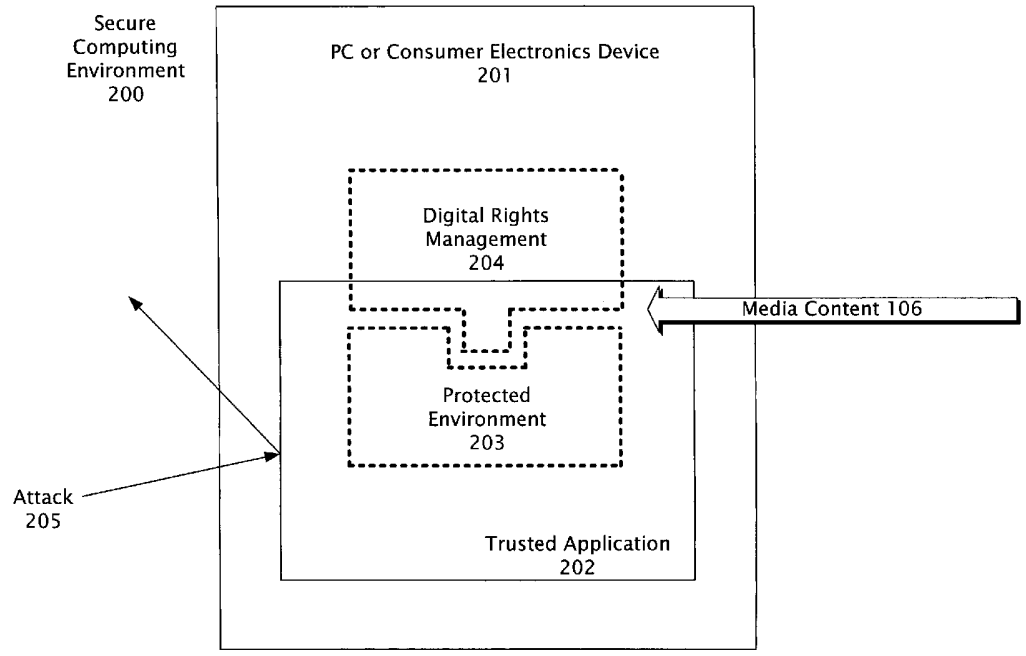
(71) Applicant *(for all designated States except US)*: **MICROSOFT CORPORATION** [US/US]; One Microsoft Way, Redmond, Washington 98052-6399 (US).

(72) Inventors: **BARDE, Sumedh, N.**; One Microsoft Way, Redmond, Washington 98052-6399 (US). **SCHWARTZ, Jonathan, D.**; One Microsoft Way, Redmond, Washington 98052-6399 (US). **KUHN, Reid, Joseph**; One Microsoft Way, Redmond, Washington 98052-6399 (US). **GRIGOROVITCH, Alexandre, Vicktorovich**; One Microsoft Way, Redmond, Washington 98052-6399 (US). **DEBIQUE, Kirt, A.**; One Microsoft Way, Redmond, Washington 98052-6399 (US). **KNOWLTON, Chadd, B.**; One Microsoft Way, Redmond, Washington 98052-6399 (US). **ALKOVE, James, M.**; One Microsoft Way, Redmond, Washington 98052-6399 (US). **DUNBAR, Geoffrey, T.**; One Microsoft Way, Redmond, Washington 98052-6399 (US). **GRIER, Michael, J.**; One Microsoft Way, Redmond, Washington 98052-6399 (US). **MA, Ming**; One Microsoft Way, Redmond, Washington 98052-6399 (US). **UPADHYAY, Chaitanya, D.**; One Microsoft Way, Redmond, Washington 98052-6399 (US). **SHERWANI, Adil, Ahmed**; One Microsoft Way, Redmond, Washington 98052-6399 (US). **KISHAN, Arun, Upadhyaya**; One Microsoft Way, Redmond, Washington 98052-6399 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SM, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

[Continued on next page]

(54) Title: PROTECTED COMPUTING ENVIRONMENT



(57) Abstract: A method of establishing a protected environment within a computing device including validating a kernel component loaded into a kernel of the computing device, establishing a security state for the kernel based on the validation, creating a secure process and loading a software component into the secure process, periodically checking the security state of the kernel, and notifying the secure process when the security state of the kernel has changed.

WO 2006/115533 A2

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:

— *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))*

— *as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii))*

Published:

— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

PROTECTED COMPUTING ENVIRONMENT

CROSS-REFERENCE TO RELATED APPLICATION

[0001] This application claims benefit to US Provisional Patent Application Number 60/673,979 (attorney docket number 313361.01), filed on Friday, April 22, 2005.

DESCRIPTION OF THE DRAWINGS

[0002] These and other features and advantages of the present example will be better understood from the following detailed description read in light of the accompanying drawings, wherein:

[0003] Fig. 1 is a block diagram showing a conventional media application processing media content operating in a conventional computing environment with an indication of an attack against the system.

[0004] Fig. 2 is a block diagram showing a trusted application processing media content and utilizing a protected environment that tends to be resistant to attacks.

[0005] Fig. 3 is a block diagram showing exemplary components of a trusted application that may be included in the protected environment.

[0006] Fig. 4 is a block diagram showing a system for downloading digital media content from a service provider that utilizes an exemplary trusted application utilizing a protected environment.

[0007] Fig. 5 is a block diagram showing exemplary attack vectors that may be exploited by a user or mechanism attempting to access media content and other data typically present in a computing environment in an unauthorized manner.

[0008] Fig. 6 is a flow diagram showing the process for creating and maintaining a protected environment that tends to limit unauthorized access to media content and other data.

[0009] Fig. 7 is a block diagram showing exemplary kernel components and other components utilized for creating an exemplary secure computing environment.

[0010] Fig. 8 and Fig. 9 are flow diagrams showing an exemplary process for loading kernel components to create an exemplary secure computing environment.

[0011] Fig. 10 is a block diagram showing a secure computing environment loading an application into an exemplary protected environment to form a trusted application that is typically resistant to attacks.

[0012] Fig. 11 is a flow diagram showing an exemplary process for creating a protected environment and loading an application into the protected environment.

[0013] Fig. 12 is a block diagram showing an exemplary trusted application utilizing an exemplary protected environment periodically checking the security state of the secure computing environment.

[0014] Fig. 13 is a flow diagram showing an exemplary process for periodically checking the security state of the secure computing environment.

[0015] Fig. 14 is a block diagram showing an exemplary computing environment in which the processes, systems and methods for establishing a secure computing environment including a protected environment may be implemented.

[0016] Like reference numerals are used to designate like elements in the accompanying drawings.

DETAILED DESCRIPTION

[0017] The detailed description provided below in connection with the appended drawings is intended as a description of the present examples and is not intended to represent the only forms in which the present examples may be constructed or utilized. The description sets forth the functions of the examples and the sequence of steps for constructing and operating the examples in connection with the examples illustrated. However, the same or equivalent functions and sequences may be accomplished by different examples.

[0018] Although the present examples are described and illustrated herein as being implemented in a computer operating system, the system described is provided as an example and not a limitation. As those skilled in the art will appreciate, the present examples are suitable for application in a variety of different types of computer systems.

INTRODUCTION

[0019] Fig. 1 is a block diagram showing a conventional media application 105 processing media content 106 operating in a conventional computing environment 100

with an indication of an attack 107 against the system 101. A conventional computing environment 100 may be provided by a personal computer ("PC") or consumer electronics ("CE") device 101 that may include operating system ("OS") 102. Typical operating systems often partition their operation into a user mode 103, and a kernel mode 104. User mode 103 and kernel mode 104 may be used by one or more application programs 105. An application program 105 may be used to process media content 106 that may be transferred to the device 101 via some mechanism, such as a CD ROM drive, Internet connection or the like. An example of content 106 would be media files that may be used to reproduce audio and video information.

[0020] The computing environment 100 may typically include an operating system ("OS") 102 that facilitates operation of the application 105, in conjunction with the one or more central processing units ("CPU"). Many operating systems 102 may allow multiple users to have access to the operation of the CPU. Multiple users may have ranges of access privileges typically ranging from those of a typical user to those of an administrator. Administrators typically have a range of access privileges to applications 105 running on the system, the user mode 103 and the kernel 104. Such a computing environment 100 may be susceptible to various types of attacks 107. Attacks may include not only outsiders seeking to gain access to the device 101 and the content 106 on it, but also attackers having administrative rights to the device 101 or other types of users having whatever access rights granted them.

[0021] Fig. 2 is a block diagram showing a trusted application 202 processing media content 106 and utilizing a protected environment 203 that tends to be resistant to attack 205. The term "trusted application", as used here, may be defined as an application that utilizes processes operating in a protected environment such that they tend to be resistant to attack 205 and limit unauthorized access to any media content 106 or other data being processed. Thus, components or elements of an application operating in a protected environment are typically considered "trusted" as they tend to limit unauthorized access and tend to be resistant to attack. Such an application 202 may be considered a trusted application itself or it may utilize another trusted application to protect a portion of its processes and/or data.

[0022] For example, a trusted media player 202 may be designed to play media content 106 that is typically licensed only for use such that the media content 106 cannot be accessed in an unauthorized manner. Such a trusted application 202 may not operate and/or process the media content 106 unless the computing environment 200 can provide the required level of security, such as by providing a protected environment 203 resistant to attack 205.

[0023] As used herein, the term "process" can be defined as an instance of a program (including executable code, machine instructions, variables, data, state information, etc.) residing and/or operating in a kernel space, user space and/or any other space of an operating system and/or computing environment.

[0024] A digital rights management system 204 or the like may be utilized with the protected environment 203. The use of a digital rights management system 204 is merely provided as an example and may not be utilized with a protected environment or a secure computing environment. Typically a digital rights management system utilizes tamper-resistant software ("TRS") which tends to be expensive to produce and may negatively impact computing performance. Utilizing a trusted application 202 may minimize the amount of TRS functionality required to provide enhanced protection.

[0025] Various mechanisms known to those skilled in this technology area may be utilized in place of, in addition to, or in conjunction with a typical digital rights management system. These mechanisms may include, but are not limited to, encryption/decryption, key exchanges, passwords, licenses, and the like. Thus, digital right management as used herein may be a mechanism as simple as decrypting an encrypted media, utilizing a password to access data, or other tamper-resistant mechanisms. The mechanisms to perform these tasks may be very simple and entirely contained within the trusted application 202 or may be accessed via interfaces that communicate with complex systems otherwise distinct from the trusted application 202.

[0026] Fig. 3 is a block diagram showing exemplary components of a trusted application 202 that may be included in the protected environment 203. A trusted application 202 will typically utilize a protected environment 203 for at least a potion of its subcomponents 302-304. Other components 301 of the trusted application may not

utilize a protected environment. Components 302-204 involved in the processing of media content or data that may call for an enhanced level of protection from attack or unauthorized access may operate within a protected environment 203. A protected environment 203 may be utilized by a single trusted application 202 or, possibly, by a plurality of trusted applications. Alternatively, a trusted application 202 may utilize a plurality of protected environments. A trusted application 202 may also couple to and/or utilize a digital rights management system 204.

[0027] In the example shown, source 302 and sink 303 are shown as part of a media pipeline 304 operating in the protected environment 203. A protected environment 203 tends to ensure that, once protected and/or encrypted content 309 has been received and decrypted, the trusted application 202 and its components prevent unauthorized access to the content 309.

[0028] Digital rights management 204 may provide a further avenue of protection for the trusted application 202 and the content 309 it processes. Through a system of licenses 308, device certificates 311, and other security mechanisms a content provider is typically able to have confidence that encrypted content 309 has been delivered to the properly authorized device and that the content 309 is used as intended.

[0029] Fig. 4 is a block diagram showing a system for downloading digital media content 410 from a service provider 407 to an exemplary trusted application 202 utilizing a protected environment 203. In the example shown the trusted application 202 is shown being employed in two places 401, 403. The trusted application 202 may be used in a CE device 401 or a PC 403. Digital media 410 may be downloaded via a service provider 407 and the Internet 405 for use by the trusted application 202. Alternatively, digital media may be made available to the trusted application via other mechanisms such as a network, a CD or DVD disk, or other storage media. Further, the digital media 410 may be provided in an encrypted form 309 requiring a system of decryption keys, licenses, certificates and/or the like which may take the form of a digital rights management system 204. The data or media content 410 provided to the trusted application may or may not be protected, i.e, encrypted or the like.

[0030] In one example, a trusted application 202 may utilize a digital rights management ("DRM") system 204 or the like along with a protected environment 203. In this case, the trusted application 202 is typically designed to acknowledge, and adhere to, the content's usage policies by limiting usage of the content to that authorized by the content provider via the policies. Implementing this may involve executing code which typically interrogates content licenses and subsequently makes decisions about whether or not a requested action can be taken on a piece of content. This functionality may be provided, at least in part, by a digital rights management system 204. An example of a Digital Rights Management system is provided in U.S. Patent Application No. 09/290,363, filed April 12, 1999, U.S. Patent Applications Nos. 10/185,527, 10/185,278, and 10/185,511, each filed on June 28, 2002 which are hereby incorporated by reference in its entirety.

[0031] Building a trusted application 202 that may be utilized in the CE device 401 or the PC 403 may include making sure the trusted application 202 which decrypts and processes the content 309 may be "secure" from malicious attacks. Thus, a protected environment 203 typically refers to an environment that may not be easy to attack.

[0032] As shown, the trusted applications 202 operate in a consumer electronics device 401, which may be periodically synced to a PC 403 that also provides a trusted application. The PC 403 is in turn coupled 404 to the internet 405. The internet connection allows digital media 410 to be provided by a service provider 407. The service provider 407 may transmit licenses and encrypted media 406 over the internet 405 to trusted application 202. Once encrypted media is delivered and decrypted it may be susceptible to various forms of attack.

PROTECTED ENVIRONMENTS AND POTENTIAL ATTACKS

[0033] A protected computing environment tends to provide an environment that limits hackers from gaining access to unauthorized content. A hacker may include hackers acting as a systems administrator. A systems administrator typically has full control of virtually all of the processes being executed on a computer, but this access may not be desirable. For example, if a system user has been granted a license to use a

media file should not be acceptable for a system administrator different from the user to be able to access the media file. A protected environment tends to contribute to the creation of a process in which code that decrypts and processes content can operate without giving hackers access to the decrypted content. A protected environment may also limit unauthorized access to users of privilege, such as administrators, and/or any other user, who may otherwise gain unauthorized access to protected content. Protection may include securing typical user mode processes (Fig. 1, 103) and kernel mode processes (Fig. 1, 104) and any data they may be processing.

[0034] Processes operating in the kernel may be susceptible to attack. For example, in the kernel of a typical operating system objects are created, including processes, that may allow unlimited access by an administrator. Thus, an administrator, typically with full access privileges, may access virtually all processes.

[0035] Protected content may include policy or similar information indicating the authorized use of the content. Such policy may be enforced via a DRM system or other security mechanism. Typically, access to protected content is granted through the DRM system or other mechanism, which may enforce policy. However, a system administrator, with full access to the system, may alter the state of the DRM system or mechanism to disregard the content policy.

[0036] A protected environment tends to provide a protected space that restricts unauthorized access to media content being processed therein, even for high-privilege users such as an administrator. When a protected environment is used in conjunction with a system of digital rights management or the like, a trusted application may be created in which a content provider may feel that adequate security is provided to protect digital media from unauthorized access and may also protect the content's policy from be tampered with along with any other data, keys or protection mechanisms that may be associated with the media content.

ATTACK VECTORS

[0037] Current operating system ("OS") architectures typically present numerous possible attack vectors that could compromise a media application and any digital media content being processed. For purposes of this example, attacks that may occur in an OS

are grouped into two types of attacks, which are kernel mode attacks and user mode attacks.

[0038] The first type of attack is the kernel mode attack. Kernel mode is typically considered to be the trusted base of the operating system. The core of the operating system and most system and peripheral drivers may operate in kernel mode. Typically any piece of code running in the kernel is susceptible to intrusion by any other piece of code running in the kernel, which tends not to be the case for user mode. Also, code running in kernel mode typically has access to substantially all user mode processes. A CPU may also provide privilege levels for various code types. Kernel mode code is typically assigned the highest level of privilege by such a CPU, typically giving it full access to the system.

[0039] The second type of attack is the user mode attack. Code that runs in user mode may or may not be considered trusted code by the system depending on the level of privilege it has been assigned. This level of privilege may be determined by the user context or account in which it is operating. User mode code running in the context of an administrator account may have full access to the other code running on the system. In addition, code that runs in user mode may be partitioned to prevent one user from accessing another's processes.

[0040] These attacks may be further broken down into specific attack vectors. The protected environment is typically designed to protect against unauthorized access that may otherwise be obtained via one or more of these attack vectors. The protected environment may protect against attack vectors that may include: process creation, malicious user mode applications, loading malicious code into a process, malicious kernel code, invalid trust authorities, and external attack vectors.

[0041] Process creation is a possible attack vector. An operating system typically includes a "create process" mechanism that allows a parent process to create a child process. A malicious parent process may, by modifying the create process code or by altering the data it creates, make unauthorized modifications to the child process being created. This could result in compromising digital media that may be processed by a child process created by a malicious parent process.

[0042] Malicious user mode applications are a possible attack vector. An operating system typically includes administrator level privileges. Processes running with administrator privileges may have unlimited access to many operating system mechanisms and to nearly all processes running on the computer. Thus, in Windows for example, a malicious user mode application running with administrator privileges may gain access to many other processes running on the computer and may thus compromise digital media. Similarly, processes operating in the context of any user may be attacked by any malicious process operating in the same context.

[0043] Loading malicious code into a secure process is a possible attack vector. It may be possible to append or add malicious code to a process. Such a compromised process cannot be trusted and may obtain unauthorized access to any media content or other data being processed by the modified process.

[0044] Malicious kernel mode code is a possible attack vector. An operating system typically includes a "system level" of privilege. In Windows, for example, all code running in kernel mode is typically running as system and therefore may have maximum privileges. The usual result is that drivers running in kernel mode may have maximum opportunity to attack any user mode application, for example. Such an attack by malicious kernel mode code may compromise digital media.

[0045] Invalid trust authorities (TAs) are a possible attack vector. TAs may participate in the validation of media licenses and may subsequently "unlock" the content of a digital media. TAs may be specific to a media type or format and may be implemented by media providers or their partners. As such, TAs may be pluggable and/or may be provided as dynamic link libraries ("DLL") or the like. A DLL may be loaded by executable code, including malicious code. In order for a TA to ensure that the media is properly utilized it needs to be able to ensure that the process in which it is running is secure. Otherwise the digital media may be compromised.

[0046] External attacks are another possible attack vector. There are a set of attacks that don't require malicious code running in a system in order to attack it. For instance, attaching a debugger to a process or a kernel debugger to the machine, looking

for sensitive data in a binary file on a disk, etc., are all possible mechanisms for finding and compromising digital media or the processes that can access digital media.

[0047] Fig. 5 is a block diagram showing exemplary attack vectors 507-510 that may be exploited by a user or mechanism attempting to access media content and other data 500 typically present in a computing environment 100 in an unauthorized manner. A protected environment may protect against these attack vectors such that unauthorized access to trusted applications and the data they process is limited and resistance to attack is provided. Such attacks may be waged by users of the system or mechanisms that may include executable code. The media application 105 is shown at the center of the diagram and the attack vectors 507-510 tend to focus on accessing sensitive data 500 being stored and/or processed by the application 105.

[0048] A possible attack vector 509 may be initiated via a malicious user mode application 502. In the exemplary operating system architecture both the parent of a process, and any process with administrative privileges, typically have unlimited access to other processes, such as one processing media content, and the data they process. Such access to media content may be unauthorized. Thus a protected environment may ensure that a trusted application and the media content it processes are resistant to attacks by other user mode applications.

[0049] A possible attack vector 508 is the loading of malicious code 503 into a process 501. Having a secure process that is resistant to attacks from the outside is typically only as secure as the code running on the inside forming the process. Given that DLLs and other code are typically loaded into processes for execution, a mechanism that may ensure that the code being loaded is trusted to run inside a process before loading it into the process may be provided in a protected environment.

[0050] A possible vector of attack 510 is through malicious kernel mode code 504. Code running in kernel mode 104 typically has maximum privileges. The result may be that drivers running in kernel mode may have a number of opportunities to attack other applications. For instance, a driver may be able to access memory directly in another process. The result of this is that a driver could, once running, get access to a processes memory which may contain decrypted "encrypted media content" (Fig. 3, 309).

Kernel Mode attacks may be prevented by ensuring that the code running in the kernel is non-malicious code, as provided by this example.

[0051] A possible attack vector 507 is by external attacks 506 to the system 100. This group represents the set of attacks that typically do not require malicious code to be running on the system 100. For instance, attaching a debugger to an application and/or a process on the system, searching a machine for sensitive data, etc. A protected environment may be created to resist these types of attacks.

CREATING AND MAINTAINING PROTECTED ENVIRONMENTS

[0052] Fig. 6 is a flow diagram showing the process 600 for creating and maintaining a protected environment that tends to limit unauthorized access to media content and other data. The sequence 600 begins when a computer system is started 602 and the kernel of the operating system is loaded and a kernel secure flag is set 604 to an initial value. The process continues through the time that a protected environment is typically created and an application is typically loaded into it 606. The process includes periodic checking 608 via the protected environment that seeks to ensure the system remains secure through the time the secure process is needed.

[0053] The term "kernel", as used here, is defined as the central module of an operating system for a computing environment, system or device. The kernel module may be implemented in the form of computer-executable instructions and/or electronic logic circuits. Typically, the kernel is responsible for memory management, process and task management, and storage media management of a computing environment. The term "kernel component", as used here, is defined to be a basic controlling mechanism, module, computer-executable instructions and/or electronic logic circuit that forms a portion of the kernel. For example, a kernel component may be a "loader", which may be responsible for loading other kernel components in order to establish a fully operational kernel.

[0054] To summarize the process of creating and maintaining a protected environment:

[0055] 1. Block 602 represents the start-up of a computer system. This typically begins what is commonly known as the boot process and includes loading of an operating system from disk or some other storage media.

[0056] 2. Typically one of the first operations during the boot process is the loading of the kernel and its components. This example provides the validation of kernel components and, if all are successfully validated as secure, the setting of a flag indicating the kernel is secure. This is shown in block 604.

[0057] 3. After the computer system is considered fully operational a user may start an application such as a trusted media player which may require a protected environment. This example provides a secure kernel with an application operating in a protected environment, as shown in block 606.

[0058] 4. Once the protected environment has been created and one or more of the processes of the application have been loaded into it and are operating, the trusted environment may periodically check the kernel secure flag to ensure the kernel remains secure, as shown in block 608. That is, from the point in time that the trusted application begins operation, a check may be made periodically to determine whether any unauthorized kernel components have been loaded. Such unauthorized kernel components could attack the trusted application or the data it may be processing. Therefore, if any such components are loaded, the kernel secure flag may be set appropriately.

LOADING AND VALIDATING A SECURE KERNEL

[0059] Fig. 7 is a block diagram showing exemplary kernel components 720-730 and other components 710-714 utilized in creating an exemplary secure computing environment 200. This figure shows a computer system containing several components 710-730 typically stored on a disk or the like, several of which are used to form the kernel of an operating system when a computer is started. Arrow 604 indicates the process of loading the kernel components into memory forming the operational kernel of the system. The loaded kernel 750 is shown containing its various components 751-762 and a kernel secure flag 790 indicating whether or not the kernel is considered secure for a protected environment. The kernel secure flag 790 being described as a "flag" is not

meant to be limiting; it may be implemented as a boolean variable or as a more complex data structure or mechanism.

[0060] Kernel components 720-730 are typically "signed" and may include a certificate data 738 that may allow the kernel to validate that they are the components they claim to be, that they have not been modified and/or are not malicious. A signature block and/or certificate data 738 may be present in each kernel component 720-730 and/or each loaded kernel component 760, 762. The signature and/or certificate data 738 may be unique to each component. The signature and/or certificate data 738 may be used in the creation and maintenance of protected environments as indicated below. Typically a component is "signed" by its provider in such as way as to securely identify the source of the component and/or indicate whether it may have been tampered with. A signature may be implemented as a hash of the component's header or by using other techniques. A conventional certificate or certificate chain may also be included with a component that may be used to determine if the component can be trusted. The signature and/or certificate data 738 are typically added to a component before it is distributed for public use. Those skilled in the art will be familiar with these technologies and their use.

[0061] When a typical computer system is started or "booted" the operating system's loading process or "kernel loader" 751 may typically load the components of the kernel from disk or the like into a portion of system memory to form the kernel of the operating system. Once all of the kernel components are loaded and operational the computer and operating system are considered "booted" and ready for normal operation.

[0062] Kernel component #1 720 thru kernel component #n 730, in the computing environment, may be stored on a disk or other storage media, along with a revocation list 714, a kernel dump flag 712 and a debugger 710 along with a debug credential 711. Arrow 604 indicates the kernel loading process which reads the various components 714-730 from their storage location and loads them into system memory forming a functional operating system kernel 750. The kernel dump flag 712 being described as a "flag" is not meant to be limiting; it may be implemented as a boolean variable or as a more complex data structure or mechanism.

[0063] The kernel loader 751 along with the PE management portion of the kernel 752, the revocation list 754 and two of the kernel components 720 and 722 are shown loaded into the kernel, the latter as blocks 760 and 762, along with an indication of space for additional kernel components yet to be loaded into the kernel, 764 and 770. Finally, the kernel 750 includes a kernel secure flag 790 which may be used to indicate whether or not the kernel 750 is currently considered secure or not. This illustration is provided as an example and is not intended to be limiting or complete. The kernel loader 751, the PE management portion of the kernel 752 and/or the other components of the kernel are shown as distinct kernel components for clarity of explanation but, in actual practice, may or may not be distinguishable from other portions of the kernel.

[0064] Included in the computing environment 200 may be a revocation list 714 that may be used in conjunction with the signature and certificate data 738 associated with the kernel components760 and 762. This object 714 may retain a list of signatures, certificates and/or certificate chains that are no longer considered valid as of the creation date of the list 714. The revocation list 714 is shown loaded into the kernel as object 754. Such lists are maintained because a validly-signed and certified component, for example components 760 and 762, may later be discovered to have some problem. The system may use such a list 754 to check kernel components 720-730 as they are loaded, which may be properly signed and/or have trusted certificate data 738, but that may have subsequently been deemed untrustworthy. Such a revocation list 754 will typically include version information 755 so that it can more easily be identified, managed and updated as required.

[0065] Another component of the system that may impact kernel security is a debugger 710. Debuggers may not typically be considered a part of the kernel but may be present in a computing environment 200. Debuggers, including those known as kernel debuggers, system analyzers, and the like, may have broad access to the system and the processes running on the system along with any data. A debugger 710 may be able access any data in a computing environment 200, including media content that should not be accessed in a manner other than that authorized. On the other hand, debugging is typically a part of developing new functionality and it typically is possible to debug within

protected environments the code intended to process protected media content. A debugger 710 may thus include debug credentials 711 which may indicate that the presence of the debugger 710 on a system is authorized. Thus detection of the presence of a debugger 710 along with any accompanying credentials 711 may be a part of the creation and maintenance of protected environments (Fig. 6, 600).

[0066] The computing environment 200 may include a kernel dump flag 712. This flag 712 may be used to indicate how much of kernel memory is available for inspection in case of a catastrophic system failure. Such kernel dumps may be used for postmortem debugging after such as failure. If such a flag 712 indicates that substantially all memory is available for inspection upon a dump then the kernel 750 may be considered insecure as hacker could run an application which exposes protected media in system memory and then force a catastrophic failure condition which may result in the memory being available for inspection including that containing the exposed media content. Thus a kernel dump flag 712 may be used in the creation and maintenance of a protected environments (Fig. 6, 600).

[0067] Fig. 8 and Fig. 9 are flow diagrams showing an exemplary process 604 for loading kernel components to create an exemplary secure computing environment. This process 604 begins after the kernel loader has been started and the PE management portion of the kernel has been loaded and made operational. Not shown in these figures, the PE management portion of the kernel may validate the kernel loader itself and/or any other kernel elements that may have been previously loaded. Validation may be defined as determining whether or not a given component is considered secure and trustworthy as illustrate in part 2 of this process 604.

[0068] The term "authorized for secure use" and the like as used below with respect to kernel components has the following specific meaning. A kernel containing any components that are not authorized for secure use does not provide a secure computing environment within which protected environments may operate. The opposite may not be true as it depends on other factors such as attack vectors.

[0069] 1. Block 801 shows the start of the loading process 604 after the PE management portion of the kernel has been loaded and made operational. Any component loaded in the kernel prior to this may be validated as described above.

[0070] 2. Block 802 shows that the kernel secure flag initially set to TRUE unless any component loaded prior to the PE management portion of the kernel, or that component itself, is found to be insecure at which point the kernel secure flag may be set to FALSE. In practice the indication of TRUE or FALSE may take various forms; the use of TRUE or FALSE here is only an example and is not meant to be limiting.

[0071] 3. Block 804 indicates a check for the presence of a debugger in the computing environment. Alternatively a debugger could reside remotely and be attached to the computing environment via a network or other communications media to a process in the computing environment. If no debugger is detected the loading process 604 continues at block 810. Otherwise it continues at block 809. Not shown in the diagram, this check may be performed periodically and the state of the kernel secure flag updated accordingly.

[0072] 4. If a debugger is detected, block 806 shows a check for debug credentials which may indicate that debugging may be authorized on the system in the presence of a protected environment. If such credentials are not present, the kernel secure flag may be set to FALSE as shown in block 808. Otherwise the loading process 604 continues at block 810.

[0073] 5. Block 810 shows a check of the kernel dump flag. If this flag indicates that a full kernel memory dump or the like may be possible then the kernel secure flag may be set to FALSE as shown in block 808. Otherwise the loading process 604 continues at block 812. Not shown in the diagram, this check may be performed periodically and the state of the kernel secure flag updated accordingly.

[0074] 6. Block 812 shows the loading of the revocation list into the kernel. In cases where the revocation list may be used to check debug credentials, or other previously loaded credentials, signatures, certificate data, or the like, this step may take place earlier in the sequence (prior to the loading of credentials and the like to be checked) than shown. Not shown in the diagram is that, once this component is loaded,

any and all previously loaded kernel components may be checked to see if their signature and/or certificate data has been revoked per the revocation list. If any have been revoked, the kernel secure flag may be set to FALSE and the loading process 604 continues at block 814. Note that a revocation list may or may not be loaded into the kernel to be used in the creation and maintenance of a protected environments.

[0075] 7. Block 814 shows the transition to part 2 of this diagram shown in figure 9 and continuing at block 901.

[0076] 8. Block 902 shows a check for any additional kernel components to be loaded. If all components have been loaded then the load process 604 is usually complete and the kernel secure flag remains in whatever state it was last set to, either TRUE or FALSE. If there are additional kernel components to be loaded the load process 604 continues at block 906.

[0077] 9. Block 906 shows a check for a valid signature of the next component to be loaded. If the signature is invalid then the kernel secure flag may be set to FALSE as shown in block 918. Otherwise the loading process 604 continues at block 908. If no component signature is available the component may be considered insecure and the kernel secure flag may be set to FALSE as shown in block 918. Signature validity may be determined by checking for a match on a list of valid signatures and/or by checking whether the signer's identity is a trusted identity. As familiar to those skilled in the security technology area, other methods could also be used to validate component signatures.

[0078] 10. Block 908 shows a check of the component's certificate data. If the certificate data is invalid then the kernel secure flag may be set to FALSE as shown in block 918. Otherwise the loading process 604 continues at block 910. If no component certificate data is available the component may be considered insecure and the kernel secure flag may be set to FALSE as shown in block 918. Certificate data validity may be determined by checking the component's certificate data to see if the component is authorized for secure use. As familiar to those skilled in the art, other methods could also be used to validate component certificate data.

[0079] 11. Block 910 shows a check of the component's signature against a revocation list loaded in the kernel. If the signature is present on the list, indicating that it has been revoked, then the kernel secure flag may be set to FALSE as shown in block 918. Otherwise the loading process 604 continues at block 912.

[0080] 12. Block 912 shows a check of the component's certificate data against a revocation list. If the certificate data is present on the list, indicating that it has been revoked, then the kernel secure flag may be set to FALSE as shown in block 918. Otherwise the loading process 604 continues at block 914.

[0081] 13. Block 914 shows a check of the component's signature to determine if it is OK for use. This check may be made by inspecting the component's leaf certificate data to see if the component is authorized for secure use. Certain attributes in the certificate data may indicate if the component is approved for protected environment usage. If not the component may not be appropriately signed and the kernel secure flag may be set to FALSE as shown in block 918. Otherwise the loading process 604 continues at block 916.

[0082] 14. Block 916 shows a check of the component's root certificate data. This check may be made by inspecting the component's root certificate data to see if it is listed on a list of trusted root certificates. If not the component may be considered insecure and the kernel secure flag may be set to FALSE as shown in block 918. Otherwise the loading process 604 continues at block 920.

[0083] 15. Block 920 shows the loading of the component into the kernel where it is now considered operational. Then the loading process 604 returns to block 902 to check for any further components to be loaded.

CREATING PROTECTED ENVIRONMENTS

[0084] Fig. 10 is a block diagram showing a secure computing environment 200 loading an application 105 into an exemplary protected environment 203 to form a trusted application that is typically resistant to attacks. In this example the kernel may be the same as that described in Fig. 7, has already been loaded and the system 200 is considered fully operational. At this point, as an example, a user starts media application 105. The media application 105 may call for the creation of a protected environment 203

for one or more of its processes and/or components to operate within. The protected environment creation process 606 creates the protected environment 203 and loads the application 105 and/or its components as described below.

[0085] Fig. 11 is a flow diagram showing an exemplary process 606 for creating a protected environment and loading an application into the protected environment. This process 606 includes the initial step of creating a secure process followed by validating the software component to be loaded into it and then loading the software component into the new secure process and making it operational. Upon success, the result may be a software component operating in a protected environment supported by a secure kernel. Such a software component, along with any digital media content or other data it processes, may be protected from various attacks, including those described above.

[0086] 1. Block 1101 shows the start of the protected environment creation process 606. This point is usually reached when some application or code calls for a protected environment to operate.

[0087] 2. Block 1102 shows the establishment of a protected environment. While not shown in the diagram, this may be accomplished by requesting the operating system to create a new secure process. Code later loaded and operating in this secure process may be considered to be operating in a protected environment. If the kernel secure flag is set to FALSE then the "create new secure process" request may fail. This may be because the system as a whole may be considered insecure and unsuitable for a protected environment and any application or data requiring a protected environment. Alternatively, the "create new secure process" request may succeed and the component loaded into the new process may be informed that the system is considered insecure so that it can modify its operations accordingly. Otherwise the process 606 continues at block 1106.

[0088] 3. Block 1106 shows a check for a valid signature of the software component to be loaded into the new secure process or protected environment. If the signature is invalid then the process 606 may fail as shown in block 1118. Otherwise the process 606 continues at block 1108. Not shown in the process is that the program, or its equivalent, creating the new secure process may also be checked for a valid signature.

Thus, for either the component itself and/or the program creating the new secure process, if no signature is available the component may be considered insecure and the process 606 may fail as shown in block 1118. Signature validity may be determined by checking for a match on a list of valid signatures and/or by checking whether the signer's identity is a trusted identity. As familiar to those skilled in the security technology area, other methods could also be used to validate component signatures.

[0089] 4. Block 1108 shows a check of the software component's certificate data. If the certificate data is invalid then the process 606 may fail as shown in block 1118. Otherwise the process 606 continues at block 1110. If no component certificate data is available the component may be considered insecure and the process 606 may fail as shown in block 1118. Certificate data validity may be determined by checking the component's certificate data to see if the component is authorized for secure use. As familiar to those skilled in the art, other methods could also be used to validate component certificate data.

[0090] 5. Block 1110 shows a check of the component's signature against a revocation list. If the signature is present on the list, indicating that it has been revoked, then the process 606 may fail as shown in block 1118. Otherwise the process 606 continues at block 1112.

[0091] 12. Block 1112 shows a check of the component's certificate data against a revocation list. If the certificate data is present on the list, indicating that it has been revoked, then the process 606 may fail as shown in block 1118. Otherwise the process 606 continues at block 1114.

[0092] 13. Block 1114 shows a check of the component's signature to determine if it is acceptable for use. This check may be made by inspecting the component's leaf certificate data to see if the component is authorized for secure use. Certain attributes in the certificate data may indicate if the component is approved for protected environment usage. If not the component may be considered to not be appropriately signed and the process 606 may fail as shown in block 1118. Otherwise the process 606 continues at block 1116.

[0093] 14. Block 1116 shows a check of the component's root certificate data. This check may be made by inspecting the component's root certificate data to see if it is listed on a list of trusted root certificates. If not the component may be considered insecure and the process 606 may fail as shown in block 1118. Otherwise the process 606 continues at block 1120.

[0094] 15. Block 1118 shows the failure of the software component to load followed by block 1130, the end of the protected environment creation process 606.

[0095] 16. Block 1120 shows the software component being loaded into the protected environment, where it is considered operational, followed by block 1130, the end of the protected environment creation process 606.

VALIDATING A SECURE KERNEL OVER TIME

[0096] Fig. 12 is a block diagram showing an exemplary trusted application utilizing an exemplary protected environment 202 periodically checking 608 the security state 790 of the secure computing environment 200. In this example, the computing environment 200 and the kernel 750 may be the same as those described in figures 7 and 8. The kernel 750 has already been loaded and the computer 200 is considered fully operational. Further, a protected environment has been created and the appropriate components of the trusted application have been loaded into it and made operational, establishing a trusted application utilizing a protected environment 202, hereafter referred to simply as the "protected environment".

[0097] The protected environment 202 may periodically check with the PE management portion of the kernel 752 to determine whether the kernel 750 remains secure over time. This periodic check may be performed because it is possible for a new component to be loaded into the kernel 750 at any time, including a component that may be considered insecure. If this were to occur, the state of the kernel secure flag 790 would change to FALSE and the code operating in the protected environment 202 has the opportunity to respond appropriately.

[0098] For example, consider a media player application that was started on a PC 200 with a secure kernel 750 and a portion of the media player application operating in a protected environment 202 processing digital media content that is licensed only for

secure use. In this example, if a new kernel component that is considered insecure is loaded while the media player application is processing the media content, then the check kernel secure state process 240 would note the kernel secure flag 790 has changed to FALSE indicating the kernel 750 may no longer be secure.

[0099] Alternatively, the revocation list 745 may be updated and a kernel component previously considered secure may no longer be considered secure, resulting in the kernel secure flag 790 being set to FALSE. At this point the application may receive notification that the system 200 is no longer considered secure and can terminate operation, or take other appropriate action to protect itself and/or the media content it is processing.

[00100] Fig. 13 is a flow diagram showing an exemplary process 608 for periodically checking the security state of the secure computing environment. This process 608 may be used by a protected environment 202 to determine if the kernel remains secure over time. The protected environment 202 may periodically use this process 608 to check the current security status of the kernel. The protected environment 202 and/or the software component operating within it may use the current security status information to modify its operation appropriately. Periodic activation of the process may be implemented using conventional techniques.

[00101] The diagram shows a sequence of communications 608, illustrated with exemplary pseudo code, between the protected environment 202 and the PE management portion of the kernel 752. This communication may include a check of the version of a revocation list which may give an application the ability to specify a revocation list of at least a certain version. This communications sequence may be cryptographically secured using conventional techniques.

[00102] 1. The protected environment 202 makes a IsKernelSecure(MinRLVer) call 1320 to the PE management portion of the kernel to query the current security state of the kernel. Included in this call 1320 may be the minimum version (MinRLVer) of the revocation list expected to be utilized.

[00103] 2. The PE management portion of the kernel checks to see if the protected environment, which is the calling process, is secure. If not, then it may provide

a Return(SecureFlag=FALSE) indication 1322 to the protected environment and the communications sequence 608 is complete. This security check may be done by the PE management portion of the kernel checking the protected environment for a valid signature and/or certificate data as described above.

[00104] 3. Otherwise, the PE management portion of the kernel checks the kernel secure flag in response to the call 1320. If the state of the flag is FALSE then it may provide a Return(SecureFlag=FALSE) indication 1324 to the protected environment and the communications sequence 608 is complete.

[00105] 4. Otherwise, the PE management portion of the kernel checks the revocation list version information for the revocation list. If the revocation list has version information that is older than that requested in the IsKernelSecure(MinRLVer) call 1320 then several options are possible. First, as indicated in the diagram, the PE management portion of the kernel may provide a Return(SecureFlag=FALSE) indication 1326 to the protected environment and the communications sequence 608 is complete.

[00106] Alternatively, and not shown in the diagram, an appropriate version revocation list may be located and loaded into the kernel, all kernel components could be re-validated using this new or updated list, the kernel secure flag updated as appropriate and the previous step #3 of this communications sequence 608 repeated.

[00107] 5. Otherwise, the PE management portion of the kernel may provide a Return(SecureFlag=TRUE) indication 1328 to the protected environment and the communications sequence 608 is complete.

[00108] EXEMPLARY COMPUTING ENVIRONMENT

[00109] Fig. 14 is a block diagram showing an exemplary computing environment 1400 in which the processes, systems and methods for establishing a secure computing environment including a protected environment 203 may be implemented. Exemplary personal computer 1400 is only one example of a computing system or device that may provide secure computing environment and/or a protected environment and is not intended to limit the examples described in this application to this particular computing environment or device type.

[00110] A suitable computing environment can be implemented with numerous other general purpose or special purpose systems. Examples of well known systems may include, but are not limited to, personal computers ("PC") 1400, hand-held or laptop devices, microprocessor-based systems, multiprocessor systems, set top boxes, programmable consumer electronics, gaming consoles, consumer electronic devices, cellular telephones, PDAs, and the like.

[00111] The PC 1400 includes a general-purpose computing system in the form of a computing device 1401 couple to various peripheral devices 1403, 1404, 1415, 1416 and the like. The components of computing device 1401 may include one or more processors (including CPUs, GPUs, microprocessors and the like) 1407, a system memory 1409, and a system bus 1408 that couples the various system components. Processor 1407 processes various computer executable instructions to control the operation of computing device 1401 and to communicate with other electronic and/or computing devices (not shown) via various communications connections such as a network connection 1414 an the like. The system bus 1408 represents any number of several types of bus structures, including a memory bus or memory controller, a peripheral bus, an accelerated graphics port, and/or a processor or local bus using any of a variety of bus architectures.

[00112] The system memory 1409 may include computer readable media in the form of volatile memory, such as random access memory (RAM), and/or non-volatile memory, such as read only memory (ROM). A basic input/output system (BIOS) may be stored in ROM. RAM typically contains data and/or program modules that are immediately accessible to and/or presently operated on by one or more of the processors 1407. By way of example, shown loaded in system memory for operation is a trusted application 202 utilizing a protected environment 203 and the media content being processed 106.

[00113] Mass storage devices 1404 and 1410 may be coupled to the computing device 1401 or incorporated into the computing device 1401 by coupling to the system bus. Such mass storage devices 1404 and 1410 may include a magnetic disk drive which reads from and writes to a removable, non volatile magnetic disk (e.g., a "floppy disk")

1405, and/or an optical disk drive that reads from and/or writes to a non-volatile optical disk such as a CD ROM, DVD ROM or the like 1406. Computer readable media 1405 and 1406 typically embody computer readable instructions, data structures, program modules and the like supplied on floppy disks, CDs, DVDs, portable memory sticks and the like.

[00114] Any number of program programs or modules may be stored on the hard disk 1410, other mass storage devices 1404, and system memory 1409 (typically limited by available space) including, by way of example, an operating system(s), one or more application programs, other program modules, and/or program data. Each of such operating system, application program, other program modules and program data (or some combination thereof) may include an embodiment of the systems and methods described herein. Kernel components 720-730 may be stored on the disk 1410 along with other operating system code. Media application 105 and/or a digital rights management system 204 may be stored on the disk 1410 along with other application programs. These components 720-730 and applications 105, 204 may be loaded into system memory 1409 and made operational.

[00115] A display device 1416 may be coupled to the system bus 1408 via an interface, such as a video adapter 1411. A user can interface with computing device 1400 via any number of different input devices 1403 such as a keyboard, pointing device, joystick, game pad, serial port, and/or the like. These and other input devices may be coupled to the processors 1407 via input/output interfaces 1412 that may be coupled to the system bus 1408, and may be coupled by other interface and bus structures, such as a parallel port(s), game port(s), and/or a universal serial bus (USB) and the like.

[00116] Computing device 1400 may operate in a networked environment using communications connections to one or more remote computers and/or devices through one or more local area networks (LANs), wide area networks (WANs), the Internet, radio links, optical links and the like. The computing device 1400 may be coupled to a network via a network adapter 1413 or alternatively via a modem, DSL, ISDN interface or the like.

[00117] Communications connection 1414 is an example of communications media. Communications media typically embody computer readable instructions, data structures, program modules and/or other data in a modulated data signal such as a carrier wave or other transport mechanism and includes any information delivery media. The term "modulated data signal" means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communications media include wired media such as a wired network or direct-wired connection, and wireless media such as acoustic, radio frequency, infrared, and other wireless media.

[00118] Those skilled in the art will realize that storage devices utilized to store computer-readable program instructions can be distributed across a network. For example a remote computer or device may store an example of the system described as software. A local or terminal computer or device may access the remote computer(s) or device(s) and download a part or all of the software to run a program(s). Alternatively the local computer may download pieces of the software as needed, or distributively process the software by executing some of the software instructions at the local terminal and some at remote computers and/or devices.

[00119] Those skilled in the art will also realize that by utilizing conventional techniques known to those skilled in the art that all, or a portion, of the software instructions may be carried out by a dedicated electronic circuit such as a digital signal processor ("DSP"), programmable logic array ("PLA"), discrete circuits, or the like. The term electronic apparatus as used herein includes computing devices and consumer electronic devices comprising any software and/or firmware and the like, and/or electronic devices or circuits comprising no software and/or firmware and the like.

[00120] The term computer readable medium may include system memory, hard disks, mass storage devices and their associated media, communications media, and the like.

CLAIM(S)

1. A method of establishing a protected environment within a computing device comprising:

validating a kernel component loaded into a kernel of the computing device;

establishing a security state for the kernel based on the validation;

creating a secure process and loading a software component into the secure process;

periodically checking the security state of the kernel; and

notifying the secure process when the security state of the kernel has changed.

2. The method of establishing a protected environment within a computing device of claim 1, in which validating a kernel component loaded into a kernel of the computing device further comprises setting a kernel secure flag.

3. The method of establishing a protected environment within a computing device of claim 2 in which periodically checking the security state of the kernel further comprises checking the kernel secure flag.

4. The method of establishing a protected environment within a computing device of claim 1, in which the kernel component includes a loader.

5. The method of establishing a protected environment within a computing device of claim 1, in which the software component is a trusted media component.

6. A method of loading kernel components to create a secure computing environment comprising:

setting a kernel secure flag to a true state:

checking to determine if a debugger is present in an operating system; and

checking to see if a debug credential associated with the debugger is present.

7. The method of loading kernel components to create a secure computing environment of claim 6, further comprising, setting the kernel secure flag to a false state if a debug credential is not present.

8. The method of loading kernel components to create a secure computing environment of claim 6, further comprising:

determining that there is another component to load;

validating a signature of the component;

verifying that a certificate is valid;

determining if the signature is on a revocation list; and

determining if the certificate in on the revocation list.

9. The method of loading kernel components to create a secure computing environment of claim 8, further comprising:

determining if the signature is acceptable for use;

determining if the is acceptable for use; and

loading the component into the kernel.

10. A method of creating a protected environment comprising:

creating a protected environment for loading a component;

checking the validity of a signature;

checking the validity of a certificate;

checking to see if the signature is in a revocation list;

checking to see if the certificate is in the revocation list;

checking to see that the signature is acceptable for use

checking to see that a certificate is acceptable for use;

setting a kernel secure flag; and

loading the component into the protected environment.

11. The method of creating a protected environment for loading a component of claim 10, in which the kernel secure flag is set to a false state if checking the validity of the signature fails.

12. The method of creating a protected environment for loading a component of claim 10, in which the kernel secure flag is set to a false state if checking the validity of a certificate fails.

13. The method of creating a protected environment for loading a component of claim 10, in which the kernel secure flag is set to a false state if checking to see if the certificate is in the revocation list fails.

14. The method of creating a protected environment for loading a component of claim 10, in which the kernel secure flag is set to a false state if checking to see that the signature is acceptable for use fails.

15. The method of creating a protected environment for loading a component of claim 10, in which the kernel secure flag is set to a false state if checking to see that the certificate is acceptable for use fails.

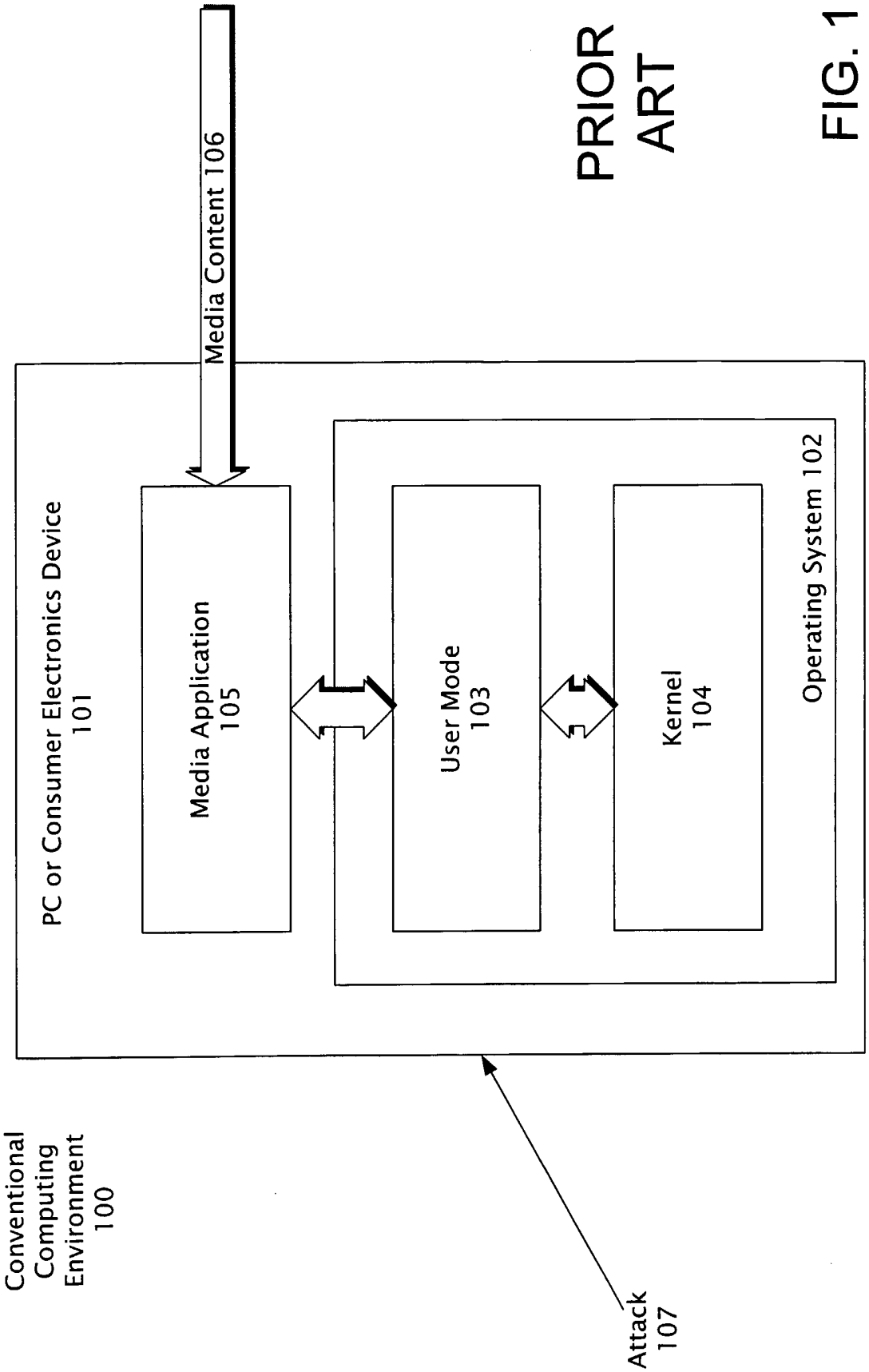
16. The method of creating a protected environment for loading a component of claim 10, in which the component is a portion of a media application.

17. The method of creating a protected environment for loading a component of claim 10, further comprising periodically checking a security state of the protected environment.

18. The method of creating a protected environment for loading a component of claim 17, in which periodically checking the security state of the protected environment further comprises checking the kernel secure flag.

19. The method of creating a protected environment for loading a component of claim 18, in which periodically checking the security state of the protected environment further comprises submitting a call of a calling process to a kernel.

20. The method of creating a protected environment for loading a component of claim 19, in which the calling process to the kernel is checked to determine if it is secure.



PRIOR ART

FIG. 1

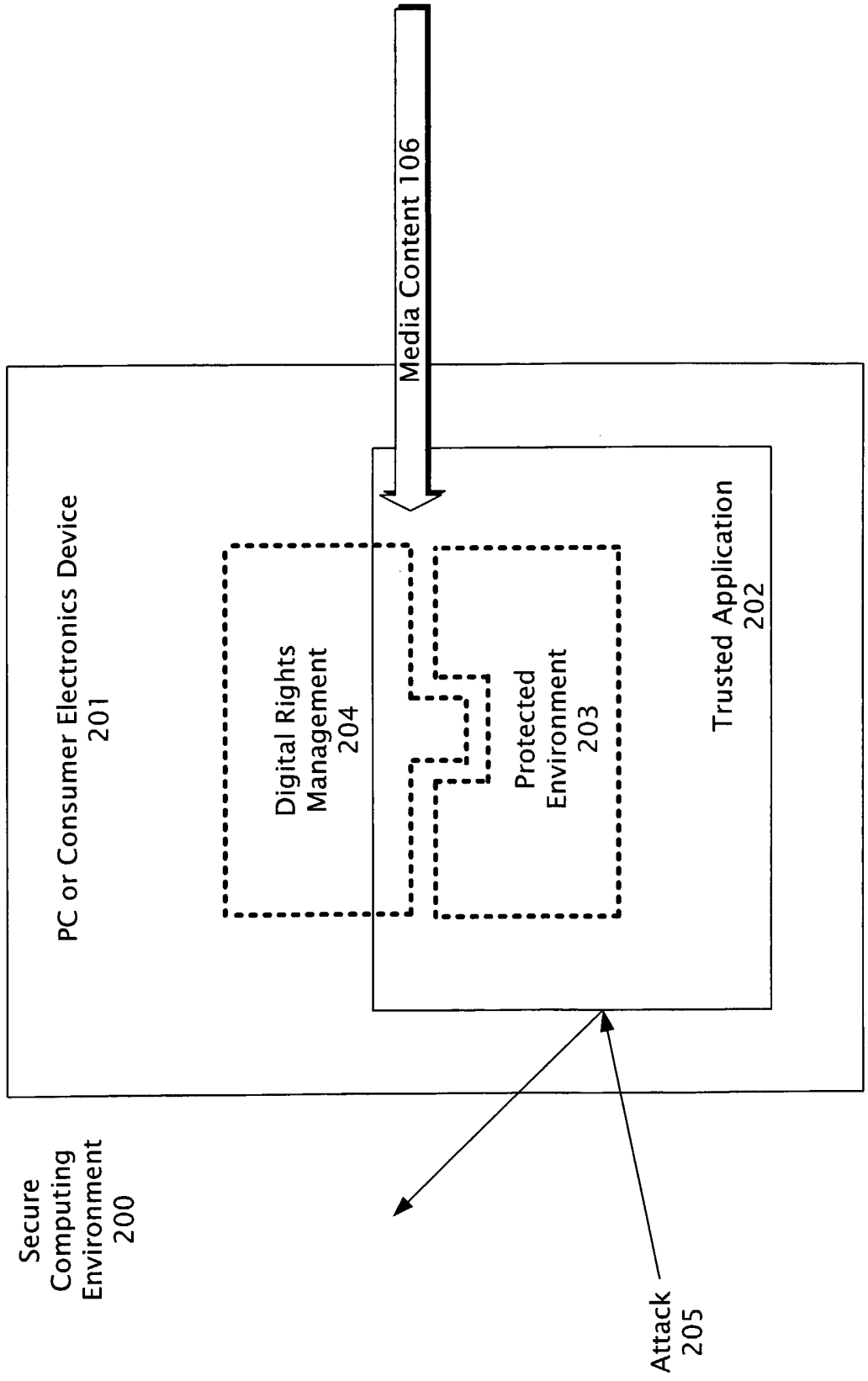

Media Content 106
PC or Consumer Electronics Device
201
Digital Rights
Management
204
Protected
Environment
203
Trusted Application
202
Secure
Computing
Environment
200
Attack
205

FIG. 2

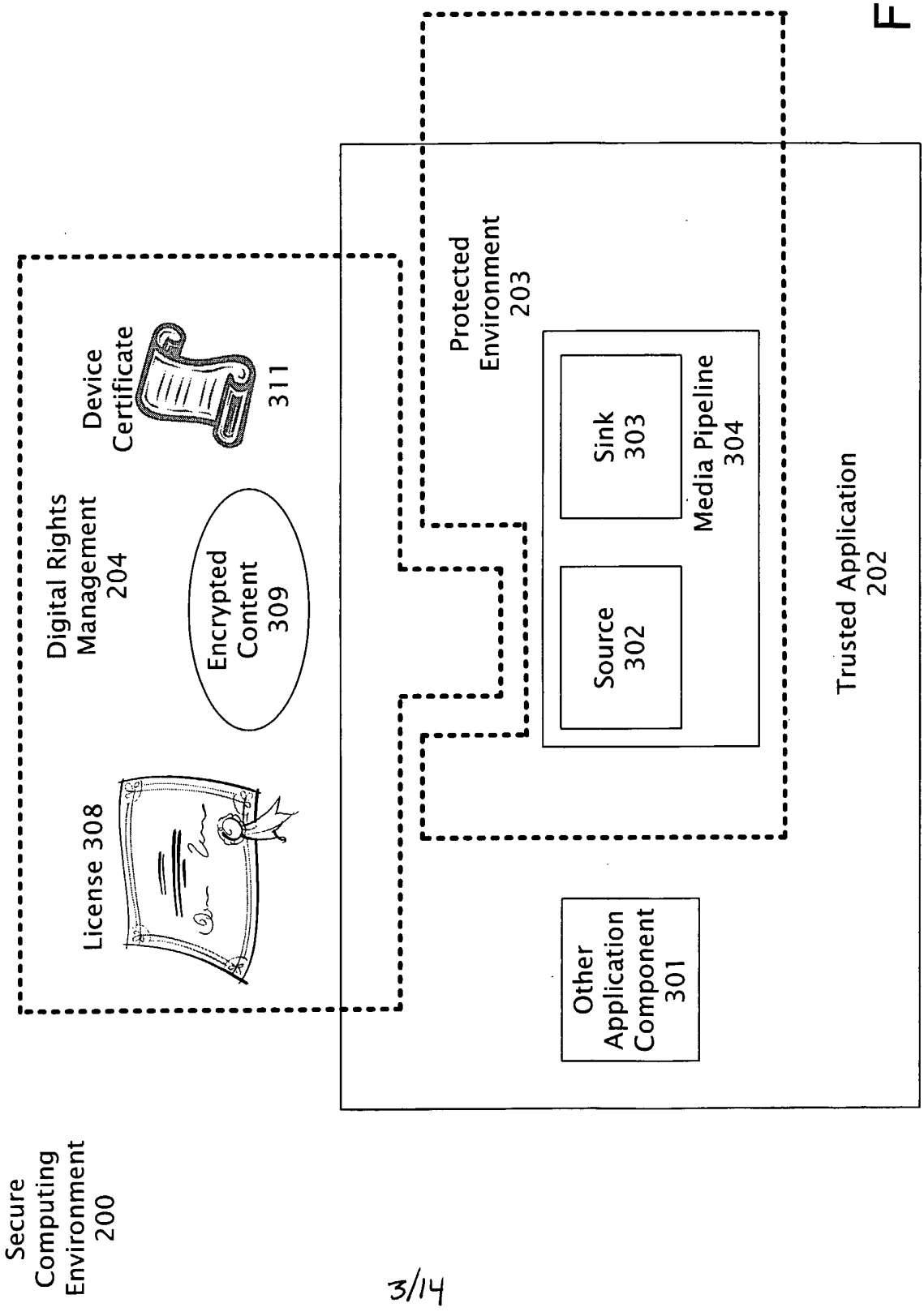

Secure Computing Environment 200
Digital Rights Management 204
License 308
Device Certificate 311
Encrypted Content 309
Trusted Application 202
Protected Environment 203
Other Application Component 301
Source 302
Sink 303
Media Pipeline 304

FIG. 3

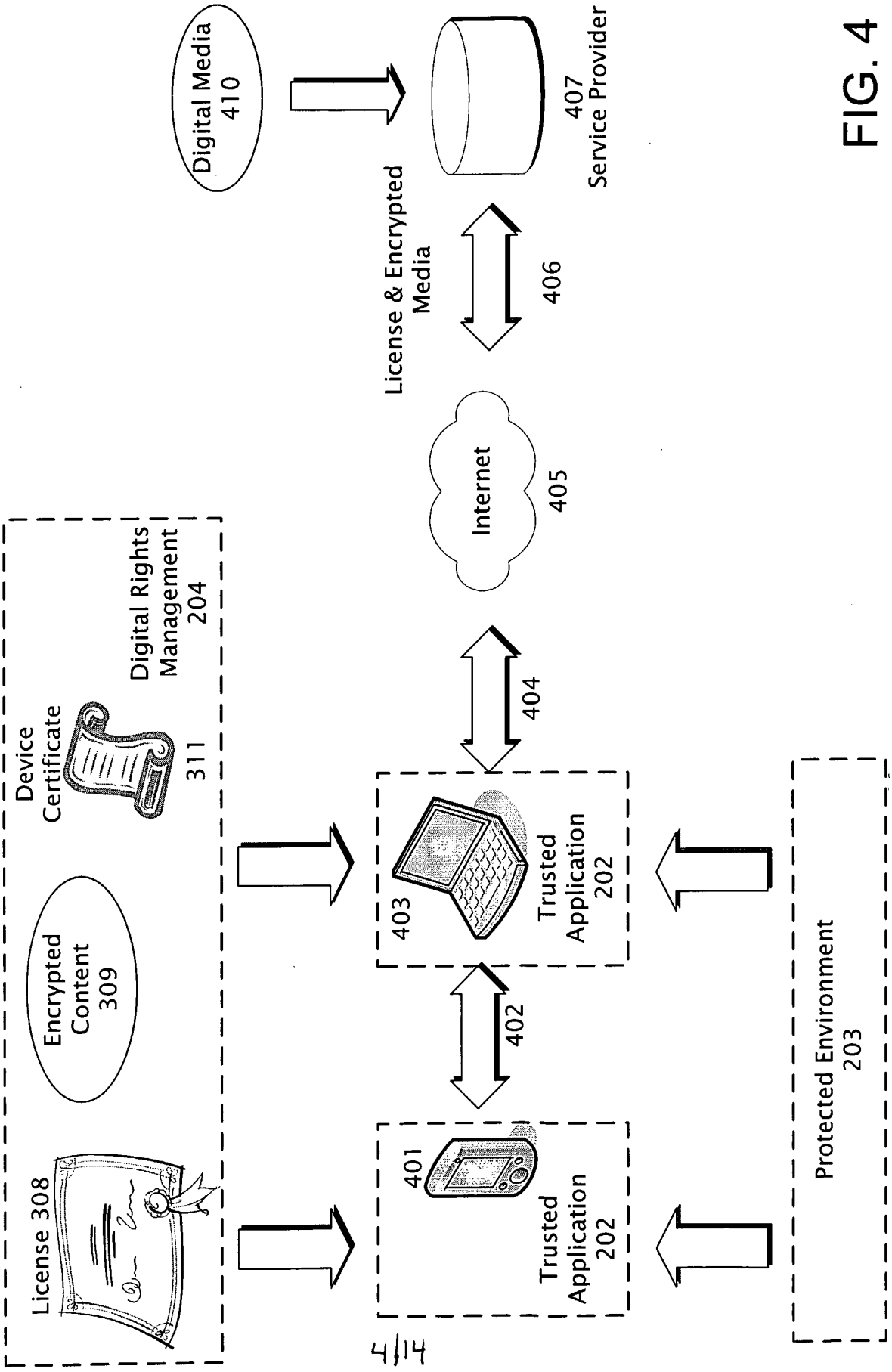

Digital Media
410
407
Service Provider
License & Encrypted
Media
406
Internet
405
404
Digital Rights
Management
204
Device
Certificate
311
Encrypted
Content
309
License 308
403
Trusted
Application
202
402
401
Trusted
Application
202
Protected Environment
203

FIG. 4

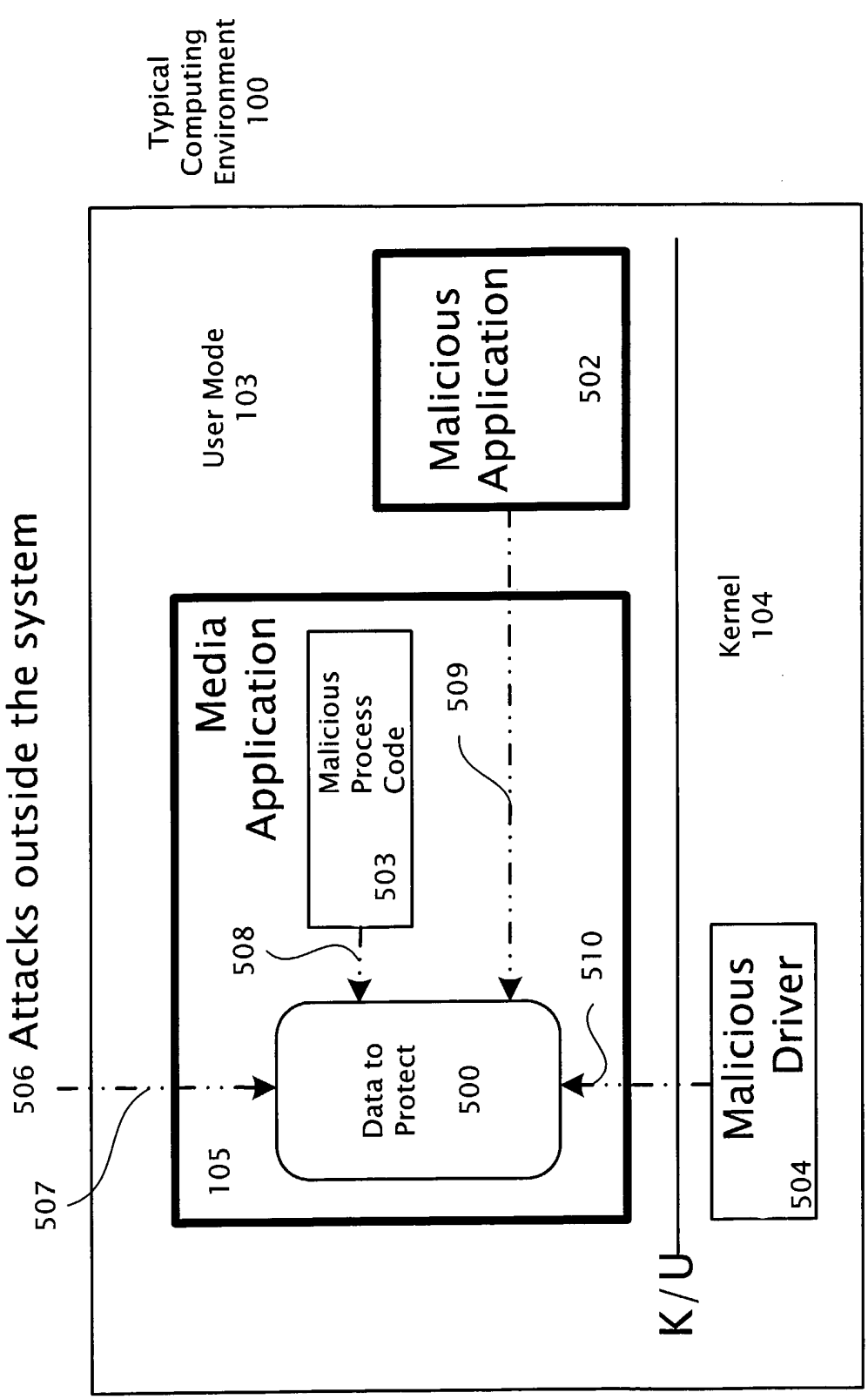

Typical Computing Environment 100
User Mode 103
Malicious Application 502
Media Application
Malicious Process Code 503
508
509
510
Data to Protect 500
105
506 Attacks outside the system
507
Kernel 104
Malicious Driver 504
K/U

FIG. 5

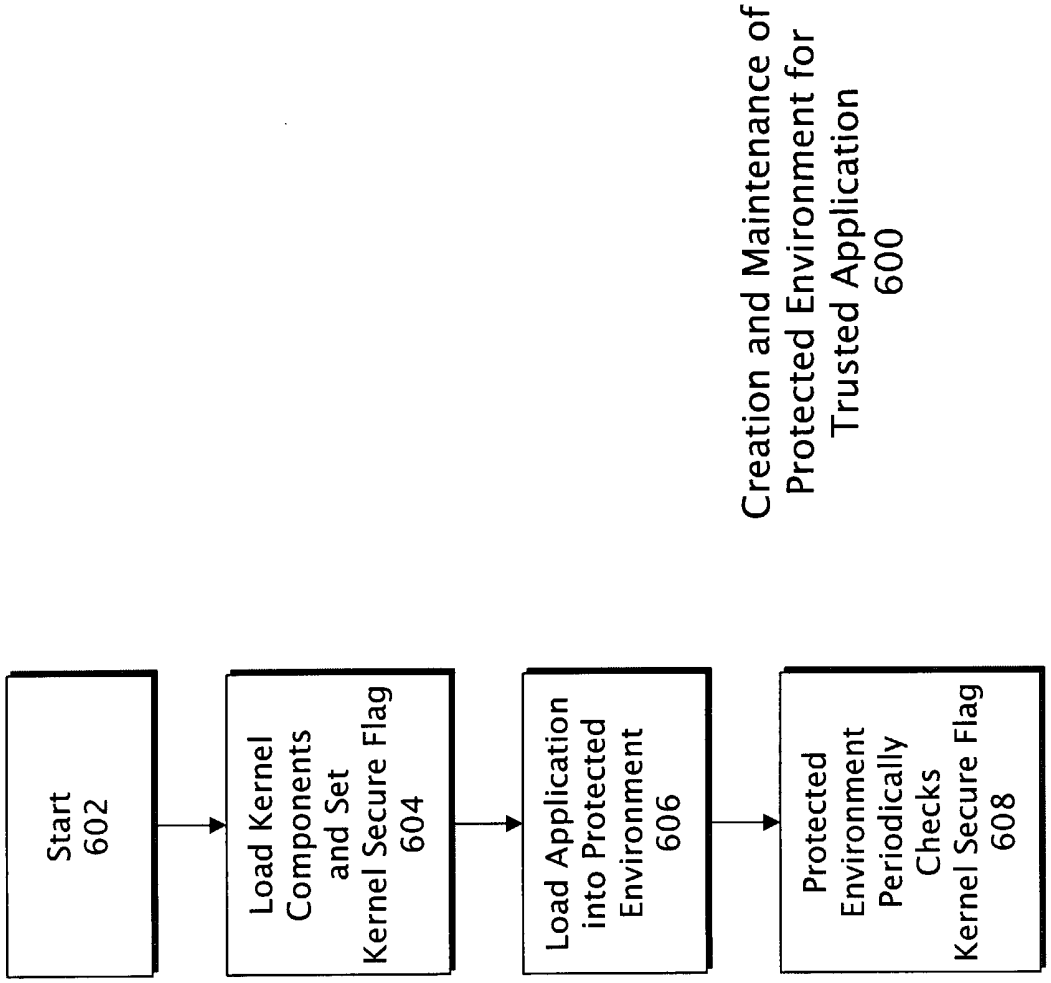

Creation and Maintenance of Protected Environment for Trusted Application 600
Start 602
Load Kernel Components and Set Kernel Secure Flag 604
Load Application into Protected Environment 606
Protected Environment Periodically Checks Kernel Secure Flag 608

FIG. 6

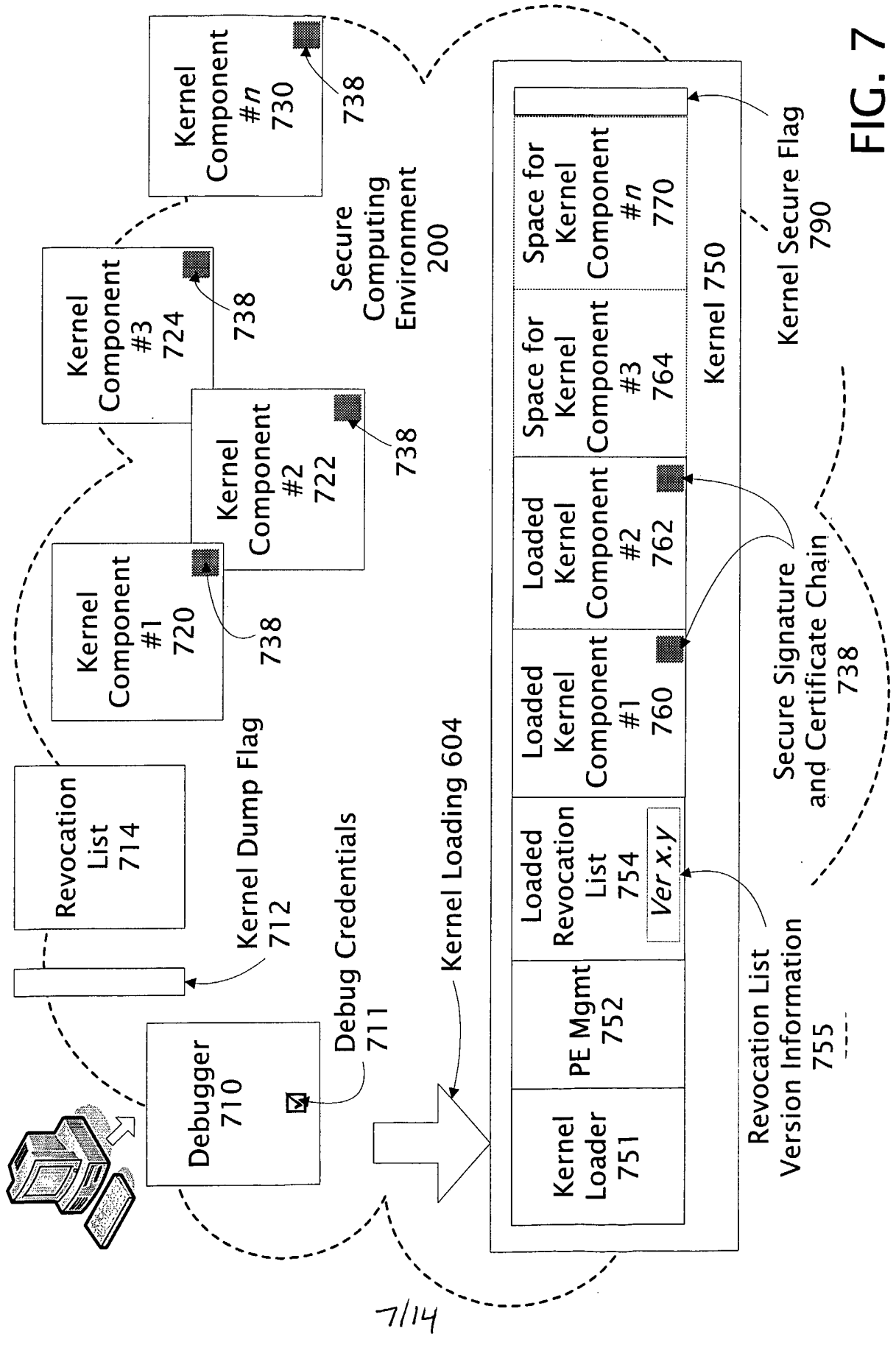

Kernel Component #n 730
738
Secure Computing Environment 200
Kernel Component #3 724
738
Kernel Component #2 722
738
Kernel Component #1 720
738
Revocation List 714
Kernel Dump Flag 712
Debug Credentials 711
Debugger 710
Kernel Loading 604
Kernel Loader 751
PE Mgmt 752
Loaded Revocation List 754
Ver x.y
Loaded Kernel Component #1 760
Loaded Kernel Component #2 762
Space for Kernel Component #3 764
Space for Kernel Component #n 770
Kernel 750
Kernel Secure Flag 790
Secure Signature and Certificate Chain 738
Revocation List Version Information 755

FIG. 7

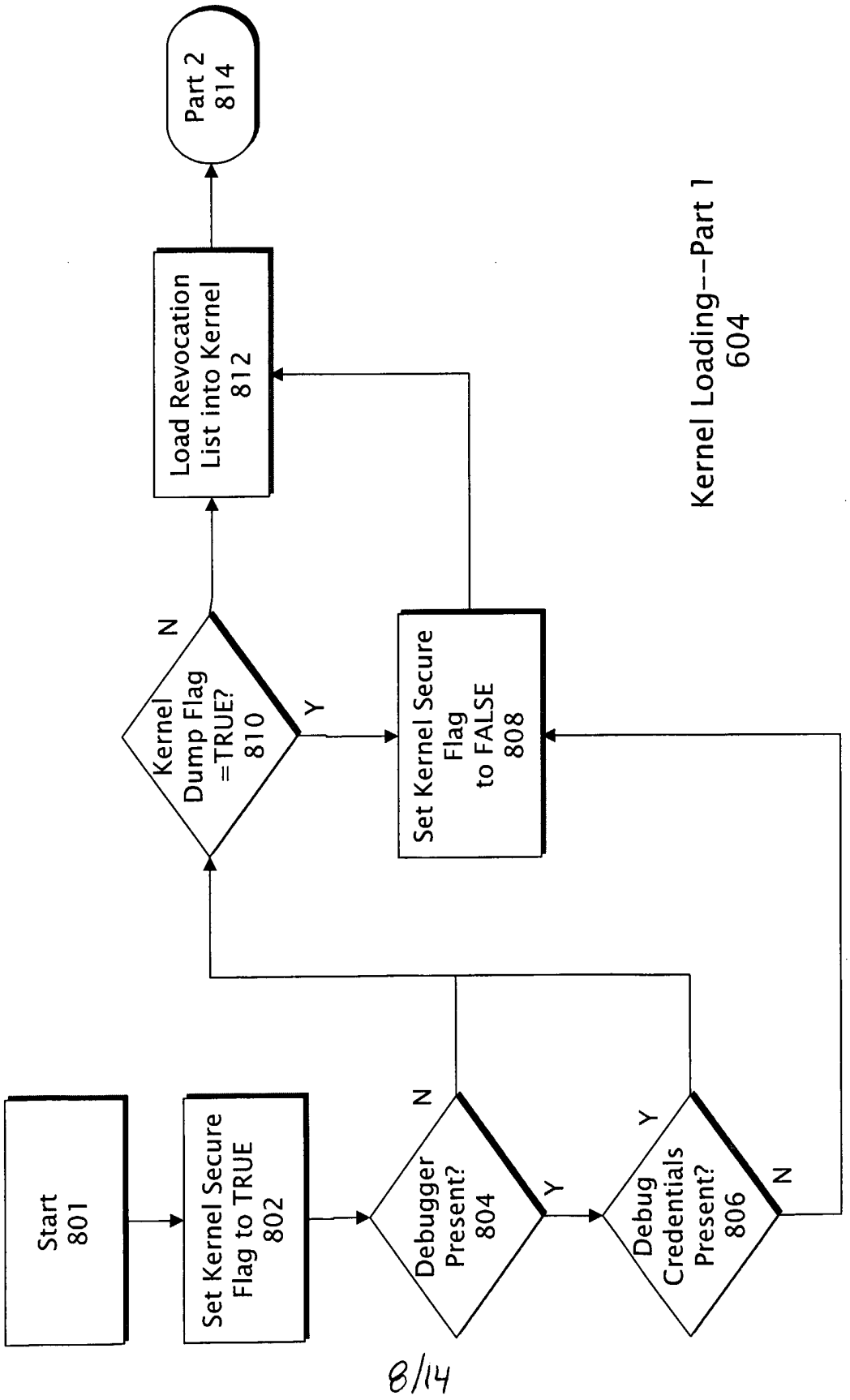

Start
801
Set Kernel Secure Flag to TRUE
802
Debugger Present?
804
N
Y
Debug Credentials Present?
806
Y
N
Kernel Dump Flag =TRUE?
810
N
Y
Set Kernel Secure Flag to FALSE
808
Load Revocation List into Kernel
812
Part 2
814
Kernel Loading--Part 1
604

FIG. 8

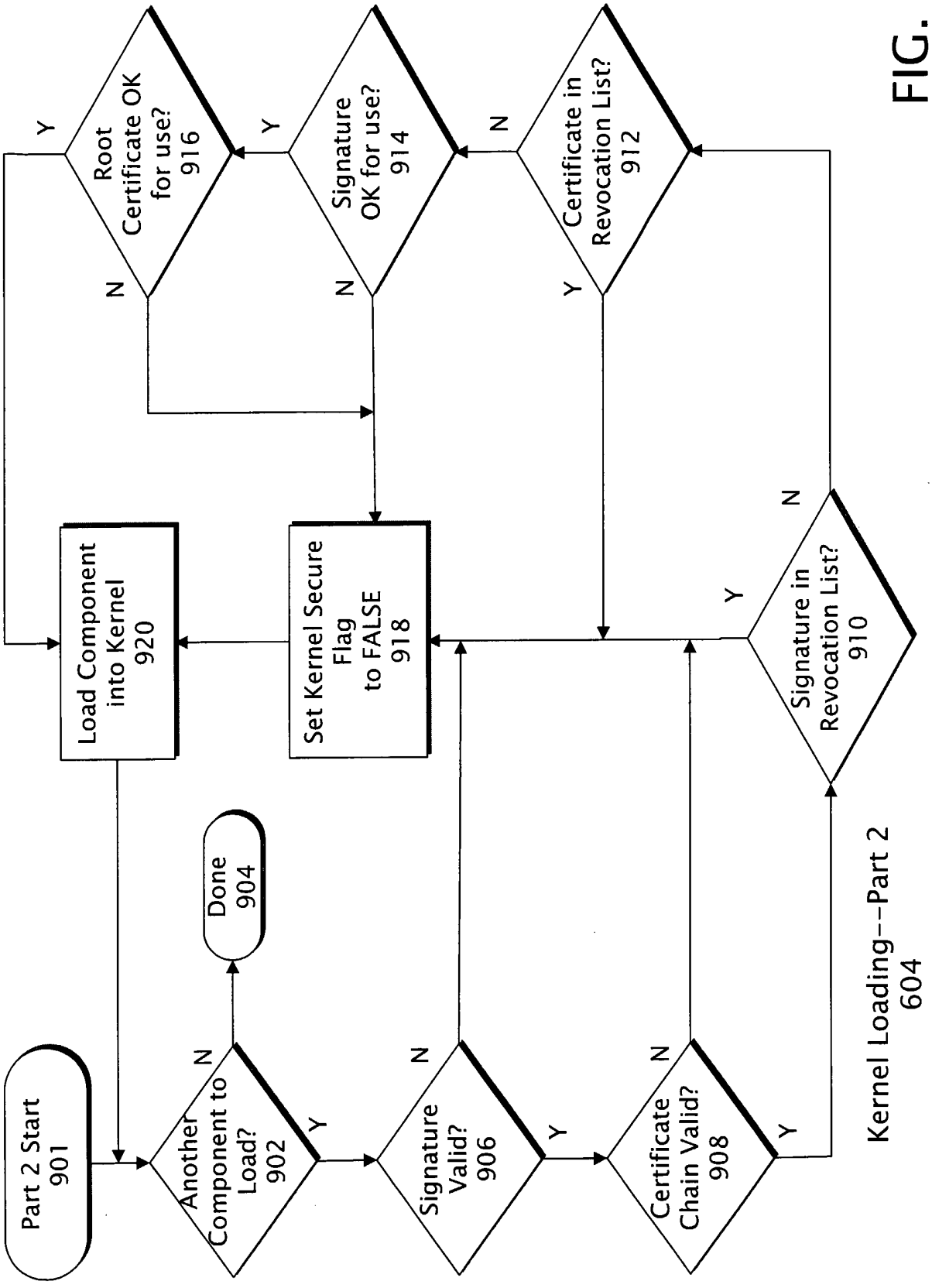

Part 2 Start 901
Another Component to Load? 902
Y
N
Done 904
Signature Valid? 906
Y
N
Certificate Chain Valid? 908
Y
N
Signature in Revocation List? 910
Y
N
Certificate in Revocation List? 912
Y
N
Signature OK for use? 914
Y
N
Root Certificate OK for use? 916
Y
N
Set Kernel Secure Flag to FALSE 918
Load Component into Kernel 920
Kernel Loading--Part 2 604

FIG. 9

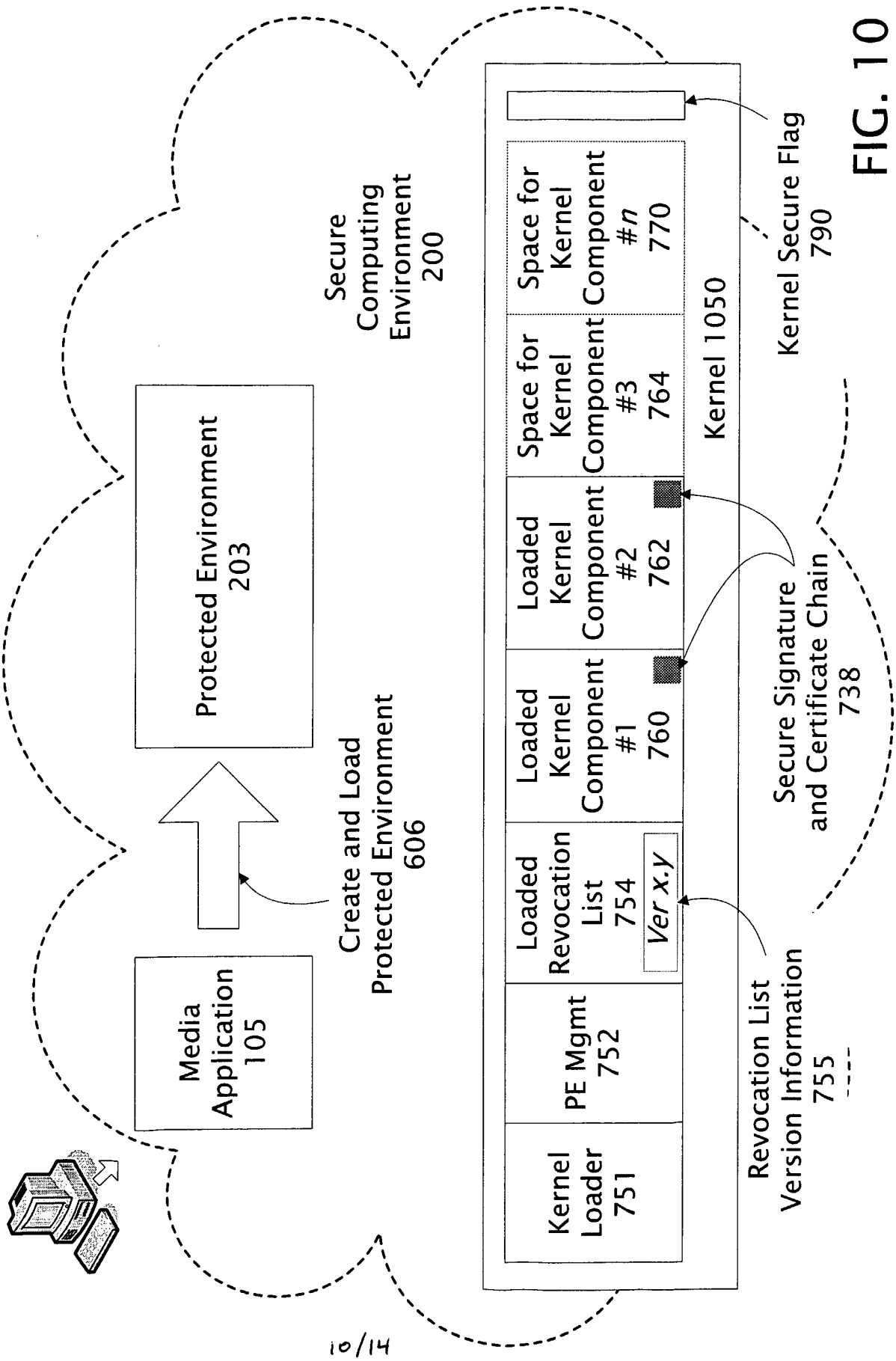

Secure Computing Environment 200
Protected Environment 203
Media Application 105
Create and Load Protected Environment 606
Kernel Loader 751
PE Mgmt 752
Loaded Revocation List 754
Ver x.y
Loaded Kernel Component #1 760
Loaded Kernel Component #2 762
Space for Kernel Component #3 764
Space for Kernel Component #n 770
Kernel 1050
Kernel Secure Flag 790
Secure Signature and Certificate Chain 738
Revocation List Version Information 755

FIG. 10

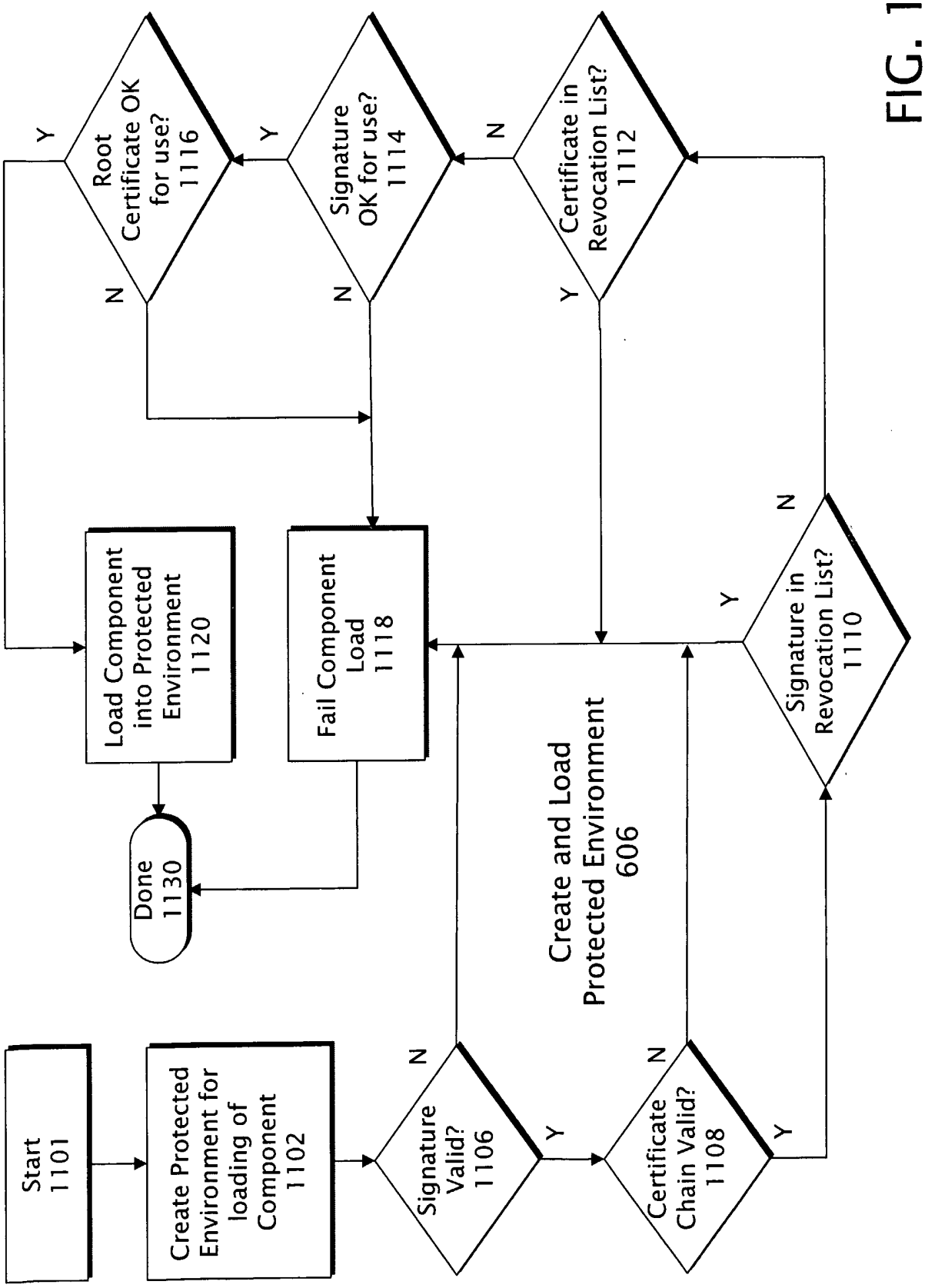

Start
1101
Create Protected Environment for loading of Component
1102
Signature Valid?
1106
Certificate Chain Valid?
1108
Signature in Revocation List?
1110
Certificate in Revocation List?
1112
Signature OK for use?
1114
Root Certificate OK for use?
1116
Fail Component Load
1118
Load Component into Protected Environment
1120
Done
1130
Create and Load Protected Environment
606
Y
N

FIG. 11

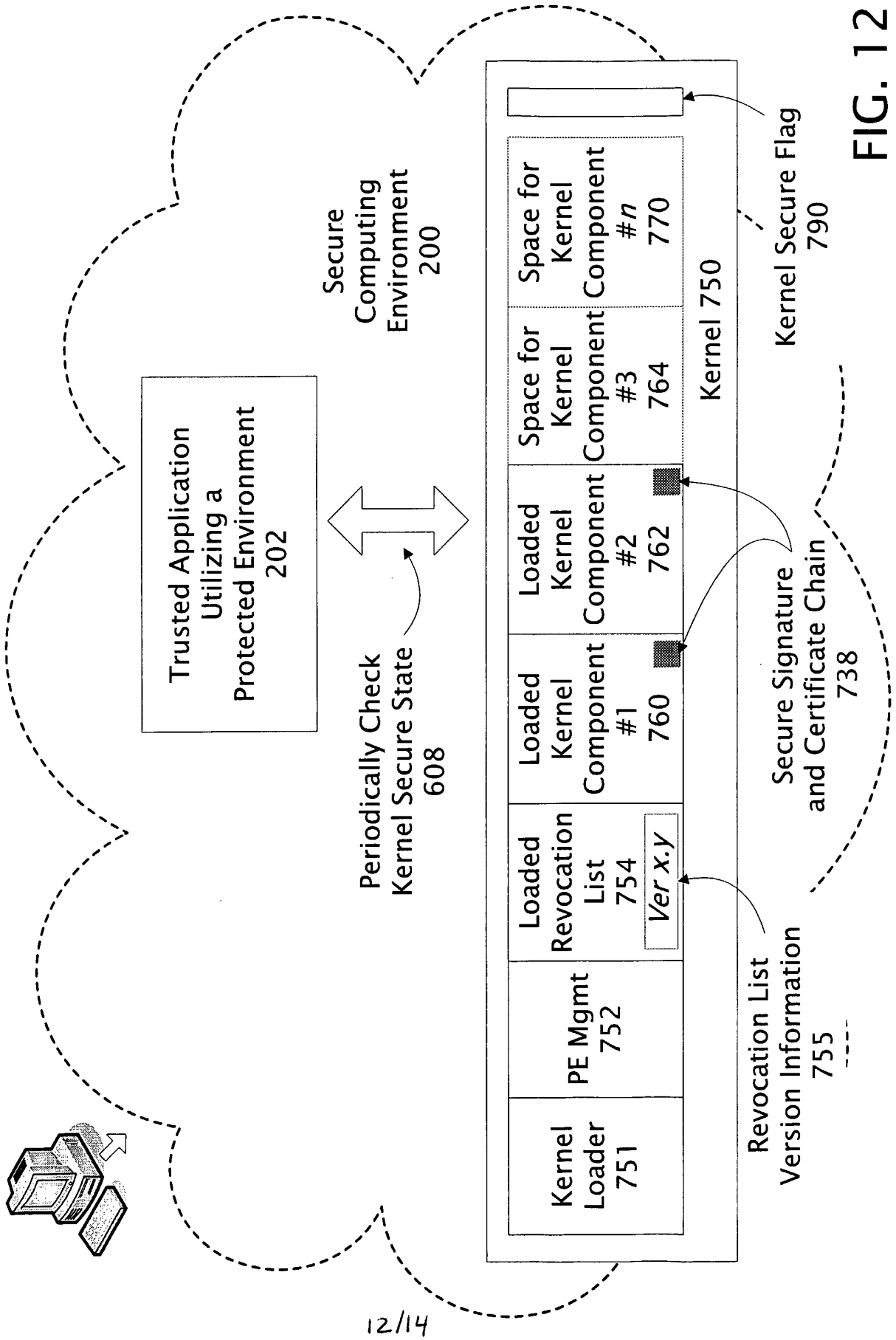

Secure Computing Environment 200
Trusted Application Utilizing a Protected Environment 202
Periodically Check Kernel Secure State 608
Kernel 750
Kernel Loader 751
PE Mgmt 752
Loaded Revocation List 754
Ver x.y
Loaded Kernel Component #1 760
Loaded Kernel Component #2 762
Space for Kernel Component #3 764
Space for Kernel Component #n 770
Kernel Secure Flag 790
Secure Signature and Certificate Chain 738
Revocation List Version Information 755

FIG. 12

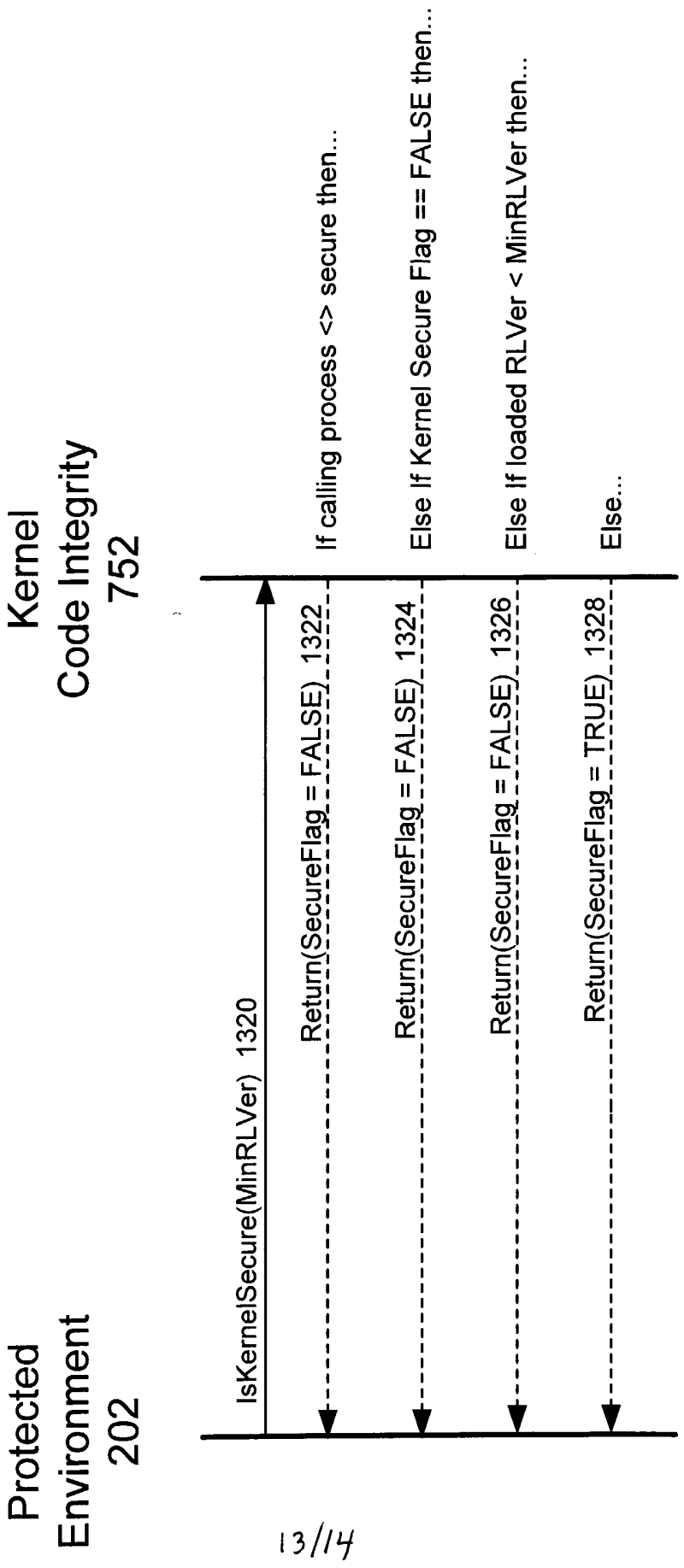

Protected Environment 202
Kernel Code Integrity 752
IsKernelSecure(MinRLVer) 1320
Return(SecureFlag = FALSE) 1322
Return(SecureFlag = FALSE) 1324
Return(SecureFlag = FALSE) 1326
Return(SecureFlag = TRUE) 1328
If calling process <> secure then...
Else If Kernel Secure Flag == FALSE then...
Else If loaded RLVer < MinRLVer then...
Else...
Check Kernel Secure State 608

FIG. 13

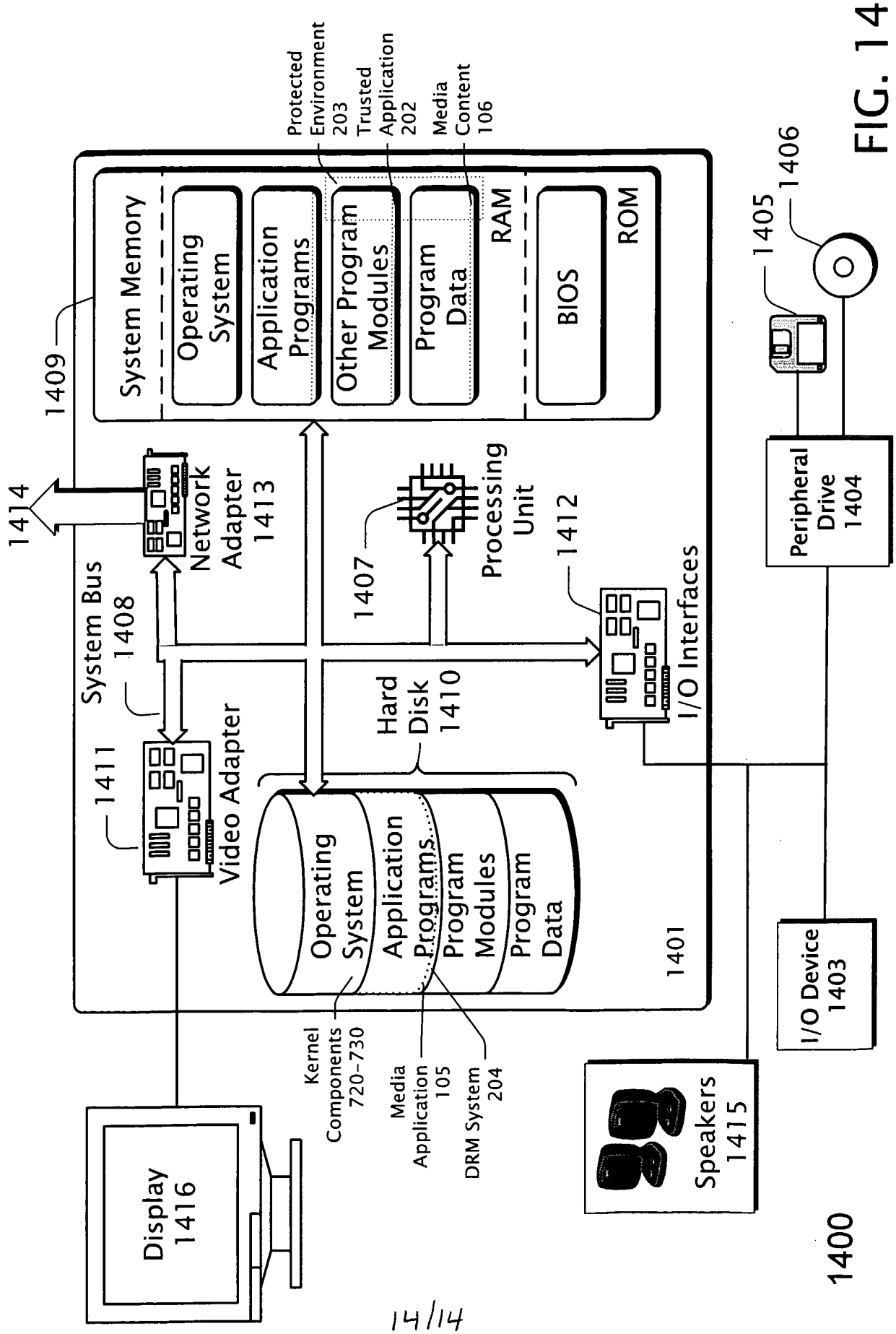

Protected Environment 203
Trusted Application 202
Media Content 106
System Memory
Operating System
Application Programs
Other Program Modules
Program Data
RAM
BIOS
ROM
1409
1414
Network Adapter 1413
System Bus 1408
1407
Processing Unit 1412
1405
1406
Peripheral Drive 1404
I/O Interfaces
Hard Disk 1410
1411
Video Adapter
Operating System
Application Programs
Program Modules
Program Data
1401
Kernel Components 720-730
Media Application 105
DRM System 204
I/O Device 1403
Speakers 1415
Display 1416
1400

FIG. 14