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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
14 March 2002 (14.03.2002)

PCT

(10) International Publication Number
WO 02/21243 A2

(51) International Patent Classification[7]: G06F 1/00

(21) International Application Number: PCT/GB01/03962

(22) International Filing Date:
5 September 2001 (05.09.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/658,218 8 September 2000 (08.09.2000) US

(71) Applicant: **INTERNATIONAL BUSINESS MACHINES CORPORATION** [US/US]; New Orchard Road, Armonk, NY 10504 (US).

(71) Applicant *(for MG only)*: **IBM UNITED KINGDOM LIMITED** [GB/GB]; PO Box 41, North Harbour, Portsmouth, Hampshire PO6 3AU (GB).

(72) Inventors: **LOTSPIECH, Jeffrey**; 992 Foothill Drive, San Jose, CA 95123 (US). **NUSSER, Stefan**; 4384 Cedar Creek Road, Boca Raton, FL 33487 (US).

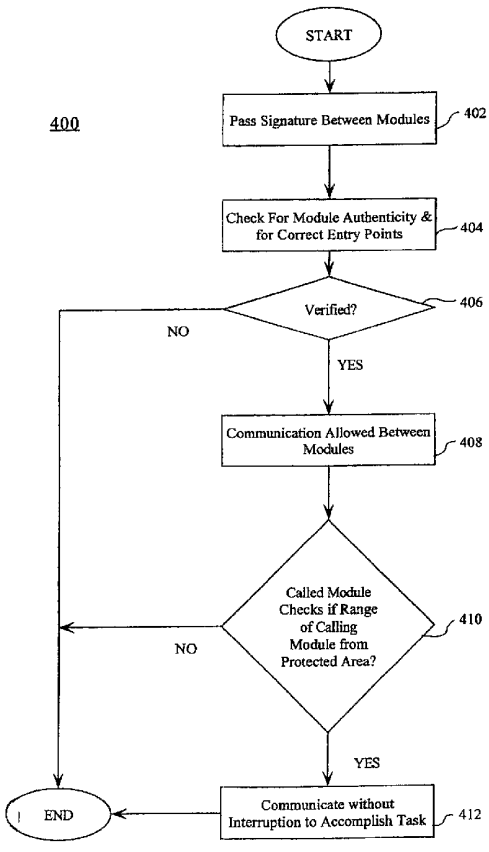
(74) Agent: **DAVIES, Simon, Robert**; IBM United Kingdom Limited, Intellectual Property Law, Hursley Park, Winchester, Hampshire SO21 2JN (GB).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European

[Continued on next page]

(54) Title: SOFTWARE SECURE AUTHENTICATED CHANNEL



(57) Abstract: Software manufacturers examine their module and determine a range of addresses in memory which the module occupies. A protected range of addresses in memory is predefined to not allow changes, such as patching by hackers. Each manufacturer delivers the range of addresses describing the protected area and a known good version of their module to other manufacturers that they want to interoperate with. The other manufacturers return digital signatures on the protected area, and these digital signatures are stored in the first manufacturer's module. Correspondingly, the other manufacturers do the same with their own modules. Then, in order to effect a secure communication channel between two modules the modules first pass each other the signatures previously produced. Then, to ensure that communication is being effected with an authentic authorized module, through the use of the signature and the address ranges in the protected area, each module checks that the other module has not been patched. They each further verify that all the entry points in the other module they intend to call are in fact within the protected area. In the event that both modules are verified as being trustworthy, the modules now call each other freely. However, each module, when it is called must verify that it was called from within the protected area of the other module.

WO 02/21243 A2

patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

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

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

SOFTWARE SECURE AUTHENTICATED CHANNEL

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention relates to software tamper resistance. More particularly, the invention provides a method and apparatus for establishing a Secure Authenticated Channel between two software modules on a system.

2. The Prior Art

For purposes of this disclosure a manufacturer of a device containing an information processing unit is known as an entity. Certain types of software applications must protect themselves against being hacked. For example, DVD video players are required by license to include defenses to prevent the users from copying DVD movies by use of a scrambling scheme. More recently, the Secure Digital Music Initiative (SDMI), started by the Recording Industry Association of America, have released requirements which can only be met by tamper resistant software.

The SDMI requirements envisions that a particular application will consist of different software modules from different manufacturers. For example, one manufacturer might produce an electronic music distribution system that allows the secure downloading of music over the Internet, like the IBM Electronic Media Management System TM. A second manufacturer might produce software supporting a secure flash memory music player, like the Sony Memory Stick Walkman TM. The consumer would like the music he has purchased and downloaded to be placed on his flash memory music player.

So, the SDMI requirement describes the idea of a *Secure Authenticated Channel*, or SAC, between any two compliant software modules. There, a model is proposed that describes communication between two modules, created by two different manufacturers, on the same machine which want to securely call each other's entry points. FIG. 1 illustrates a Secure Authenticated Channel (100) which comprises at least two modules (102, 104) loaded into memory on the same machine (106). These modules are created by different vendors and communicate which each other using their public APIs. There is no protection against a hacking program, which might interpose itself and steal music content for illegal

reproduction. For that reason, SDMI requires a mechanism of authentication and a mechanism to secure the communication between these two modules - the Secure Authenticated Channel.

FIG. 2 illustrates the security contents (200) of a prior art software architecture that have been designed in response to SDMI requirements. The software module contains a public key certificate (202), a public key (204) and a private key (206). For example, the SAC is implemented as follows:

1. Each module is responsible for its own integrity.

2. Each module has a standard public-key certificate (202) that certifies its public key (204).

3. Each module has a private key (206) which it must keep secret.

4. A standard public-key protocol is used to establish a session key between the modules.

5. Subsequent data and control passed between the modules is encrypted with a session key (not shown in FIG. 2) to prevent illegal tampering with the data, or eavesdropping.

This approach is essentially the well-known protocol for secure communication between any two parties. Unfortunately, there are several problems using this approach for the SAC. In a secure communications protocol, the attacker is presumed to be outside of the two communicating parties' communication pipeline, which is not true in software. A hacker can view both modules, and patch either or both of them. Thus, keeping the private key a secret is a real problem. Likewise, self-Integrity checking, where each module ascertains that it has not been patched or tampered with, is also very difficult. As such, this protocol is as weak as the weakest link, meaning that if one of the private keys becomes known, or if one of the modules is successfully patched, the SAC has been defeated.

SUMMARY OF THE INVENTION

Accordingly, the invention provides a method for peer to peer authentication of a software module on a single machine, the method comprising the steps of: determining a protected range of addresses that a

software module occupies; verifying the integrity of code in said protected range; and verifying that all function calls performed by that module originate from and return to that range.

In accordance with a preferred embodiment a more efficient and effective method, apparatus, and product for securing and verifying the authenticity and security of the connection between two software modules is provided.

An improved system and method for controlling distribution of software is preferably provided that enables the modules to distinguish between authorized and unauthorized modules.

The invention preferably provides a new Secure Authenticated Channel or SAC. In accordance with a preferred embodiment, the strength of this protocol is that each module, rather than being responsible for its integrity, is responsible for the integrity of other modules. The fundamental weakness of a software communication protocol is that both implementations of the SAC are visible to each other and to an attacker. Therefore, the invention preferably converts what would be a weakness into a strength.

The invention is implemented in accordance with a preferred embodiment firstly by each software manufacturer examining their module and determining the range of addresses in memory which the module occupies. A decision is made about which ranges of addresses in memory should not be allowed to be patched by hackers. Such a range is called a protected area. Typically, the protected area contains all the program code of the module, and some of the data necessary to accomplish the functions of the program code.

Modern programming techniques often makes software modules in location independent code. The address of a module is finally resolved during loading at the final product memory map. Therefore, because of location independent code, the absolute addresses and lengths of the protected areas at run time are preferably determined. For this reason, the specification of the base address and length using the start of the code segment as a relative base is preferably allowed. Also, if no address range is provided, the entire code segment is preferably defaulted to.

In accordance with a preferred embodiment, next each manufacturer delivers the range of addresses describing the protected area and a known good version of their module to every other manufacturer they want to interoperate with. The other manufacturers preferably return digital signatures on the protected area, and these digital signatures are stored in the first manufacturer's module. The signature is preferably computed on the machine code instructions in the protected area. This all preferably occurs at manufacturing time and before the modules have been deployed in the field.

Correspondingly, the other manufacturers preferably do the same with their own modules. In order to effect a secure communication channel between two modules, the modules, according to one preferred embodiment of the present invention, first pass each other the signatures previously produced. This can happen when the modules are actually run in the field, because the signatures are not secret. In an alternative preferred embodiment of the present invention, each module contains copies of the signatures of the other modules that it expects to communicate with during actual run in the field.

Then, in accordance with a preferred embodiment, to ensure that communication is being effected with an authentic authorized module, through the use of the signature and the address ranges in the protected area, each module checks that the other module has not been patched by verifying the digital signature, such as with respect to the address range of the other module.

Preferably in the event that both modules are verified as being trustworthy, the modules now call each other freely. However, each module, when it is called preferably verifies that it was called from within the protected area of the other module. Also, before calling an entry point in the other module, in a preferred embodiment of the present invention, a module makes sure that the function it intends to call is in fact within the protected area of the target module. These checks are intended to prevent an attacker from loading the two modules into memory, waiting for the signature check to complete and than calling the entry points in either module from within a third module.

According to another aspect, the invention provides a computer readable medium comprising instructions for peer to peer authentication of a software module on a single machine, the computer readable medium comprising instructions for: determining a protected range of addresses

that a software module occupies; verifying the integrity of code in said protected range; and verifying that all function calls performed by that module originate from and return to that range.

According to yet another aspect, the invention provides a system for peer to peer authentication of a software module on a single machine, the system comprising: a determination unit for determining a protected range of addresses that a software module occupies; a verification unit for verifying the integrity of code in said protected range; and a second verification unit for verifying that all function calls performed by that module originate from and return to that range.

BRIEF DESCRIPTION OF THE FIGURES

A preferred embodiment of the present invention will now be described, by way of example only, and with reference to the following drawings:

FIG. 1 illustrates a Secure Authenticated Channel (100) implemented on a single machine as found in the prior art.

FIG. 2 illustrates the security contents (200) of a software module as described in the prior art SDMI specification.

FIG. 3 is a flow diagram (300) for generation of module signatures in accordance with a preferred embodiment of the present invention.

FIG. 4 is a flow diagram (400) showing the improved software module peer to peer integrity checking in accordance with a preferred embodiment of the present invention.

FIG. 5 illustrates the memory space (500) of a computer system including the space occupied by a software module including a protected area.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Exemplary Embodiment-Software Secure Authenticated Channel

The embodiment comprises a method of verifying the integrity of a computer program code module within a computer to detect tampering or substitution of the module, and thus to prevent hacking of the program

module. In particular, each program module is responsible for the verification of the integrity of the other program code modules. In other words, independent peer modules check each other without a master-slave relationship.

The embodiment of the present invention provides a new Secure Authenticated Channel or SAC. The strength of this protocol is that each module, rather than being responsible for its integrity, is responsible for the integrity of other modules. A traditionally fundamental weakness of a software communication protocol is now turned into a strength in that both sides of the SAC are visible to each other and to other parties, including even an attacker.

FIG. 3 comprises a flow diagram (300) illustrating a preferred embodiment of the present invention for generation of module signatures. Firstly, each software manufacturer either manually or automatically examines their module and determines the range of addresses (302) in memory which the module occupies. A determination is made about which ranges of addresses in memory should be kept intact (304) because of the potential for damaging attacks by hackers. Alternatively, a determination is made about which ranges of addresses in memory comprises the main functionality of the module. This range includes all outgoing calls to other modules which make use of the SAC. Such a range is called a protected area. Typically, the protected area contains all the program code of the module, and some of the data necessary to accomplish the functions of the program code.

Next, each manufacturer delivers the range of addresses describing the protected area and a known good version of their module (306) to every other manufacturer they want to interoperate with. The other manufacturers return digital signatures (308) on the protected area. The signatures are preferably generated by hashing the entire protected area and using that hash for a digital signature. Then these digital signatures are stored in the first manufacturer's module (310) as supplied by the other manufacturers of the other modules and the amended software module is forwarded back to the manufacturer of the software module. Also, the range of addresses is stored in the first manufacturer's module. This is so that it is possible to correctly verify the signature, and determine the entry points of the calling and called routines. In other words, the SAC preferably directs a module to reveal its exported entry points and its signed area. All of the preceding steps preferably occur at manufacturing time and before the modules have been deployed in the

field. Correspondingly, the other manufacturers do the same with their own other modules.

FIG. 4 illustrates the invention according to a preferred embodiment in a flow diagram (400) showing improved software module peer to peer integrity checking. Next, a description of the basic operation of the integrity checking will be described. In order to effect a secure communication channel between two modules they first pass each other the signatures (402) previously produced. Then, to ensure that communication is being effected with an authentic authorized module, through the use of the signature and the address ranges in the protected area, each module checks (404) that the other module has not been patched. The signature is based on the hash of the code in the specified address ranges. The signature, the address ranges and the available APIs are permanently stored in each module and are retrieved for signature verification. During verification, the hash on the specified address range is recalculated. This will determine whether the code in the specified address range is authentic or not. If the code is patched, the hash of the provided address ranges will be different and the signature verification will fail. Also, the origin of a function call can easily be determined by looking at the return address of the function on a stack. The modules each further verify that all the entry points in the other module they intend to call are in fact within the protected area of the other module. In order to accomplish this, every module retrieves the other module's protected area - for example by having that information built in or by dynamically querying the other module. If they are not verified as being authentic modules then the process terminates. Otherwise, in the event that both modules are verified (406) as being trustworthy, the modules now can call each other freely (408). However, each module, when it is called preferably verifies that it was called from within the protected area of the other module (410). If the calling module is not calling from its protected area then the process terminates. Otherwise, if called from the protected area of the other module, then communication continues (412) between the modules to accomplish a task. Therefore, any attempt to attack the software modules is defeated. A called module determines that a calling module is calling from the protected area address space by the following process. Every function call depends on a return address being left on a stack. This is how a function returns to a caller. As a consequence, that return address is used to make sure that the caller is within the specified protected area. In other words, module B can make sure that it is called by module A only

from within the address range that module B verified during the initial signature verification process. This is shown in FIG. 5.

FIG. 5 illustrates the memory space (500) of a computer system including the space occupied by a software module including a protected area in accordance with a preferred embodiment of the present invention. The software module comprises a code space defined by (504) and (506) for example. The module contains a protected area (506) that defines an address range that must be protected against being patched by hackers. A function call (508) is shown with a return address being left on a stack (510).

High Level Overview

A high-level overview of the present invention is now provided in accordance with a preferred embodiment.

1. Because of the nature of the process, the transmission of secret information like private keys, is not required.

2. The use of any known cryptographic means to generate the digital signature is permitted therein. The term "digital signature" should be interpreted most broadly. Each manufacturer may decide for themselves what constitutes a digital signature. It could be a standard RSA or DSA digital signature, in other words, the data in the range of addresses are hashed with a cryptographic hash such as SHA-1, and the hash would then be signed. Because digital signatures generally produce only a single bit output indicating the validity of the suspect data, the code protection they provide tends to be defeated fairly easily by attackers. Hence, the invention preferably provides for an alternative use of what is called a digital signet. A signet may be used instead of signatures, because signets are stronger in code integrity applications as they provide a string of bits as output, rather than the typical single bit of the signature. In addition, signets may be part of the code itself which furthers the code protection function. Signet technology is taught in US Patents 6,038,316 and 5,978,482, the teachings of which are incorporated herein by reference. All such solutions are within the scope of this invention.

3. The invention preferably includes a tool to convert old computer program code modules to new verified program code modules via a new signature.

4. The verification can be performed on a one-way basis. In other words, some modules may be used without verification, while others must be verified. The protocol described herein allows for a sensible solution for modules like compression modules, that need no security for themselves, but need it only when connected to another module.

5. One problem related to digital signatures on address ranges in virtual memory is the fact that the operating system loader might relocate a module to a different base address in case of a collision. During this relocation, the loader performs fix-ups on certain instructions in the code which reference absolute addresses. As a consequence, a special hashing technique is preferably required which temporarily reverts these changes to a canonical form so that a hash of an address range always returns the same value, no matter what the absolute base address of the module. This technology is taught in a US patent application filed by International Business Machines on 7-13-1999, entitled "Integrity Checking An Executable Module And Associated Protected Service Provider Module", Patent Application Number 09/352285, (Attorney Docket Number END919990035) a copy of which is placed on the file.

6. Although in the preferred embodiment each module is responsible for the integrity of the other, that is not to say that there is no self-integrity logic for other reasons. Modules also having self-integrity are certainly within the scope of the present invention.

7. In the preferred embodiment, there is no protection against a rogue manufacturer built into this invention. Therefore, manufacturers should form legal obligations, such as by contracts, with each other in order to protect the rights of those manufacturers that agree to participate in the system.

8. There is preferably no session key and no overhead associated with encrypting data on one side of an interface and decrypting it on the other as there is in the prior art.

9. This protocol is as strong as the strongest link. If one module has been successfully hacked, the other intact module would detect that fact since the signature wouldn't match. Then, the second module would refuse to participate in the exchange of data.

Another Embodiment Signature Generation Implementations

As described above, the software secure authentication process comprises a process wherein each manufacturer has to communicate with every other manufacturer to enable the protocol. To accomplish the certification of the modules, several implementations are envisioned. In particular, to assist in the verification process, a certifying agency is provided that performs the generation and storage of signatures in the modules on behalf of all manufacturers. Of course, this requires that manufacturers agree on the technology used for the digital signatures. An alternate implementation comprises a distributed signature generation process. A distributed process provides for a plurality of certifying agencies and individual manufacturers signing modules for themselves with their own signing technologies.

Another Embodiment - A Transformation Tool

Unfortunately, some manufacturers might not agree to enact the protocol as described. Therefore, this possibility is anticipated and by providing a tool that can transform a non-compliant module into one that performs the necessary steps in the protocol. In particular, the tool adds an additional entry point to the module to perform the cross module signature swapping, and also transforms all the other entry points in the module into entry points that check to make sure their caller is coming from the other module's protected area. Of course, the transformed module would be the one that would be signed.

Another Embodiment - One Way Integrity Checking

In another embodiment, the protocol is used in a one-way mode. There are certain software modules, for example music compression modules, that are unconcerned with the nature of the calling entity, such as a calling module. There is no security problem with the music compression modules performing their function, i.e., compressing music for anyone. On the other hand, an electronic music distribution system, which has been entrusted with music and typically wants to enforce limited copying, is concerned that the compression module it is about to send a song to is, in

fact, a good compression module and not some hacked program that will additionally steal the song by illicitly writing the song on the user's disk. In that case, the electronic music distribution module needs a signature from the compression module, but the reverse is not necessarily true.

Another Embodiment - Periodic Signature Checking

Although it is more difficult for the hacker, he may be able to connect to running process after the signature checking has been done, and insert his code patch at that time. Therefore, in one embodiment periodic, random or even each time a module is used, a signature check may be performed to detect newly inserted patches. This may be done based on elapsed time, or based on the number of actual call/returns across the protected interface. Since many digital signatures are relatively expensive, one embodiment of the invention further allows that the subsequent digital signaturing checking may be done with less expensive means, for example, by simply recording the first hash and detecting whether or not a new hash is ever different. Note, that in broad definition of digital signature as contemplated herein, simply checking the hash in the first place would be a type of "digital signature check", although not a particularly strong one.

Conclusion

Therefore, a more efficient method, apparatus and product for securing and verifying the authenticity of data files, for establishing a Secure Authenticated Channel and for securing the communication between two modules has been described in accordance with various embodiments of the present invention that overcomes many of the limitations in the prior art.

Discussion of Hardware and Software Implementation Options

The present invention, as would be known to one of ordinary skill in the art, could be produced in hardware or software, or in a combination of hardware and software. The system, or method, according to the preferred embodiment, may be produced in a single computer system having separate elements or means for performing the individual functions or steps described or claimed or one or more elements or means combining the performance of any of the functions or steps disclosed or claimed, or may

be arranged in a distributed computer system, interconnected by any suitable means as would be known by one of ordinary skill in art.

According to the preferred embodiment, the invention is not limited to any particular kind of computer system but may be used with any general purpose computer, as would be known to one of ordinary skill in the art, arranged to perform the functions described and the method steps described. The operations of such a computer, as described above, may be according to a computer program contained on a medium for use in the operation or control of the computer, as would be known to one of ordinary skill in the art. The computer readable medium which may be used to hold or contain or deliver the computer program product, may be a fixture of the computer such as an embedded memory or may be on a transportable medium such as a disk, as would be known to one of ordinary skill in the art.

The invention is not limited to any particular computer program or logic or language, or instruction but may be practiced with any such suitable program, logic or language, or instructions as would be known to one of ordinary skill in the art. Without limiting the disclosed invention any such computing system can include, inter alia, at least a computer readable medium allowing a computer to read data, instructions, messages or message packets, and other computer readable information from the computer readable medium. The computer readable medium may include nonvolatile memory, such as ROM, Flash memory, floppy disk, Disk drive memory, CD-ROM, and other permanent storage. Additionally, a computer readable medium may include, for example, volatile storage such as RAM, buffers, cache memory, and network circuits.

Furthermore, the computer readable medium may include computer readable information in a transitory state medium such as a network link and/or a network interface, including a wired network or a wireless network, that allow a computer to read such computer readable information.

To summarise, software manufacturers examine their module and determine a range of addresses in memory which the module occupies. A protected range of addresses in memory is predefined to not allow changes, such as patching by hackers. Each manufacturer delivers the range of addresses describing the protected area and a known good version of their module to other manufacturers that they want to interoperate with. The other manufacturers return digital signatures on the protected area, and these digital signatures are stored in the first manufacturer's module.

Correspondingly, the other manufacturers do the same with their own modules. Then, in order to effect a secure communication channel between two modules the modules first pass each other the signatures previously produced. Then, to ensure that communication is being effected with an authentic authorized module, through the use of the signature and the address ranges in the protected area, each module checks that the other module has not been patched. They each further verify that all the entry points in the other module they intend to call are in fact within the protected area. In the event that both modules are verified as being trustworthy, the modules now call each other freely. However, each module, when it is called first verifies that it was called from within the protected area of the other module.

CLAIMS

1. A method for peer to peer authentication of a software module on a single machine, the method comprising the steps of:

determining a protected range of addresses that a software module occupies;

verifying the integrity of code in said protected range; and

verifying that all function calls performed by that module originate from and return to that range.

2. The method as defined in claim 1, wherein the authentication method is performed between two modules so that each module authenticates the other one.

3. The method as defined in claim 2, wherein one or both of the modules deploy one or more mechanisms for self-integrity checking.

4. The method as defined in claim 1, 2 or 3, further comprising the steps of:

delivering the protected address range and the software module to entities that are using the authentication method.

5. The method as defined in claim 4, further comprising the step of:

receiving a returned amended software module with a digital signature on the protected address range.

6. The method as defined in claim 5, wherein the digital signature comprises a digital signature selected from a group of digital signatures consisting of: an RSA signature, a DSA signature, a cryptographic hash such as SHA-1 that is signed, and a digital signet.

7. The method as defined in claim 5 or 6, further comprising a tool which transforms non-compliant software modules into compliant ones.

8. The method as defined in claim 5, further comprising a signature based on a canonical hash that is independent of the location of the module in memory.

9. The method as defined in claim 5, 6, 7 or 8, wherein the digital signature is checked using a checking technique selected from a group of checking techniques comprising: periodic, random and each time modules are used, the checking being performed during operation of the software modules.

10. The method as defined in any of claims 5 to 9, wherein signatures of modules are generated and stored via several implementations wherein the implementations comprise implementations selected from a group of implementations consisting of: a certifying agency for generating and storing the signatures for all manufacturers and a distributed implementation wherein a plurality of certifying agencies and manufacturers distribute the generation and storage of signatures among themselves.

11. The method as defined in any preceding claim further comprising the step of:

verifying that all functions to be called exist within a verified protected range.

12. A computer readable medium comprising instructions for peer to peer authentication of a software module on a single machine, the computer readable medium comprising instructions for:

determining a protected range of addresses that a software module occupies;

verifying the integrity of code in said protected range; and

verifying that all function calls performed by that module originate from and return to that range.

13. The computer readable medium as defined in claim 12, wherein the authentication instructions are performed between two modules so that each module authenticates the other one.

14. The computer readable medium as defined in claim 13, wherein one or both of the modules deploy one or more mechanisms for self-integrity checking.

15. The computer readable medium as defined in claim 12, 13 or 14, further comprising the instruction of:

delivering the protected address range and the software module to entities that are using the authentication method.

16. The computer readable medium as defined in claim 15, further comprising the instruction of:

receiving a returned amended software module with a digital signature on the protected address range.

17. The computer readable medium as defined in claim 16, wherein the digital signature comprises a digital signature selected from a group of digital signatures consisting of: an RSA signature, a DSA signature, a cryptographic hash such as SHA-1 that is signed, and a digital signet.

18. The computer readable medium as defined in claim 16 or 17, further comprising a tool which transforms non-compliant software modules into compliant ones.

19. The computer readable medium as defined in claim 16, further comprising a signature based on a canonical hash that is independent of the location of the module in memory.

20. The computer readable medium as defined in any of claims 16 to 19, wherein the digital signature is checked using a checking technique selected from a group of checking techniques comprising: periodic, random and each time modules are used, the checking being performed during operation of the software modules.

21. The computer readable medium as defined in any of claims 16 to 20, wherein signatures of modules are generated and stored via several implementations wherein the implementations comprise implementations selected from a group of implementations consisting of: a certifying agency for generating and storing the signatures for all manufacturers and a distributed implementation wherein a plurality of certifying agencies and manufacturers distribute the generation and storage of signatures among themselves.

22. The computer readable medium as defined in any of claims 12 to 21 comprising:

verifying that all functions to be called exist within a verified protected range.

23. A system for peer to peer authentication of a software module on a single machine, the system comprising:

a determination unit for determining a protected range of addresses that a software module occupies;

a verification unit for verifying the integrity of code in said protected range; and

a second verification unit for verifying that all function calls performed by that module originate from and return to that range.

24. The system as defined in claim 23, wherein the authentication is performed between two modules so that each module authenticates the other one.

25. The system as defined in claim 24, wherein one or both of the modules deploy one or more mechanisms for self-integrity checking.

26. The system as defined in claim 23, 24 or 25 further comprising:

a sending unit for delivering the protected address range and the software module to entities that are using the authentication method.

27. The system as defined in claim 26, further comprising:

a reception unit for receiving a returned amended software module with a digital signature on the protected address range.

28. The system as defined in claim 27, wherein the digital signature comprises a digital signature selected from a group of digital signatures comprising: an RSA signature, a DSA signature, a cryptographic hash such as SHA-1 that is signed, and a digital signet.

29. The system as defined in claim 27 or 28, further comprising a tool unit which transforms non-compliant software modules into compliant ones.

30. The system as defined in claim 27, further comprising a signature based on a canonical hash that is independent of the location of the module in memory.

31. The system as defined in any of claims 27 to 30, wherein the digital signature is checked using a checking technique selected from a group of checking techniques consisting of: periodic, random and each time modules are used, the checking being performed during operation of the software module.

32. The system as defined in any of claims 27 to 31, wherein signatures of modules are generated and stored via several implementations wherein the implementations comprise implementations selected from a group of implementations consisting of: a certifying agency for generating and storing the signatures for all manufacturers and a distributed implementation wherein a plurality of certifying agencies and manufacturers distribute the generation and storage of signatures among themselves.

33. The system as defined in any of claims 23 to 32 comprising:

a verification unit for verifying that all functions to be called exist within a verified protected range.

100

PRIOR ART

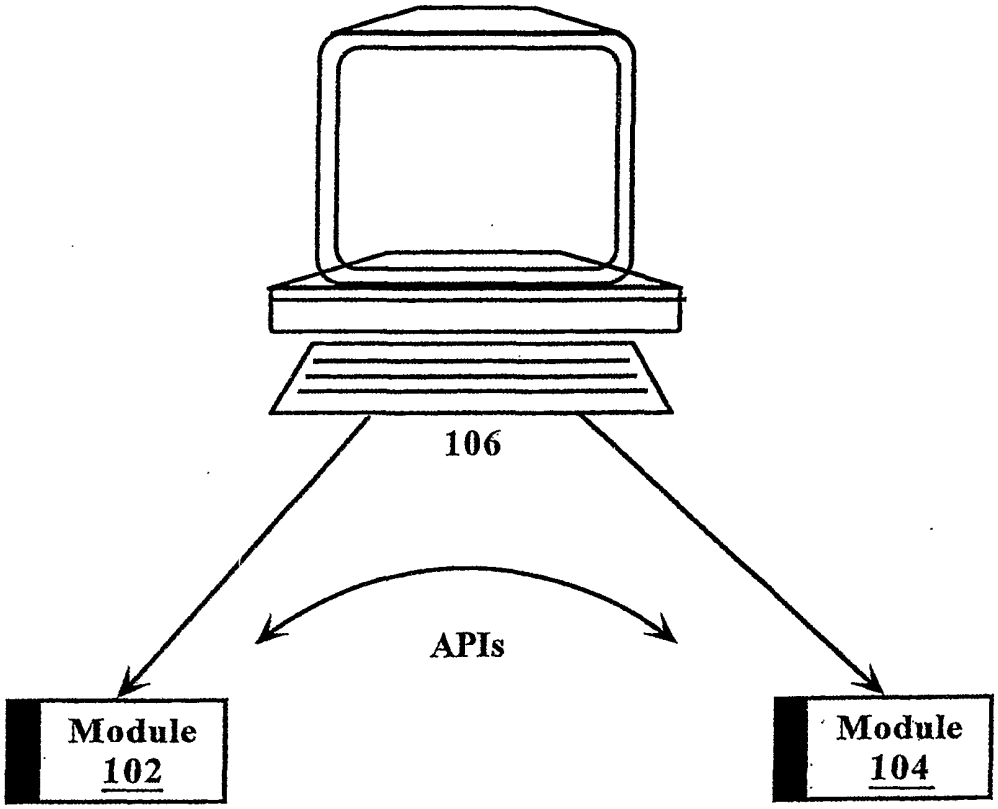


FIG. 1

PRIOR ART

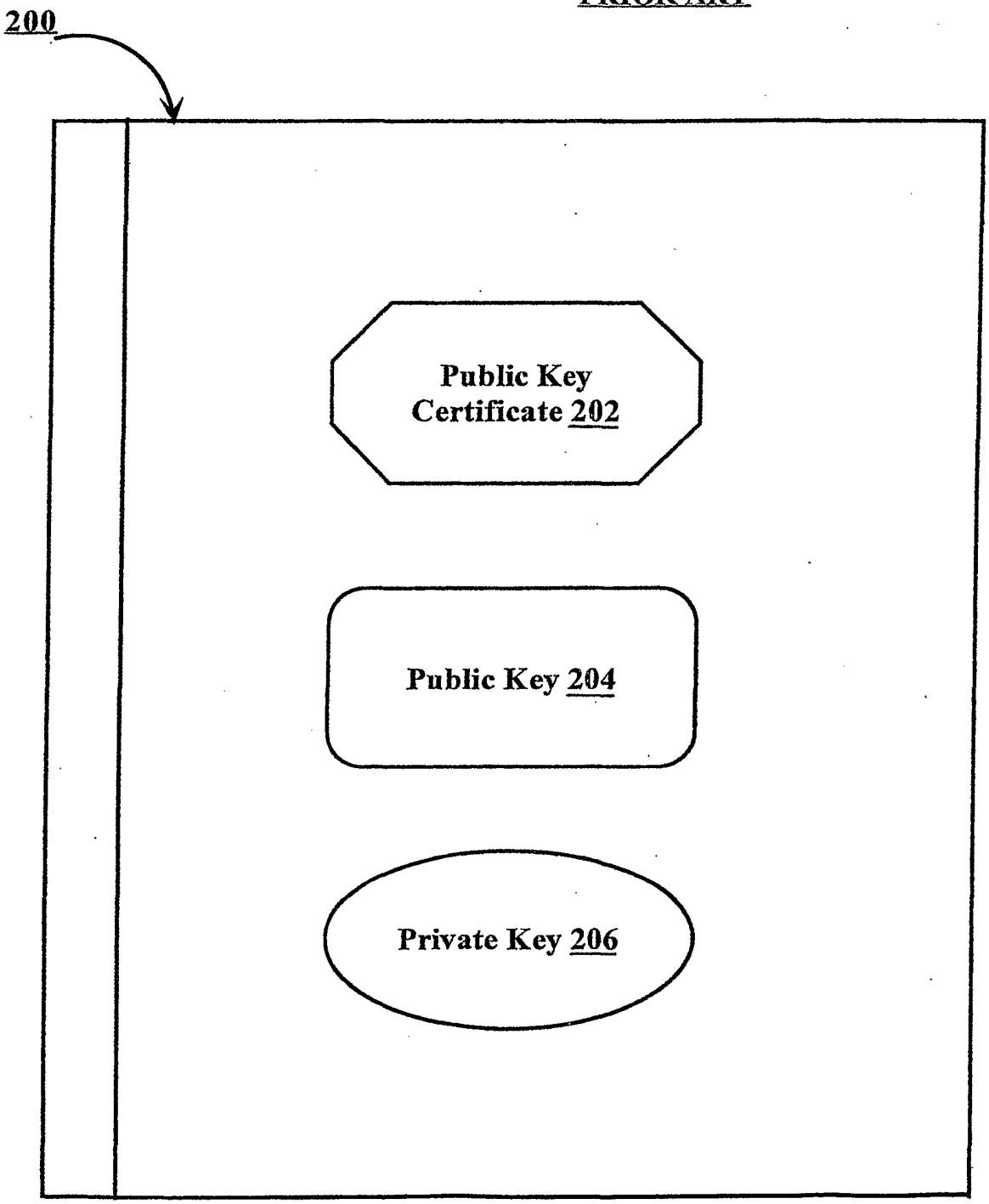


FIG. 2

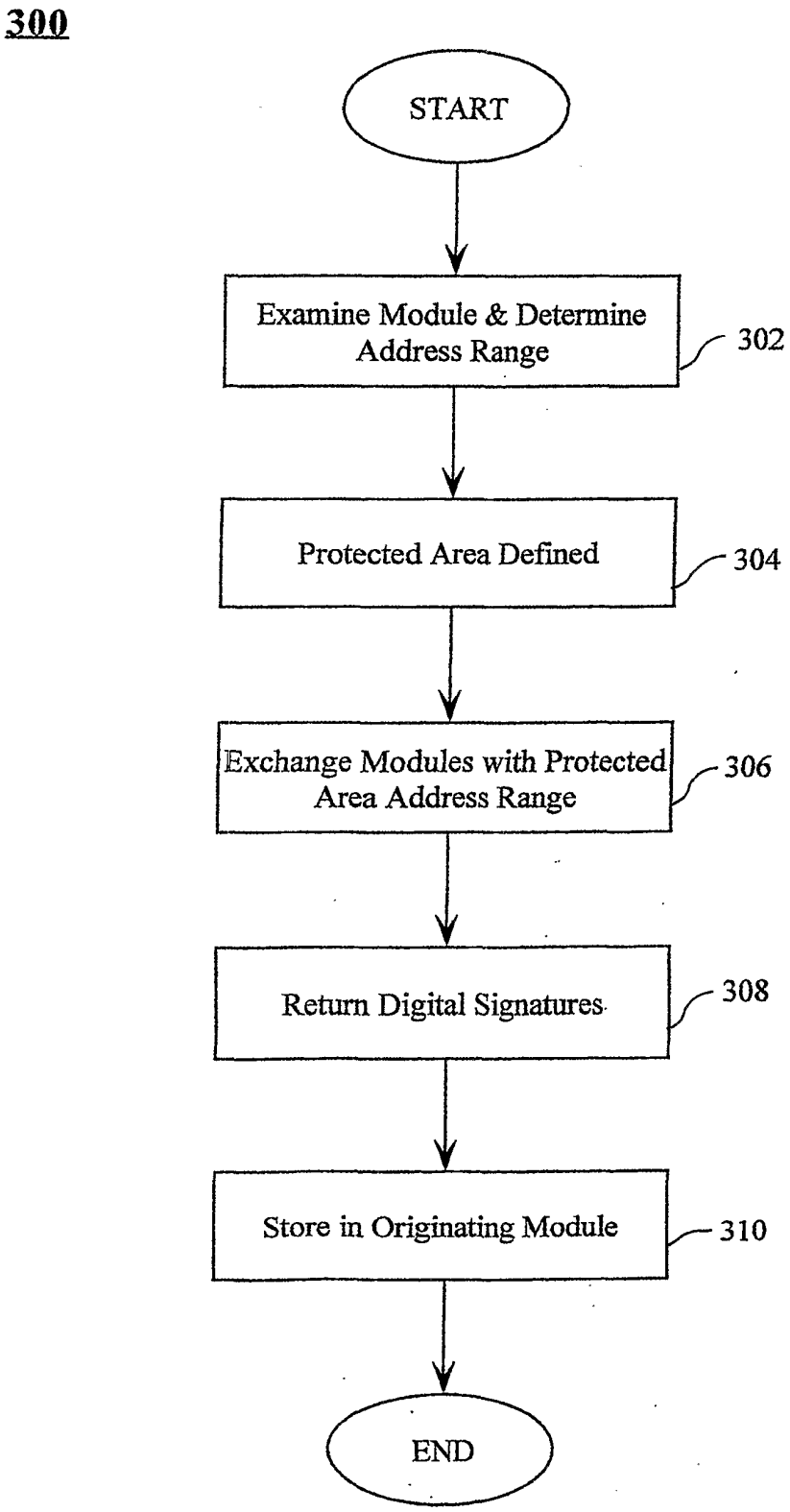
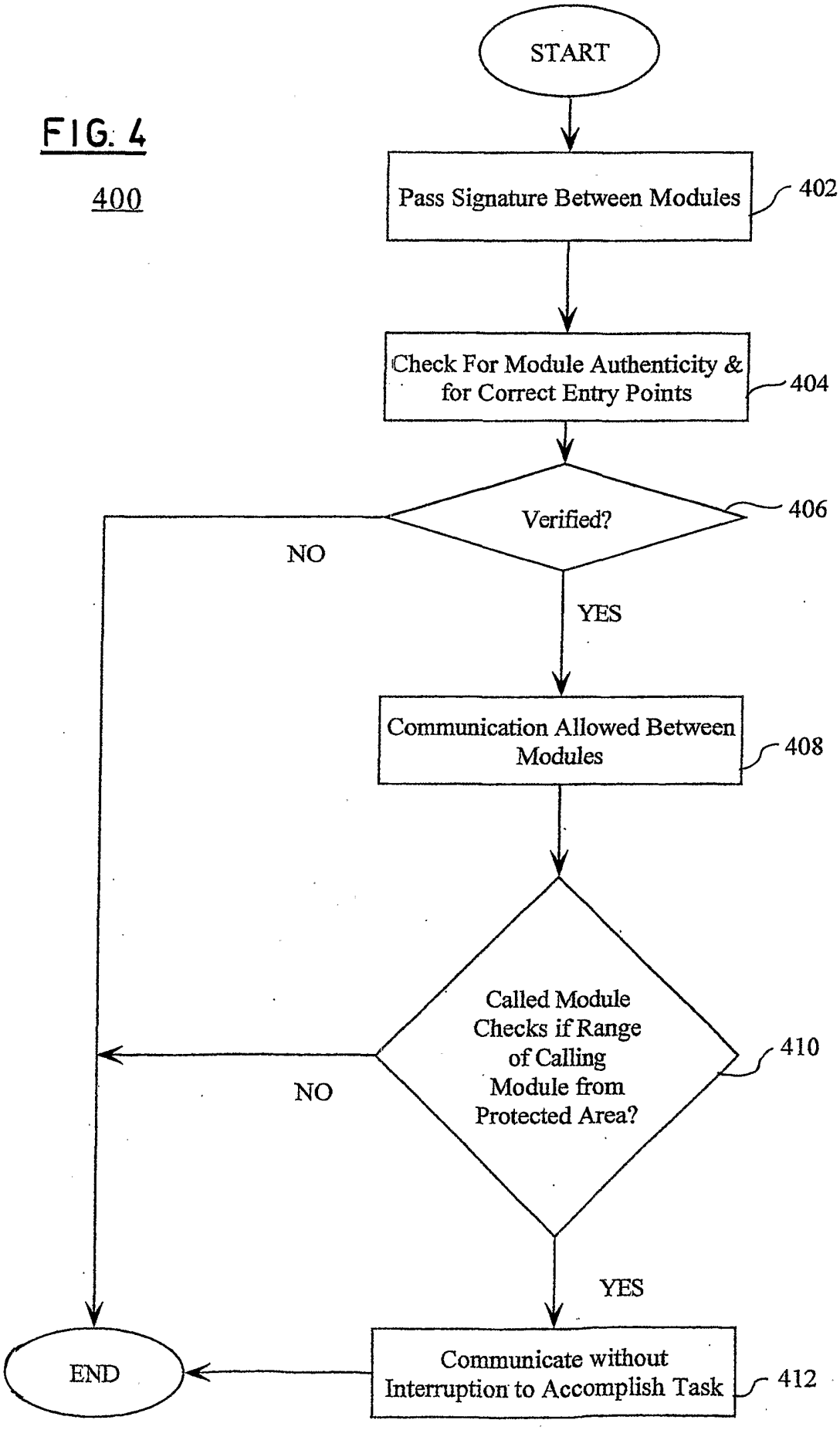

300
START
Examine Module & Determine Address Range
302
Protected Area Defined
304
Exchange Modules with Protected Area Address Range
306
Return Digital Signatures
308
Store in Originating Module
310
END

FIG. 3

FIG. 4

400


START
Pass Signature Between Modules
402
Check For Module Authenticity & for Correct Entry Points
404
Verified?
406
NO
YES
Communication Allowed Between Modules
408
Called Module Checks if Range of Calling Module from Protected Area?
410
NO
YES
Communicate without Interruption to Accomplish Task
412
END

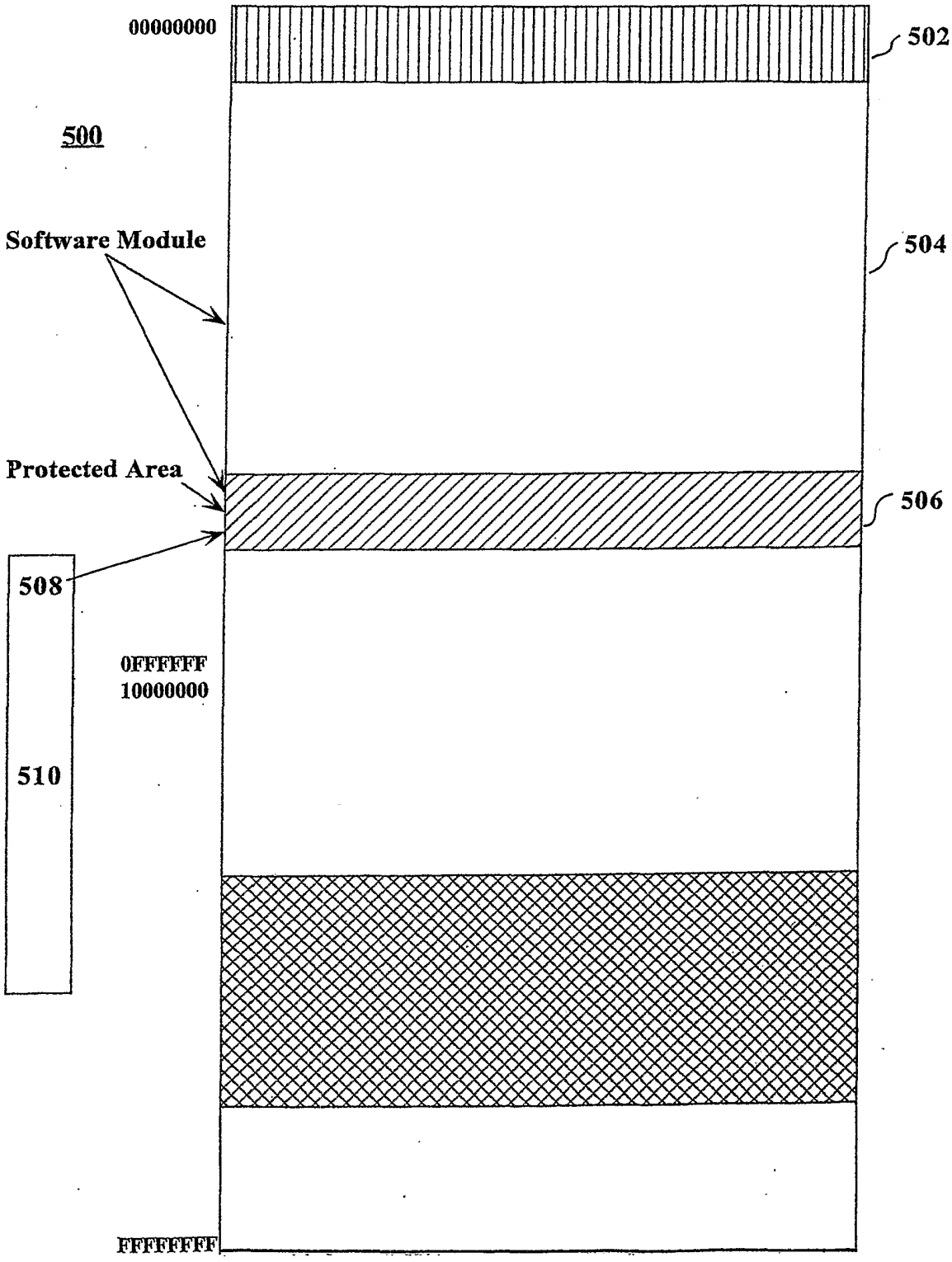

00000000
502
500
Software Module
504
Protected Area
506
508
0FFFFFFF
10000000
510
FFFFFFFF

FIG. 5