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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
19 May 2016 (19.05.2016)

WIPO | PCT

(10) International Publication Number
WO 2016/075570 A1

(51) International Patent Classification:
G06F 9/44 (2006.01)

(21) International Application Number:
PCT/IB2015/058058

(22) International Filing Date:
20 October 2015 (20.10.2015)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
1420046.3 11 November 2014 (11.11.2014) GB

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

(71) Applicants *(for MG only)*: **IBM (CHINA) INVESTMENT COMPANY LTD.** [CN/CN]; 25/F, Pangu Plaza, No. 27, Central North 4th Ring Road, Chaoyang District, Beijing 100101 (CN). **IBM RESEARCH GMBH** [CH/CH]; IBM Research - Zurich, Saeumerstrasse 4, 8803 Rueschlikon (CH).

(72) Inventors: **BACHER, Utz**; c/o IBM Deutschland Research & Development GmbH, Schoenaicher Strasse 220, 71032 Boeblingen (DE). **BUENDGEN, Reinhard, Theodor**; c/o IBM Deutschland Research & Development GmbH, Schoenaicher Strasse 220, 71032 Boeblingen (DE).

(74) Agent: **KLETT, Peter M.**; IBM Research GmbH, IBM Research - Zurich, Intellectual Property Law, Saeumerstrasse 4, 8803 Rueschlikon (CH).

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

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK,

[Continued on next page]

(54) Title: PROCESSING GUEST EVENT IN HYPERVISOR-CONTROLLED SYSTEM

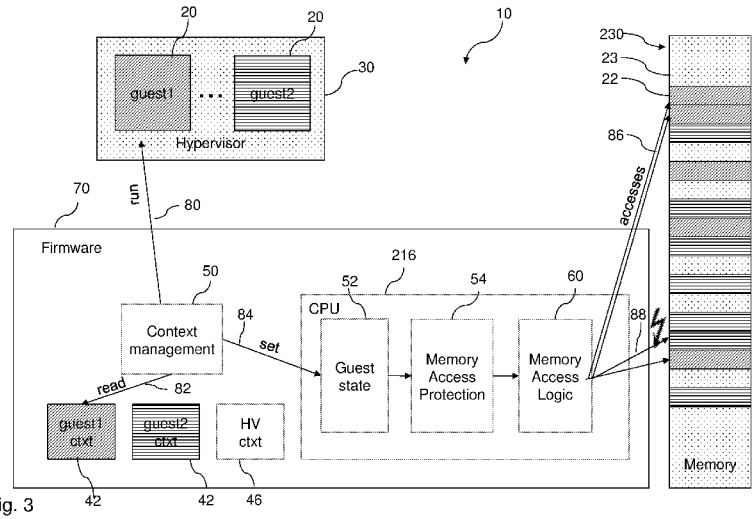

Fig. 3

(57) Abstract: A method for processing a guest event in a hypervisor-controUed system (lO), comprising the steps: (i) the guest event triggering a first firmware service being specific for the guest event in a firmware (70), the guest event being associated with a guest (20) and with a guest state (52) and a protected guest memory (22) accessible only by the guest (20) and the firmware (70), and a guest key (24); (ii) the firmware (70) processing information associated with the guest event, comprising information of the guest state (52) and the protected guest memory (22), and presenting only a subset of the information of the guest state (52) and the protected guest memory (22) to a hypervisor (30), wherein the subset of the information is selected to suffice for the hypervisor(30) to process the guest event; (iii) the firmware(70) retaining a part of the information of the guest state (52) and the protected guest memory (22) that is not being sent to the hypervisor(30); (iv) the hypervisor (30) processing the guest event based on the received subset of the information of the guest state (52) and the protected guest memory (22) and sending a process result to the firmware (70) triggering a second firmware service being specific for the guest even; (v)the firmware (70) processing the received process result together with the part of the information of the guest state (52) and the protected guest memory (22) that was not sent to the hypervisor (30),generating a state and/or memory modification; (vi) the firmware (70) performing the state and/or memory modification associated with the guest event at the protected guest memory (22).

WO 2016/075570 A1

SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report (Art. 21(3))*

D E S C R I P T I O N

PROCESSING GUEST EVENT IN HYPERVISOR-CONTROLLED SYSTEM

The present invention relates in general to data processing systems, and in particular, to a method and a system for processing a guest event in a hypervisor-controlled system.

BACKGROUND

Customer adoption of public Clouds is limited to non-mission critical data. Very often, the core business data is an essential asset to a customer, and the data's confidentiality is critical for business success. As long as customers do not trust Cloud environments, Cloud adoption of those business sensitive environments stays minimal. Among the main concerns of customers are lack of trust in the Cloud provider and the security of the Cloud.

Trust in the Cloud provider is critical since an administrator of the provider has the capability to fully inspect the customer's workload and data. This potential breach for espionage is the reason for being reluctant for many customers.

Trust in Cloud security relates to the threat of a hypervisor breach, i.e. if an attacker gains access to the hypervisor, the customer's workload and data are at risk again.

Approaches to guarantee confidentiality and privacy are limited to input/output (I/O) at this time: network encryption like secure sockets layer (SSL) can be used to encrypt socket connections and disk encryption tools like dm-crypt in LINUX can be used to encrypt data on a disk device.

A trusted platform module (TPM) has been developed to ensure the boot chain is valid at the time the customer runs its workload, yet it is not deployed in a Cloud environment. Also, TPMs do not ensure privacy but integrity of the setup at best.

All these technologies, even if used, do not address the issue that a hypervisor can always fully inspect its guests, where guests may in general be virtual machines on a hypervisor-controlled system, and read memory contents with potentially sensitive data of the image running in the guest. The concerns mentioned above cannot be eliminated by these technologies.

US 2011/0302400 Al describes a method that generally includes receiving, by a trust anchor on a central processing unit (CPU) having a plurality of processing cores, a virtual machine (VM) image. As received, the VM image is encrypted using a VM image encryption key. The method also includes obtaining the VM image encryption key and configuring a first encrypt/ decrypt block with the VM image encryption key. The method also includes generating a memory session key and configuring a second encrypt/decrypt block with the memory session key. The method also includes fetching one or more pages of the VM image into a memory accessible by the plurality of processing cores. Each fetched page is decrypted by the first encrypt/decrypt block using the VM image encryption key and subsequently encrypted by the second encrypt/decrypt block using the memory session key.

SUMMARY

It is an objective of the invention to provide a method for securely processing guest data in an untrusted Cloud environment.

Another objective is to provide a system for securely processing guest data in an untrusted Cloud environment.

These objectives are achieved by the features of the independent claims. The other claims, the drawings and the specification disclose advantageous embodiments of the invention.

According to a first aspect of the invention a method is proposed for processing a guest event in a hypervisor-controlled system, comprising the steps: (i) the guest event triggering a first firmware service being specific for the guest event in a firmware, the guest event being associated with a guest and with a guest state and a protected guest memory accessible only by the guest and the firmware, and a guest key; (ii) the firmware processing information associated with the guest event, comprising information of the guest state and the protected guest memory, and presenting only a subset of the information of the guest state and the protected guest memory to a hypervisor, wherein the subset of the information is selected to suffice for the hypervisor to process the guest event; (iii) the firmware retaining a part of the information of the guest state and the protected guest memory that is not being sent to the hypervisor; (iv) the hypervisor processing the guest event based on the received subset of the information of the guest state and the protected guest memory and sending a process result to the firmware triggering a second firmware service being specific for the guest event; (v) the firmware processing the received process result together with the part of the information of the guest state and the protected guest memory that was not sent to the hypervisor, generating a state and/or memory modification; (vi) the firmware performing the state and/or memory modification associated with the guest event at the protected guest memory.

The first firmware service may favorably comprise steps (ii) and (iii), namely (ii) the firmware processing information associated with the guest event, comprising information of the guest state and the protected guest memory, and presenting only a subset of the information of the guest state and the protected guest memory to a hypervisor, wherein the subset of information is selected to suffice for the hypervisor to process the guest event; and (iii) the firmware retaining a part of the information of the guest state and the protected guest memory that was not being sent to the hypervisor.

The second firmware service may favorably comprise steps (v) and (vi), namely (v) the firmware processing the received process result together with the part of the information of the guest state and the protected guest memory that was not sent to the hypervisor, generating a state and/or memory modification; and (vi) the firmware performing the state and/or memory modification associated with the guest event at the protected guest memory.

Particularly, a method for processing a guest event in a hypervisor-controlled system is addressed exhibiting the advantage of protecting guest confidentiality. Thus the method according to the invention generally describes securely managing virtual machines while maintaining privacy of virtual machine contents towards the hypervisor comprising one or more VMs each with resources including concealed memory and context data, a hypervisor managing VM resources and VM states, CPU assisted virtualization enforcing restricted access of the hypervisor to VM state/memory/context through firmware services.

Particularly, the method according to the invention describes processing a guest event in a hypervisor-controlled system where guest data is encrypted with a guest key not accessible to the hypervisor and where CPUs and firmware are considered trusted

and have access to the guest key when running in guest context. The firmware in this context particularly means system software implemented in a hardware based environment.

As said the method according to the invention describes running virtual machines from protected memory such that the protected memory belonging to a specific virtual machine cannot be accessed by another virtual machine even if the hypervisor allowed such memory access. Also, the method prevents that a hypervisor can always fully inspect its guests, i.e. virtual machines/images, and read memory contents with potentially sensitive data. The advantage is that the described method does not use processes like authentication of a trust anchor with a (customer) key service. It is able to deal with interrupts or hypervisor intercepts. The described method does not require an attestation module (e.g. a TPM) on the CPU.

In an advantageous embodiment, particularly for enabling access to a memory page for the hypervisor, the method may further comprise (i) the hypervisor requesting access to a page of the protected guest memory from firmware; (ii) the firmware disabling access for the guest to that page; (iii) the firmware encrypting that page with the guest key; and (iv) the firmware enabling access for the hypervisor to that page.

Further, particularly for enabling access to a memory page for a guest, the method may favorably comprise (i) the hypervisor providing an encrypted page to the firmware; (ii) the firmware disabling access for the hypervisor to that page; (iii) the firmware decrypting that page; and (iv) the firmware enabling access for the guest to that page.

According to the above described embodiments, particularly for allowing decryption of a page, the method may advantageously comprise (i) the firmware computing an integrity value of the

page to be encrypted and made accessible to the hypervisor; (ii) the hypervisor providing an encrypted page to be added to the protected pages of a virtual machine to the firmware; and (iii) the firmware only allowing adding the decrypted page to the protected pages of the virtual machine when the integrity value matches the page.

Further, particularly for a secure deployment and execution of a virtual machine, the method may advantageously comprise the steps (i) providing the guest with the guest key being encrypted with the public key associated with a private key of the hypervisor-controlled system for transfer to the key store of the hypervisor-controlled system; (ii) providing the hypervisor-controlled system with the private key, being stored in the hypervisor-controlled system and being used to decrypt the encrypted guest key; (iii) the guest key being used to encrypt and decrypt the guest data, when being transferred out of or into the protected guest memory. Thus a secure deployment and execution of a virtual machine in a hypervisor-controlled system may be enabled.

For encryption/decryption of guest data, the hypervisor controlled system may get a key pair; its private key may be stored in the trusted firmware or hardware, and may be used to decrypt guest keys. The public key may be used to encrypt (and transfer) the private guest key to the trusted firmware or hardware, in which the guest key may be stored and used securely.

The guest may generate a key as well. The guest key may be encrypted with the public key of the hypervisor controlled system before it is transferred to the hypervisor controlled system. The trusted firmware or hardware of the hypervisor controlled system may use this guest key to encrypt the guest's memory (but only when running the guest in the context of the

CPU virtualization function). The guest key may also be used to deploy images from the guest in a Cloud environment.

Favorably, particularly for boot image generation and deployment, the method according to the invention may further comprise the steps (i) generating a boot image by a client or customer; (ii) encrypting the boot image with the guest key; (iii) transferring the encrypted boot image to a boot disk; (iv) loading the encrypted boot image of a guest by the hypervisor to the guest memory; (v) transforming the guest memory (23) into the protected guest memory (22); (vi) decrypting contents of the protected guest memory; (vii) starting an execution of a guest as a virtual machine at the CPU level, where the guest is defined by an area of an encrypted memory, an area of an unencrypted memory and an encrypted guest key. The guest key may be known only to the client and the guest respectively and the trusted firmware or hardware in guest context and for the transport to the trusted firmware or hardware the guest key may be encrypted with the public key associated with the private key of the hypervisor controlled system. It need not be known to a Cloud operator or the hypervisor.

The hypervisor may read contents of a boot image from the boot disk into the guest memory without relocation, where the boot disk contents may comprise a kernel, parameters, an initial ram disk. Loading the boot image may also comprise mounting a conventionally encrypted (e.g. via dm-crypt, a usually applied LINUX encrypting tool) root file system. Further the boot disk contents may comprise a kernel execution (kexec) environment that loads a new kernel from a conventionally encrypted target boot device.

The CPU architecture may be extended to provide a well-defined means to access a guest state, where access methods may provide the hypervisor only with the necessary information to perform

its tasks (e.g. handling traps). However, the guest memory and register file may not be accessible outside of said access methods. Thus confidentiality of the guest may be protected, since the hypervisor cannot read a guest state or guest data entirely. The register file may not be accessible to the hypervisor directly, but may be stored away and restored through a hypervisor service. Other contexts than the guest itself may only see encrypted memory contents, as the hypervisor may not see the unencrypted guest memory. An area of the guest memory may remain unencrypted and unprotected in order to exchange data with the hypervisor or I/O devices. An I/O scratch area may be outside the protected memory area of the guest.

In an advantageous embodiment, particularly for an interaction between the virtual machine and the hypervisor, the method may further comprise the steps (i) keeping the range of the protected guest memory or registers associated with the guest event being not accessible to the hypervisor in decrypted form; (ii) extending a virtualization function of the hypervisor-controlled system by access methods to specific guest data associated with the guest event. This step may be advantageous for a hypervisor operation, but may not reveal data or code (other than reason and relevant parameters for instructions that trap) of the guest and enable to continue execution of the guest event. Some traps may be disabled entirely since they may only be meaningful (e.g. single stepping), when a hypervisor has full access to a guest.

Advantageously, particularly for an I/O process of a guest, the method may further comprise the steps (i) defining a non-protected memory area for I/O buffers and I/O control structures of the guest outside the area of the protected guest memory; (ii) starting the I/O process by the guest using that non-protected area of the guest memory; (iii) the virtualization function of the hypervisor-controlled system generating a guest

event; (iv) the hypervisor reading a reason for the guest event and performing the I/O process. By putting the I/O buffers outside the protected guest memory, the hypervisor and I/O devices may have access to I/O control structures and data.

The hypervisor may store pages on a hypervisor owned swap device, where still the page contents may be encrypted. The CPU's virtualization function may trap, where the hypervisor may read a reason for the trap (e.g. „page fault") and may read a guest address. Then the hypervisor may put the page back to the same guest address, which maintains guest data integrity when encryption results are kept non-relocatable. Then the hypervisor may restart the guest.

In an advantageous embodiment, particularly for establishing trust in the described method, the method may further comprise checking a guest integrity with a checking process that knows the guest key, the checking process comprising the steps (i) the guest reading a memory content in clear text from the protected guest memory transferring an arbitrary range of the protected guest memory via a secure communication path to the checking process; (ii) requesting the same memory range of the protected guest memory from the hypervisor and transferring it to the checking process; (iii) comparing the memory content obtained from the guest with the result of decrypting the memory content obtained from the hypervisor; (iv) delivering a comparison result depending on the contents of the two memory ranges; (v) returning the result of the checking process being positive if the comparison result equals zero, otherwise being negative. These method steps may be especially advantageous because the hypervisor is not able to read/inject code or data since it is not provided with the key for guest memory decryption/encryption.

According to a further advantageous aspect of the invention a data processing program for execution in a data processing system is proposed comprising an implementation of an instruction set for performing a method as described above when the data processing program is run on a computer.

Further a favorable computer program product is proposed comprising a computer usable medium including a computer readable program, wherein the computer readable program when executed on a computer causes the computer to perform a method for processing a guest event in a hypervisor-controlled system, comprising the steps: (i) the guest event triggering a first firmware service being specific for the guest event in a firmware, the guest event being associated with a guest and with a guest state and a protected guest memory accessible only by the guest and the firmware, and a guest key; (ii) the firmware processing information associated with the guest event, comprising information of the guest state and the protected guest memory, and presenting only a subset of the information of the guest state and the protected guest memory to a hypervisor, wherein the subset of the information is selected to suffice for the hypervisor to process the guest event; (iii) the firmware retaining a part of the information of the guest state and the protected guest memory that is not being sent to the hypervisor; (iv) the hypervisor processing the guest event based on the received subset of the information of the guest state and the protected guest memory and sending a process result to the firmware triggering a second firmware service being specific for the guest event; (v) the firmware processing the received process result together with the part of the information of the guest state and the protected guest memory that was not sent to the hypervisor, generating a state and/or memory modification; (vi) the firmware performing the state and/or memory modification associated with the guest event at the protected guest memory.

As will be appreciated by one skilled in the art, aspects of the present invention may be embodied as a system, method or computer program product. Accordingly, aspects of the present invention may take the form of an entirely hardware embodiment, an entirely software embodiment (including firmware, resident software, micro-code, etc.) or an embodiment combining software and hardware aspects that may all generally be referred to herein as a "circuit," "module" or "system."

Furthermore, aspects of the present invention may take the form of a computer program product embodied in one or more computer readable medium(s) having computer readable program code embodied thereon.

Any combination of one or more computer readable medium(s) may be utilized. The computer readable medium may be a computer readable signal medium or a computer readable storage medium. A computer readable storage medium may be, for example, but not limited to, an electronic, magnetic, optical, electromagnetic, infrared, or semiconductor system, apparatus, or device, or any suitable combination of the foregoing. More specific examples (a non-exhaustive list) of the computer readable storage medium would include the following: an electrical connection having one or more wires, a portable computer diskette, a hard disk, a random access memory (RAM), a read-only memory (ROM), an erasable programmable read-only memory (EPROM or Flash memory), an optical fiber, a portable compact disc read-only memory (CD-ROM), an optical storage device, a magnetic storage device, or any suitable combination of the foregoing. In the context of this document, a computer readable storage medium may be any tangible medium that can contain, or store a program for use by or in connection with an instruction execution system, apparatus, or device.

A computer readable signal medium may include a propagated data signal with computer readable program code embodied therein, for example, in baseband or as part of a carrier wave. Such a propagated signal may take any of a variety of forms, including, but not limited to, electro-magnetic, optical, or any suitable combination thereof. A computer readable signal medium may be any computer readable medium that is not a computer readable storage medium and that can communicate, propagate, or transport a program for use by or in connection with an instruction execution system, apparatus, or device.

Program code embodied on a computer readable medium may be transmitted using any appropriate medium, including but not limited to wireless, wireline, optical fiber cable, RF, etc., or any suitable combination of the foregoing.

Computer program code for carrying out operations for aspects of the present invention may be written in any combination of one or more programming languages, including an object oriented programming language such as Java, Smalltalk, C++ or the like and conventional procedural programming languages, such as the "C" programming language or similar programming languages. The program code may execute entirely on the user's computer, partly on the user's computer, as a stand-alone software package, partly on the user's computer and partly on a remote computer or entirely on the remote computer or server. In the latter scenario, the remote computer may be connected to the user's computer through any type of network, including a local area network (LAN) or a wide area network (WAN), or the connection may be made to an external computer (for example, through the Internet using an Internet Service Provider).

Aspects of the present invention are described below with reference to block diagrams of methods, apparatus (systems) and computer program products according to embodiments of the

invention. It will be understood that each block of the flowchart illustrations and/or block diagrams, and combinations of blocks in the block diagrams, can be implemented by computer program instructions. These computer program instructions may be provided to a processor of a general purpose computer, special purpose computer, or other programmable data processing apparatus to produce a machine, such that the instructions, which execute via the processor of the computer or other programmable data processing apparatus, create means for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks.

These computer program instructions may also be stored in a computer readable medium that can direct a computer, other programmable data processing apparatus, or other devices to function in a particular manner, such that the instructions stored in the computer readable medium produce an article of manufacture including instructions which implement the function/act specified in the block diagram block or blocks.

The computer program instructions may also be loaded onto a computer, other programmable data processing apparatus, or other devices to cause a series of operational steps to be performed on the computer, other programmable apparatus or other devices to produce a computer implemented process such that the instructions which execute on the computer or other programmable apparatus provide processes for implementing the functions/acts specified in the block diagram block or blocks.

Due to a further aspect of the invention, a data processing system for execution of a data processing program is proposed, comprising software code portions for performing a method described above.

BRIEF DESCRIPTION OF THE SEVERAL VIEWS OF THE DRAWINGS

The present invention together with the above-mentioned and other objects and advantages may best be understood from the following detailed description of the embodiments, but not restricted to the embodiments, wherein is shown in:

Fig. 1 a stack of components in a hypervisor-controlled system according to prior art;

Fig. 2 a general overview of a method for a secure execution of guests in an insecure environment according to an embodiment of the invention;

Fig. 3 a system diagram of a hypervisor-controlled system for a secure execution of guests in an insecure environment according to an embodiment of the invention;

Fig. 4 a generic flow chart for running a secure guest according to an embodiment of the invention;

Fig. 5 a detailed flow chart for interception handling with a secure guest running according to an embodiment of the invention;

Fig. 6 an example embodiment of a data processing system for carrying out a method according to the invention.

DETAILED DESCRIPTION

In the drawings, like elements are referred to with equal reference numerals. The drawings are merely schematic

representations, not intended to portray specific parameters of the invention. Moreover, the drawings are intended to depict only typical embodiments of the invention and therefore should not be considered as limiting the scope of the invention.

Figure 1 shows a stack of components in a hypervisor-controlled system according to prior art. The different components comprise one or more guests 20, realized as a virtual machine, running on a hypervisor-controlled system as a virtual server system, consisting of firmware 70, hardware 72, as e.g. one or more CPUs, memory, I/O devices 74 for storage networking. A hypervisor 30 manages the resources of the hardware 72 and I/O devices 74 and allocates appropriate portions of these resources to the guests 20. In a Cloud environment the guest VM 20 is operated by a client or customer, whereas the hypervisor 30 is operated by a Cloud provider, who may be untrusted by the client. The firmware 70 for the virtual server system, as well as the hardware 72, are manufactured by a hardware vendor, who may be considered as trusted. It is an objective of the invention to provide a method for securely processing the guest VM 20 in a Cloud environment where the Cloud provider may not be trusted.

In one embodiment, the first firmware service may favorably comprise at least one of the steps (ii) and (iii) of the description above, wherein in step (ii), the firmware 70 processing information associated with the guest event, comprising information of the guest state 52 and the protected guest memory 22, and presenting only a subset of the information of the guest state 52 and the protected guest memory 22 in decrypted form to a hypervisor 30, wherein the subset of information is selected to suffice for the hypervisor 30 to process the guest event. Step (iii) comprises the firmware 70 retaining a part of the information of the guest state 52 and the protected guest memory 22 that was not being sent to the

hypervisor 30.

In one embodiment, the second firmware service may favorably comprise at least one of the steps (v) and (vi) of the description above, wherein step (v) comprises the firmware 70 processing the received process result together with the part of the information of the guest state 52 and the protected guest memory 22 that was not sent to the hypervisor 30, generating a state and/or memory modification; and wherein in step (vi) the firmware 70 performing the state and/or memory modification associated with the guest event at the protected guest memory 22.

Figure 2 gives a general overview of a method for a secure execution of guests 20 in an insecure environment of a hypervisor-controlled system 10 according to an embodiment of the invention. This may be achieved in analogy to a secure socket layer, where secure operation (i.e. messaging) is also achieved over an unsecure medium. A hypervisor 30, which is considered as untrusted, may control secure guests 20 as well as unsecure guests 40 at the same time and in the same system. The system 10, that runs the hypervisor 30 and the guests 20, 40 on a CPU 216 maintains a specific context 42 for each secure guest 20, a specific context 44 for each unsecure guest 40 and a context 46 for the hypervisor 30 respectively. Each context 42 of a secure guest 20 contains a guest key 24 associated which the according secure guest 20. The memory of each secure guest 20 is protected against access form code running in a context that does not belong to the secure guest 20. When accessed from a context not belonging to the secure guest 20 the guest memory 23 is encrypted with the guest key 24 of the according secure guest 20. These guests, 20, 40 as well as the hypervisor 30 may run on the system 10 in their own contexts 42, 44, 46. Also on the system 10 there is the hypervisor 30 running in its own context 46. When a CPU 216 runs in one context it has no access

to information maintained by another context. When a CPU 216 runs in a guest context 42 of a secure guest 20 it has access to the guest key 24 of that guest 20 to encrypt and decrypt data of that guest 20. Further a CPU 216 enters a context of a guest 20, 40 or a hypervisor 30 only if it processes code of that guest 20, 40 or that hypervisor 30 respectively or if it runs a firmware service in support of that guest 20, 40 or that hypervisor 30 respectively. The firmware 70 also comprises a private key 26 of the system 10.

Figure 3 shows a system diagram of a hypervisor-controlled system 10 for a secure execution of guests 20 in an insecure environment according to an embodiment of the invention. Numerals referenced in the following description which are not shown in Figure 3, reference objects of Figures 1 and 2.

First generic functions of an execution of guests 20 in an insecure environment according to an embodiment of the invention will be explained using the diagram in Figure 3, before the behavior of the hypervisor-controlled system for handling interrupts or interceptions due to a guest event will be discussed.

The diagram in Figure 3 shows two guests 20, named guest 1 and guest 2, controlled by a hypervisor 30, all components running on a firmware 70 comprising a CPU 216, which is connected to a memory 230. The CPU 216 comprises access means for a state of guest context of the CPU's virtualization function, where the access is controlled based on the context the CPU 216 is in.

The firmware 70 is running a context management function 50, which controls the different contexts under which the guests 20 as well as the hypervisor 30 are operating. The context management function 50 is triggering memory access operations through the CPU 216, where guest state functions 52, memory

access protection 54 as well as a memory access logic 60 is implemented. The context management function 50 controls running of the guests 20 on the hypervisor 30, as indicated by the operation 80.

Shortly, the method according to the invention for processing a guest event in a hypervisor-controlled system 10 is comprising the following steps as shown in Figure 3. These steps are referenced in the description. First a guest event, originating from the guest 1 in the embodiment shown in Figure 3, is triggering a first firmware service being specific for the guest event in the firmware 70, where the guest event is being associated with the guest 20 and with a guest state 52 and a protected guest memory 22 accessible only by the guest 20 and the firmware 70, and a guest key 24. This is controlled by the context management function 50 by reading guest context 42, indicated by operation 82. Secondly the firmware 70 is processing information associated with the guest event, comprising information of the guest state 52 and the protected guest memory 22, and presenting only a subset of the information of the guest state 52 and the protected guest memory 22 to the hypervisor 30. The subset of the information is selected to suffice for the hypervisor 30 to process the guest event. Thirdly the firmware 70 is retaining a part of the information of the guest state 52 and the protected guest memory 22 that was not being sent to the hypervisor 30. Fourthly the hypervisor 30 is processing the guest event based on the received subset of the information of the guest state 52 and the protected guest memory 22 and sending a process result to the firmware 70 triggering a second firmware service being specific for the guest event. Fifthly the firmware 70 is processing the received process result together with the part of the information of the guest state 52 and the protected guest memory 22 that was not sent to the hypervisor 30, generating a state and/or memory modification. Therefore the context management function 50 sets

the guest state 52 in the CPU 216 by operation 84, where information is passed to a memory access protection function 54 and finally to a memory access logic 60, controlling access to the memory 230. Sixthly the firmware 70 is performing the state and/or memory modification associated with the guest event at the protected guest memory 22. This is performed by the memory accesses 86, where memory access 86 is allowed only to the protected memory 22 of the respective guest 20 and not to the memory of a guest which is not the originator of the guest event, as indicated by the operation 88.

The hypervisor 30 has only limited access to data and/or code in the memory 230, because there exist ranges which are encrypted with a guest key 24 and other ranges which are visible to the hypervisor 30. Further there are protected guest memory 22 ranges, which are protected by the firmware 70 and which are also not accessible to the hypervisor 30.

Following, the realization of the handling of interruptions or interceptions by means of firmware services as services implemented in the firmware 70 according to an embodiment of the invention is explained. In a short description, processing a guest event in a hypervisor-controlled system 10 is comprising the steps: (i) triggering a first firmware service; (ii) the firmware 70 processing information associated with the guest event, presenting only a subset of the information to a hypervisor 30; (iii) the firmware retaining a part of the information that is not being sent to the hypervisor 30; (iv) the hypervisor 30 triggering a second firmware service; (v) the firmware generating a guest state 52 and/or a protected guest memory 22 modification; (vi) the firmware performing the guest state 52 and/or protected guest memory 22 modification.

Enabling access for the hypervisor 30 to a memory page by the firmware 70 comprises (i) the hypervisor 30 requesting access to

a page of the protected guest memory 22 from firmware 70; (ii) the firmware 70 disabling access for the guest 20 to that page; (iii) the firmware 70 encrypting that page with the guest key 24; and (iv) the firmware 70 enabling access for the hypervisor 30 to that page.

Further enabling access for a guest 20 to a memory page by the firmware 70 comprises (i) the hypervisor 30 providing an encrypted page to the firmware 70; (ii) the firmware 70 disabling access for the hypervisor 30 to that page; (iii) the firmware 70 decrypting that page; and (iv) the firmware 70 enabling access for the guest 20 to that page.

Allowing decryption by the firmware 70 further comprises (i) the firmware 70 computing an integrity value of the page to be encrypted and made accessible to the hypervisor 30; (ii) the hypervisor 30 providing an encrypted page to be added to the protected pages of a virtual machine to the firmware 70; and (iii) the firmware 70 only allowing adding the decrypted page to the protected pages of the virtual machine when the integrity value matches the page.

A secure deployment and execution of a guest 20 comprises providing the guest 20 with the guest key 24 being encrypted with the public key 32 associated with a private key 26 of the hypervisor-controlled system 10 for transfer to the key store 28 of the hypervisor-controlled system 10; providing the hypervisor-controlled system 10 with the private key 26, being stored in the hypervisor-controlled system 10 and being used to decrypt the encrypted guest key 24; the guest key 24 being used to encrypt and decrypt the guest data, when being transferred out of or into the protected guest memory 22.

A boot image generation and deployment process covers generating a boot image by a client or customer for the guest 20;

encrypting the boot image with the guest key 24; transferring the encrypted boot image to a boot disk; loading the encrypted boot image of a guest 20 by the hypervisor 30 to the guest memory 23; transforming the guest memory 23 into the protected guest memory 22; decrypting contents of the protected guest memory 22; starting an execution of a guest 20 as a virtual machine.

An interaction between the guest 20 and the hypervisor 30 further covers a range of the protected guest memory 22 or registers associated with the guest event being not accessible to the hypervisor 30; extending a virtualization function 34 of the hypervisor-controlled system 10 by access methods to specific guest data associated with the guest event.

An I/O process of a guest 20 further comprises defining a non-protected memory area for I/O buffers and I/O control structures of the guest 20 outside the area of the protected guest memory 22; starting the I/O process by the guest 20 using that non-protected area of the guest memory 22; the virtualization function 34 of the hypervisor-controlled system 10 generating a guest event; the hypervisor 30 reading a reason for the guest event and performing the I/O process.

Further checking a guest integrity with a checking process, the checking process comprises the steps: the checking process knowing the guest key 24; the guest 20 reading a memory content in clear text from the protected guest memory 22 transferring an arbitrary range of the protected guest memory 22 via a secure communication path to the checking process; requesting the same memory range of the protected guest memory 22 from the hypervisor 30 and transferring it to the checking process; comparing the memory content obtained from the guest 20 with the result of decrypting the memory content obtained from the hypervisor 30; delivering a comparison result depending on the

contents of the two memory ranges; the result of the checking process being positive if the comparison result equals zero, otherwise being negative.

In Figure 4 a generic flow chart for executing a secure guest running according to an embodiment of the invention from a hypervisor 30 and CPU 216 view is shown. Figure 4 as well as Figure 5 reference in the flowcharts objects defined in the diagrams of Figures 2 and 3, so the reference numerals used also are referencing the objects of these Figures. Running a secure guest 20 starts with step S410, where the hypervisor 30 reads an encrypted guest image from an initial program loader (IPL) device together with an encrypted guest key 24. Next, in step S420, the hypervisor 30 stores the encrypted guest image into the protected guest memory 22. In step S430 the hypervisor 30 prepares an initial guest state 52 that includes the encrypted guest key 24 and submits it to the firmware 70. Thus the steps S410 to S430 serve for initializing a guest 20 in a hypervisor-controlled system 10. Next in step S440 the firmware 70 decrypts the protected guest memory 22 thus protecting the protected guest memory 22 from access by non-trusted components, as e.g. the hypervisor 30 or other guests. In step S450 a secure guest 20 virtual machine is started according to the current guest state 52. Following in step S460, a CPU 216 runs the secure guest 20 in a secure guest context 42 as described by the current guest state 52. In step S470, if a guest event in the form of an interrupt or an instruction interception occurs, the guest 20 exits the guest context 42 with an updated guest state 52 due to this interrupt or interception. The hypervisor 30 is now able in step S480, to handle the interrupt or interception using a first firmware service to read data from the secure guest 20 or a second firmware service to write data to the secure guest 20.

Generally, a first firmware service may be triggered, meaning

that (ii) the firmware 70 is processing information associated with the guest event and presenting only a subset of the information of the guest state 52 and the protected guest memory 22 to the hypervisor 30, and (iii) the firmware 70 is retaining a part of the information that is not being sent to the hypervisor 30. Further, (iv) based on the received subset of the information, the hypervisor 30 may be triggering a second firmware service for (v) generating a state and/or memory modification of the guest 20, and for (vi) performing the state and/or memory modification associated with the guest event at the protected guest memory 22.

If the secure guest 20 is finished, at branch S490 the whole process comes to an end. If the secure guest 20 is not finished, a loop to step S450 is closed and the hypervisor 30 is starting the secure guest 20 again.

In Figure 5 a detailed flow chart for interception handling with a secure guest running according to an embodiment of the invention is depicted. Step S510 starts with a guest event, meaning that the secure guest 20 is issuing an instruction which requests interpretation or support by the hypervisor 30, e.g., an instruction to store system environment parameters, which is usually provided by the hypervisor 30. In a next step S520, the execution of the virtualization function 34 leaves the guest context 42 and passes initiative to a CPU-internal virtualization logic. Then in step S530 the CPU-internal virtualization logic detects a reason for exit of the guest 20, e.g., identifies instruction to store system environment parameters. In step S540, the CPU-internal virtualization logic prepares handles for the hypervisor 30 to access input and output parameters, according to the reason for exit of the guest 20, e.g., associates the storage location for the requested information with a handle. Then in step S550, the CPU-internal virtualization logic masks the part of the execution state not

needed to process the interception and returns the initiative to the hypervisor 30, indicating the exit of the guest 20 and a hint to input and output parameter handles, e.g., hides registers and context data from the hypervisor 30, e.g. by encrypting them. The CPU-internal virtualization logic in steps S520 to S550 can alternatively be implemented as a first firmware service. Next, in step S560, the hypervisor 30 detects the reason for the exit of the guest 20 by reading the reason indication from the CPU-internal virtualization logic, e.g., reads a reason code to identify a virtual server's request to store system environment parameters. The hypervisor 30 triggers in step S570 (if necessary, repeatedly) firmware services (e.g. the first firmware service) to work with input and output parameters to process the exit of the guest 20. To perform this, the hypervisor 30 uses previously established handles as means to reference contents of the memory 230 and registers required for processing, e.g., the hypervisor 30 stores system environment parameters into the virtual server's memory through firmware services (e.g. the second firmware service) using the received handle.

By this way in the detailed flow chart for interception handling with a secure guest in Figure 5 it is described how first and second firmware services are used for processing a guest event in a hypervisor-controlled system 10, how (ii) the firmware 70 is processing information associated with the guest event and presenting only a subset of the information to the hypervisor 30, as well as (iii) the firmware 70 is retaining a part of the information and (iv) based on the received subset of the information the second firmware service may be triggered.

Next, in step S580, the hypervisor 30 restarts virtualization function 34 execution by issuing an according CPU instruction, until in step S590, the CPU-internal virtualization logic clears handles from the previous exit of the guest 20, unmasks the

virtual server context for the virtualization function 34 execution and starts execution of the virtual server.

Thus a state and/or memory modification of the guest 20 may be generated and performed at the guest memory 22 in encrypted form.

Referring now to Figure 6, a schematic of an example of a data processing system 210 is shown. Data processing system 210 is only one example of a suitable data processing system and is not intended to suggest any limitation as to the scope of use or functionality of embodiments of the invention described herein. Regardless, data processing system 210 is capable of being implemented and/or performing any of the functionality set forth herein above.

In data processing system 210 there is a computer system/server 212, which is operational with numerous other general purpose or special purpose computing system environments or configurations. Examples of well-known computing systems, environments, and/or configurations that may be suitable for use with computer system/server 212 include, but are not limited to, personal computer systems, server computer systems, thin clients, thick clients, handheld or laptop devices, multiprocessor systems, microprocessor-based systems, set top boxes, programmable consumer electronics, network PCs, minicomputer systems, mainframe computer systems, and distributed Cloud computing environments that include any of the above systems or devices, and the like.

Computer system/server 212 may be described in the general context of computer system executable instructions, such as program modules, being executed by a computer system. Generally, program modules may include routines, programs, objects, components, logic, data structures, and so on that perform

particular tasks or implement particular abstract data types. Computer system/server 212 may be practiced in distributed Cloud computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed Cloud computing environment, program modules may be located in both local and remote computer system storage media including memory storage devices.

As shown in Fig. 6, computer system/server 212 in data processing system 210 is shown in the form of a general-purpose computing device. The components of computer system/server 212 may include, but are not limited to, one or more processors or processing units 216, a system memory 228, and a bus 218 that couples various system components including system memory 228 to processor 216.

Bus 218 represents one or more of any of several types of bus structures, including a memory bus or memory controller, a peripheral bus, an accelerated graphics port, and a processor or local bus using any of a variety of bus architectures. By way of example, and not limitation, such architectures include Industry Standard Architecture (ISA) bus, Micro Channel Architecture (MCA) bus, Enhanced ISA (EISA) bus, Video Electronics Standards Association (VESA) local bus, and Peripheral Component Interconnect (PCI) bus.

Computer system/server 212 typically includes a variety of computer system readable media. Such media may be any available media that is accessible by computer system/server 212, and it includes both volatile and non-volatile media, removable and non-removable media.

System memory 228 can include computer system readable media in the form of volatile memory, such as random access memory (RAM) 230 and/or cache memory 232. Computer system/server 212 may

further include other removable/non-removable, volatile/non-volatile computer system storage media. By way of example only, storage system 234 can be provided for reading from and writing to a non-removable, non-volatile magnetic media (not shown and typically called a "hard drive"). Although not shown, a magnetic disk drive for reading from and writing to a removable, non-volatile magnetic disk (e.g., a "floppy disk"), and an optical disk drive for reading from or writing to a removable, non-volatile optical disk such as a CD-ROM, DVD-ROM or other optical media can be provided. In such instances, each can be connected to bus 218 by one or more data media interfaces. As will be further depicted and described below, memory 228 may include at least one program product having a set (e.g., at least one) of program modules that are configured to carry out the functions of embodiments of the invention.

Program/utility 240, having a set (at least one) of program modules 242, may be stored in memory 228 by way of example, and not limitation, as well as an operating system, one or more application programs, other program modules, and program data. Each of the operating system, one or more application programs, other program modules, and program data or some combination thereof, may include an implementation of a networking environment. Program modules 242 generally carry out the functions and/or methodologies of embodiments of the invention as described herein. Computer system/server 212 may also communicate with one or more external devices 214 such as a keyboard, a pointing device, a display 224, etc.; one or more devices that enable a user to interact with computer system/server 212; and/or any devices (e.g., network card, modem, etc.) that enable computer system/server 212 to communicate with one or more other computing devices. Such communication can occur via Input/Output (I/O) interfaces 222. Still yet, computer system/server 212 can communicate with one or more networks such as a local area network (LAN), a general

wide area network (WAN), and/or a public network (e.g., the Internet) via network adapter 220. As depicted, network adapter 220 communicates with the other components of computer system/server 212 via bus 218. It should be understood that although not shown, other hardware and/or software components could be used in conjunction with computer system/server 212. Examples, include, but are not limited to: microcode, device drivers, redundant processing units, external disk drive arrays, RAID systems, tape drives, and data archival storage systems, etc.

The block diagrams in the figures illustrate the architecture, functionality, and operation of possible implementations of systems, methods and computer program products according to various embodiments of the present invention. In this regard, each block in the block diagrams may represent a module, segment, or portion of code, which comprises one or more executable instructions for implementing the specified logical functions. It should also be noted that, in some alternative implementations, the functions noted in the block may occur out of the order noted in the figures. For example, two blocks shown in succession may, in fact, be executed substantially concurrently, or the blocks may sometimes be executed in the reverse order, depending upon the functionality involved. It will also be noted that each block of the block diagrams, and combinations of blocks in the block diagrams, can be implemented by special purpose hardware-based systems that perform the specified functions or acts, or combinations of special purpose hardware and computer instructions.

C L A I M S

1. Method for processing a guest event in a hypervisor-controlled system (10), comprising the steps:

(i) the guest event triggering a first firmware service being specific for the guest event in a firmware (70), the guest event being associated with a guest (20) and with a guest state (52) and a protected guest memory (22) accessible only by the guest (20) and the firmware (70), and a guest key (24);

(ii) the firmware (70) processing information associated with the guest event, comprising information of the guest state (52) and the protected guest memory (22), and presenting only a subset of the information of the guest state (52) and the protected guest memory (22) to a hypervisor (30), wherein the subset of the information is selected to suffice for the hypervisor (30) to process the guest event;

(iii) the firmware (70) retaining a part of the information of the guest state (52) and the protected guest memory (22) that is not being sent to the hypervisor (30);

(iv) the hypervisor (30) processing the guest event based on the received subset of the information of the guest state (52) and the protected guest memory (22) and sending a process result to the firmware (70) triggering a second firmware service being specific for the guest event;

(v) the firmware (70) processing the received process result together with the part of the information of the guest state (52) and the protected guest memory (22) that was not sent to the hypervisor (30), generating a state and/or memory modification;

(vi) the firmware (70) performing the state and/or memory modification associated with the guest event at the protected guest memory (22).

2. Method according to claim 1, further comprising the steps of

(i) the hypervisor (30) requesting access to a page of the protected guest memory (22) from firmware (70);

(ii) the firmware (70) disabling access for the guest (20) to that page;

(iii) the firmware (70) encrypting that page with the guest key (24);

(iv) the firmware (70) enabling access for the hypervisor (30) to that page.

3. Method according to claim 1 or 2, further comprising the steps of

(i) the hypervisor (30) providing an encrypted page to the firmware (70);

(ii) the firmware (70) disabling access for the hypervisor (30) to that page;

(iii) the firmware (70) decrypting that page;

(iv) the firmware (70) enabling access for the guest (20) to that page.

4. Method according to claim 2 or 3, further comprising the steps of

(i) the firmware (70) computing an integrity value of the page to be encrypted and made accessible to the hypervisor (30);

(ii) the hypervisor (30) providing an encrypted page to be added to the protected pages of a virtual machine to the firmware (70);

(iii) the firmware (70) only allowing adding the decrypted page to the protected pages of the virtual machine when the integrity value matches the page.

5. Method according to any one of the preceding claims, further comprising the steps of

(i) providing the guest (20) with the guest key (24) being encrypted with a public key associated with a private key (26) of the hypervisor-controlled system (10) for transfer to the key store (28) of the hypervisor-controlled system (10);

(ii) providing the hypervisor-controlled system (10) with the private key (26), being stored in the hypervisor-controlled system (10) and being used to decrypt the encrypted guest key (24);

(iii) the guest key (24) being used to encrypt and decrypt the guest data, when being transferred out of or into the protected guest memory (22).

6. Method according to any one of the preceding claims, further comprising the steps of

(i) generating a boot image by a client;

(ii) encrypting the boot image with the guest key (24);

(iii) transferring the encrypted boot image to a boot disk;

(iv) loading the encrypted boot image of a guest (20) by the hypervisor (30) to the guest memory (23);

(v) transforming the guest memory (23) into the protected guest memory (22);

(vi) decrypting contents of the protected guest memory (22);

(vii) starting an execution of a guest (20) as a virtual machine.

7. Method according to any one of the preceding claims, further comprising the steps of

(i) keeping the range of the protected guest memory (22) or registers associated with the guest event not accessible to the hypervisor (30) in decrypted form;

(ii) extending a virtualization function of the hypervisor-controlled system (10) by access methods to specific guest data associated with the guest event.

8. Method according to any one of the preceding claims, further comprising the steps of

(i) defining a non-protected memory area for I/O buffers and I/O control structures of the guest (20) outside the area of the protected guest memory (22);

(ii) starting the I/O process by the guest (20) using that non-protected area of the guest memory (22);

(iii) the virtualization function of the hypervisor-controlled system (10) generating a guest event;

(iv) the hypervisor (30) reading a reason for the guest event and performing the I/O process.

9. Method according to any one of the preceding claims, further checking a guest integrity with a checking process that knows the guest key (24), the checking process comprising the steps:

(i) the guest (20) reading a memory content in clear text from the protected guest memory (22) transferring an arbitrary range of the protected guest memory (22) via a secure communication path to the checking process;

(ii) requesting the same memory range of the protected guest memory (22) from the hypervisor (30) and transferring it to the checking process;

(iii) comparing the memory content obtained from the guest (20) with the result of decrypting the memory content obtained from the hypervisor (30);

(iv) delivering a comparison result depending on the contents of the two memory ranges;

(v) returning a result of the checking process being positive if the comparison result equals zero, otherwise being negative.

10. A data processing program (240) for execution in a data processing system (210) comprising an implementation of an instruction set for performing a method according to anyone of the claims 1 to 9 when the data processing program (240) is run on a computer (212).

11. A computer program product comprising a computer usable medium including a computer readable program, wherein the computer readable program when executed on a computer (212) causes the computer (212) to perform a method for processing a guest event in a hypervisor-controlled system, comprising the steps:

(i) the guest event triggering a first firmware service being specific for the guest event in a firmware (70), the guest event being associated with a guest (20) and with guest state (52) and a protected guest memory (22) accessible only by the guest (20) and the firmware (70), and a guest key (24);

(ii) the firmware (70) processing information associated with the guest event, comprising information of the guest state (52) and the protected guest memory (22), and presenting only a subset of the information of the guest state (52) and the protected guest memory (22) to a hypervisor (30), wherein the subset of the information is selected to suffice for the hypervisor (30) to process the guest event;

(iii) the firmware (70) retaining a part of the information of the guest state (52) and the protected guest

memory (22) that is not being sent to the hypervisor (30);

(iv) the hypervisor (30) processing the guest event based on the received subset of the information of the guest state (52) and the protected guest memory (22) and sending a process result to the firmware (70) triggering a second firmware service being specific for the guest event;

(v) the firmware (70) processing the received process result together with the part of the information of the guest state (52) and the protected guest memory (22) that was not sent to the hypervisor (30), generating a state and/or memory modification;

(vi) the firmware (70) performing the state and/or memory modification associated with the guest event at the protected guest memory (22).

12. A data processing system (210) for execution of a data processing program (240) comprising software code portions for performing a method according to anyone of the claims 1 to 9.

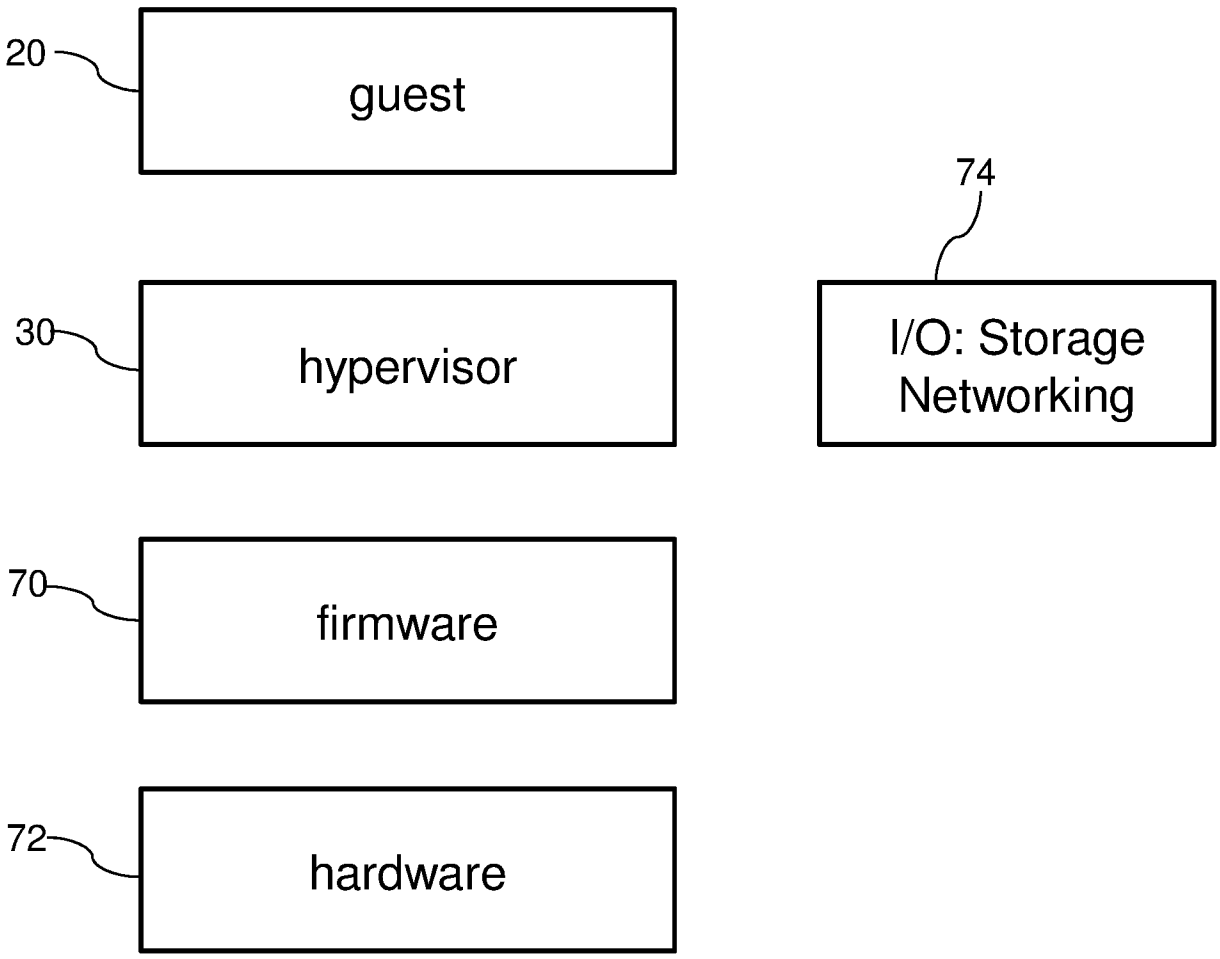

Fig. 1 (Prior Art)

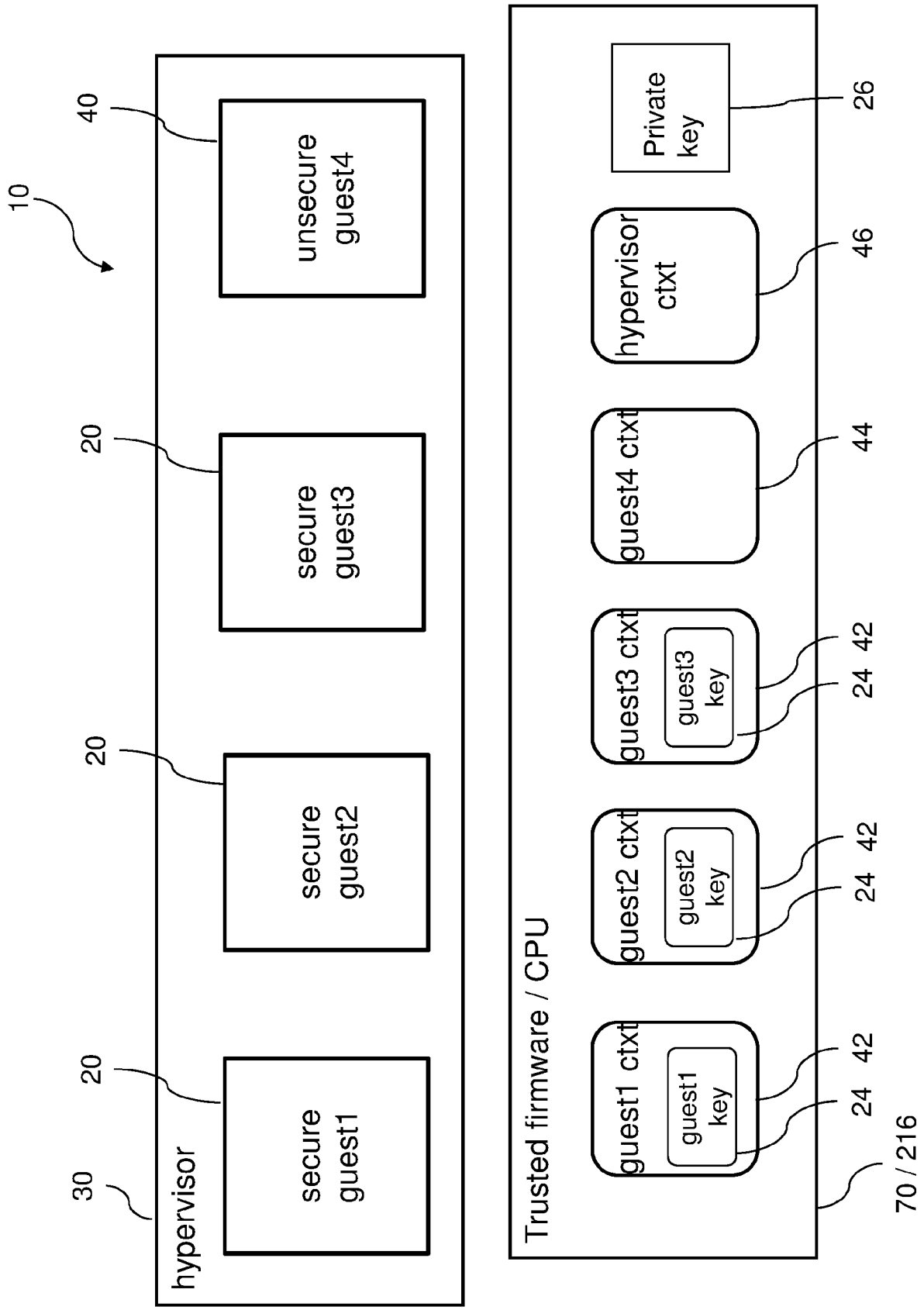
10
40
unsecure guest4
20
secure guest3
20
secure guest2
20
secure guest1
30
hypervisor
Trusted firmware / CPU
guest1 ctxt
guest1 key
guest2 ctxt
guest2 key
guest3 ctxt
guest3 key
guest4 ctxt
hypervisor ctxt
Private key
24 42
24 42
24 42
44
46
26
70 / 216

Fig. 2

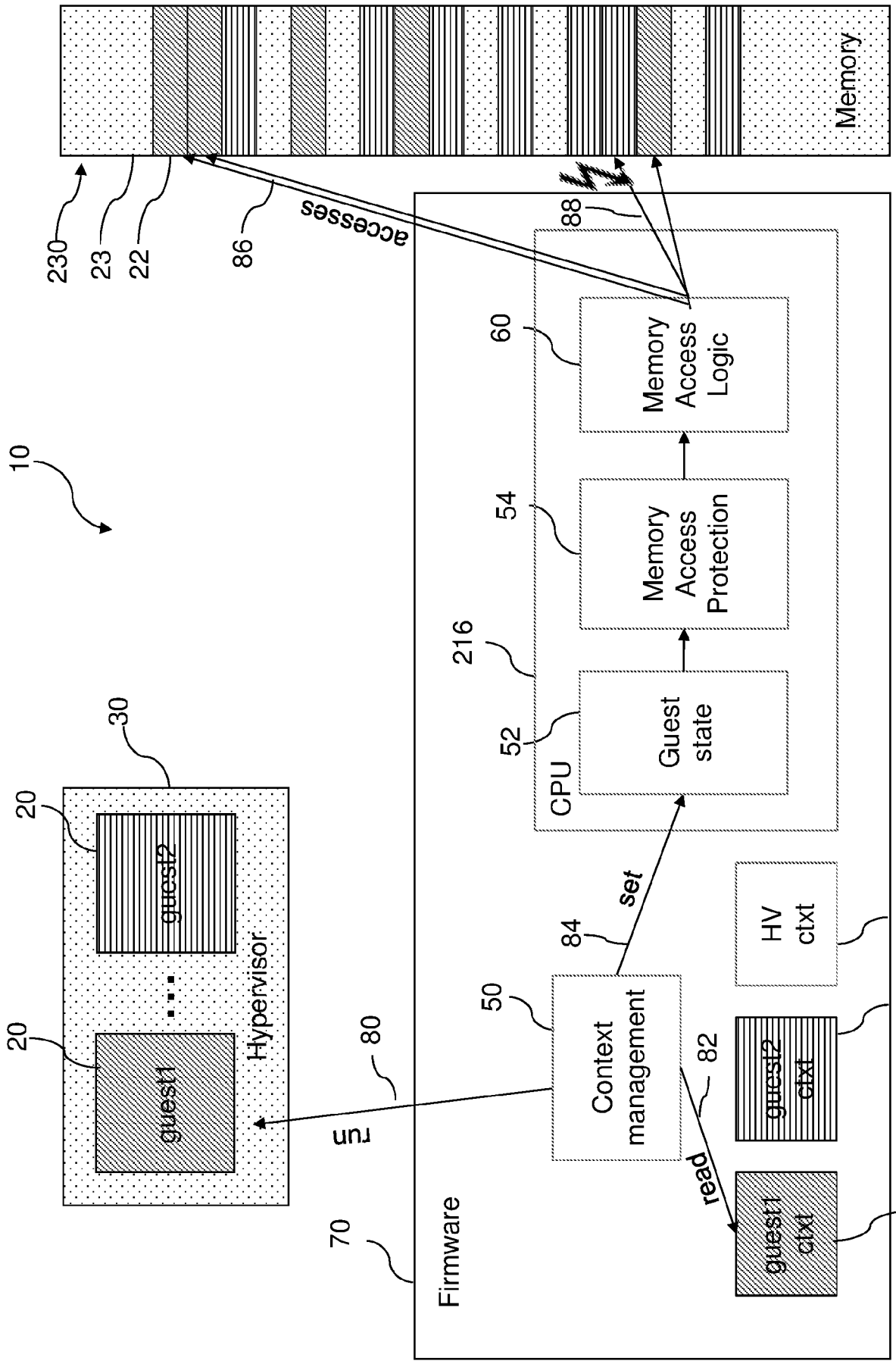
Memory
230
23
22
86
accesses
88
10
60
Memory Access Logic
54
Memory Access Protection
216
52
Guest state
CPU
30
20
guest2
20
guest1
Hypervisor
80
run
84
set
50
Context management
82
read
HV ctxt
46
guest2 ctxt
42
guest1 ctxt
42
70
Firmware

Fig. 3

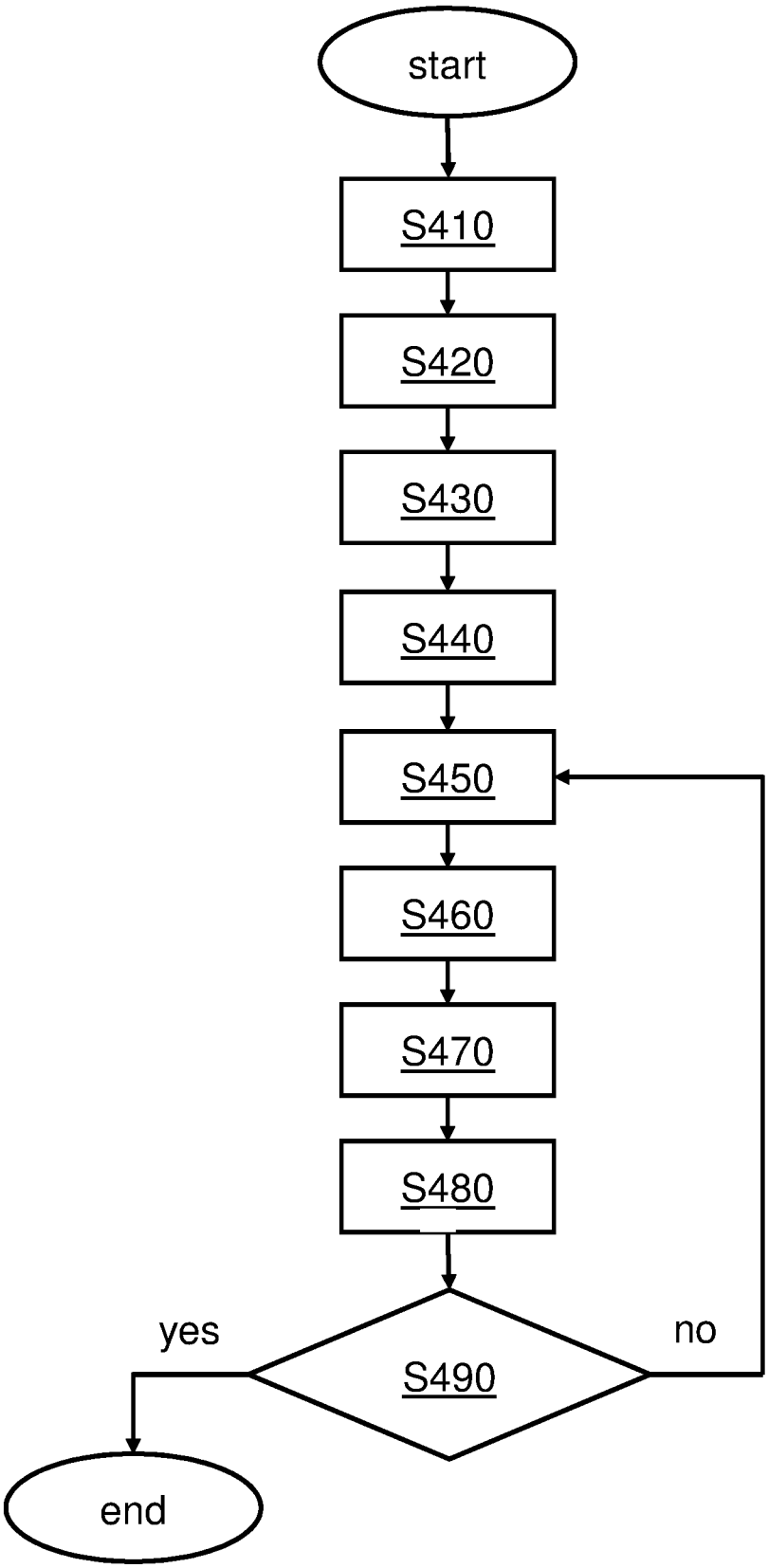

Fig. 4

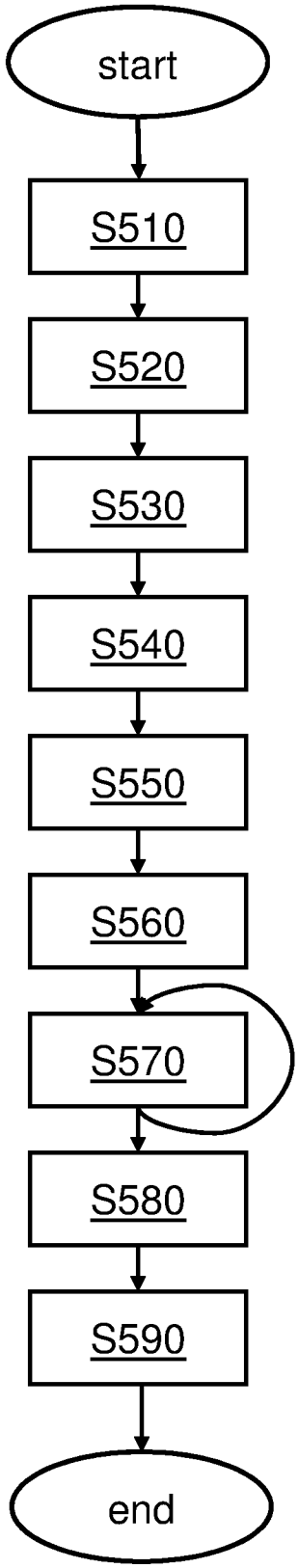

Fig. 5

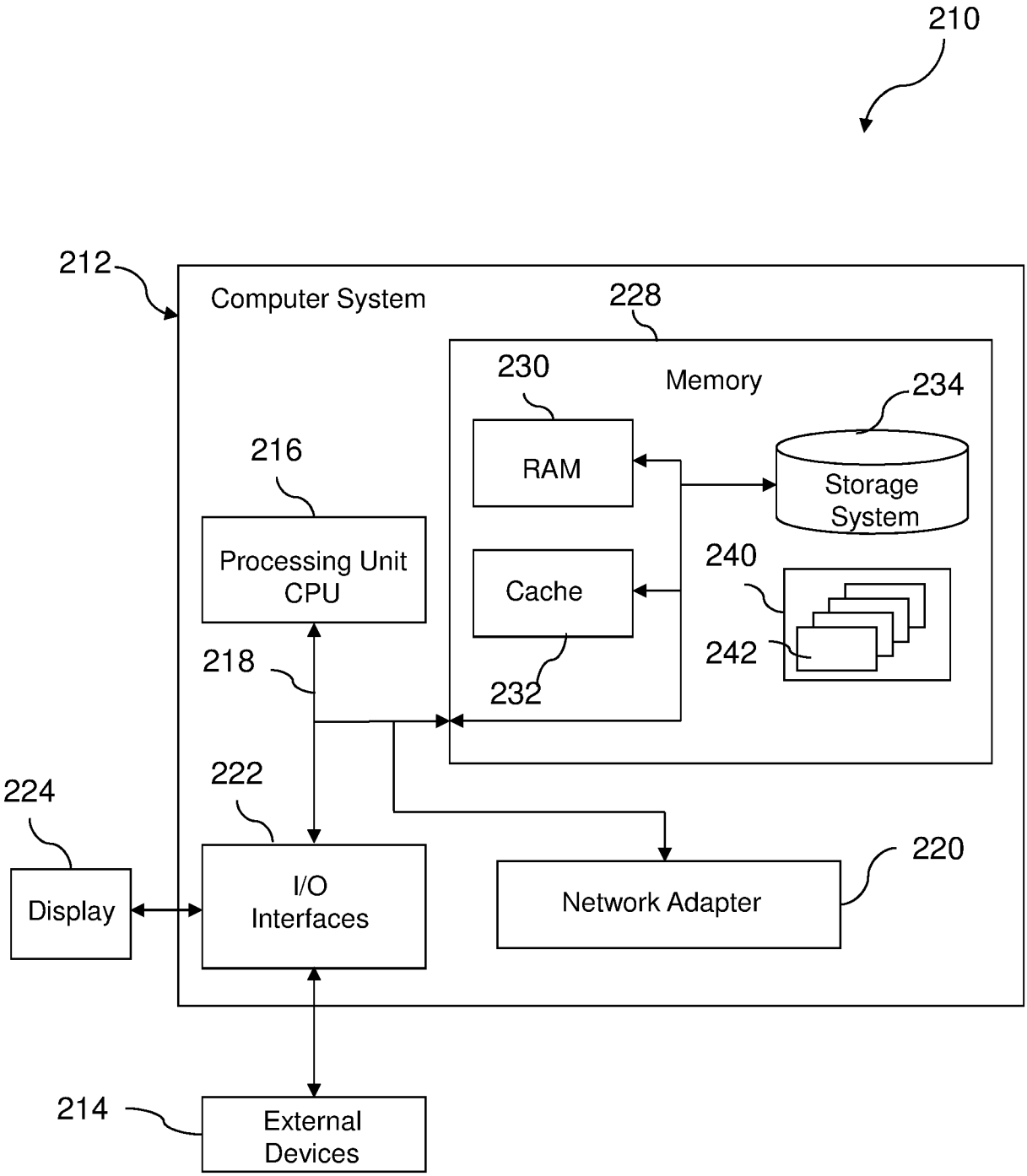
210
212
Computer System
228
230
Memory
234
RAM
Storage System
216
Processing Unit CPU
Cache
240
242
218
232
222
224
220
Display
I/O Interfaces
Network Adapter
214
External Devices

Fig. 6

INTERNATIONAL SEARCH REPORT

International application No.
PCT/IB2015/058058

A. CLASSIFICATION OF SUBJECT MATTER

G06F 9/44(2006.01)i

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

EPODOC, WPI, CNPAT, CNKI, IEEE, GOOGLE: hypervisor, virtual machine, cloud?, secur+, safety, privacy, indication, client, guest?, user?, firmware, service, key+, event, encrypt+

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 8479190 B2 (SONY CORPORATION) 02 July 2013 (2013-07-02) description, column 2, line 50 to column 5, line 15	1-12
A	CN 102420692 A (GCI SCI&TECHNOLOGY CO., LTD.) 18 April 2012 (2012-04-18) the whole document	1-12
A	CN 103403732 A (HUAWEI TECHNOLOGIES CO., LTD.) 20 November 2013 (2013-11-20) the whole document	1-12
PX	GB 2515536 A (INTERNATIONAL BUSINESS MACHINES CORPORATION) 31 December 2014 (2014-12-31) description, pages 11-16, and figures 1-6	1-12

☐ Further documents are listed in the continuation of Box C. ☑ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
15 January 2016	**14 February 2016**
Name and mailing address of the ISA/CN **STATE INTELLECTUAL PROPERTY OFFICE OF THE P.R.CHINA** **6, Xitucheng Rd., Jimen Bridge, Haidian District, Beijing 100088, China** Facsimile No. **(86-10)62019451**	Authorized officer **WU,Qing** Telephone No. **(86-10)61648111**

INTERNATIONAL SEARCH REPORT
Information on patent family members

International application No.
PCT/IB2015/058058

Patent document cited in search report			Publication date (day/month/year)	Patent family member(s)			Publication date (day/month/year)
US	8479190	B2	02 July 2013	CN	101231595	A	30 July 2008
				JP	2008181228	A	07 August 2008
CN	102420692	A	18 April 2012		None		
CN	103403732	A	20 November 2013	WO	2014059575	A1	24 April 2014
GB	2515536	A	31 December 2014	WO	2014207581	A2	31 December 2014
				DE	112014000965	T5	03 December 2015