

- (51) International Patent Classification:
G06F 9/50 (2006.01) G06F 12/00 (2006.01)
- (21) International Application Number:
PCT/US2020/044307
- (22) International Filing Date:
30 July 2020 (30.07.2020)
- (25) Filing Language: English
- (26) Publication Language: English
- (71) Applicant: HEWLETT-PACKARD DEVELOPMENT COMPANY, L.P. [US/US]; 10300 Energy Drive, Spring, Texas 77389 (US).
- (72) Inventors: PINHEIRO, Endrigo, Nadin; 10300 Energy Drive, Spring, Texas 77389 (US). SCHIFFMAN, Joshua, Serratelli; 1 Redcliff Street, Long Down Ave, Stoke Gifford, Bristol BS1 6NP (GB). BRAMLEY, Richard Alden, Jr.; 12 Gloria Drive, Mansfield, Massachusetts 02048 (US). ALI, Valiuddin; 10300 Energy Drive, Spring, Texas 77389 (US).
- (74) Agent: SU, Benjamin et al.; HP Inc., 3390 E. Harmony Road, Mail Stop 35, Fort Collins, Colorado 80528 (US).

- (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, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, IT, JO, JP, KE, KG, KH, KN, KP, KR, KW, 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, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, WS, 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, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

(54) Title: BIOS ACTION REQUEST FOR AUTHORIZED APPLICATION

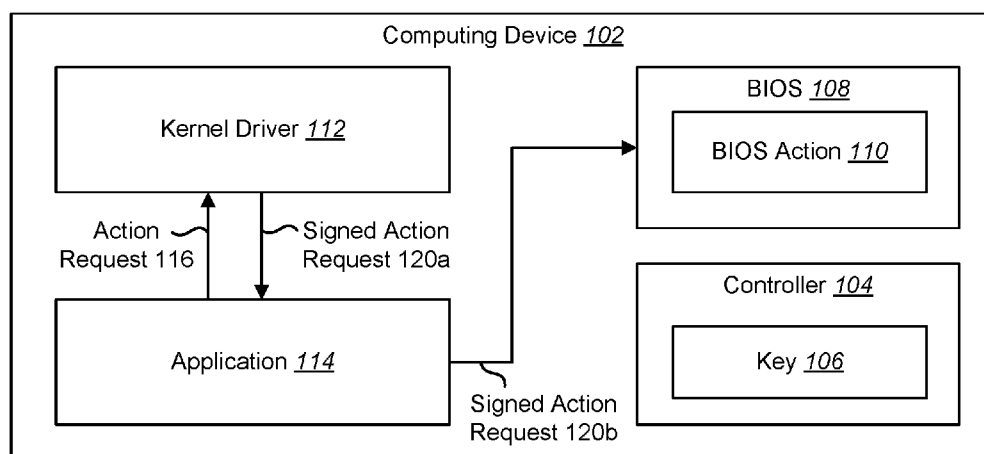


FIG. 1

(57) Abstract: Examples of computing devices are described herein. In some examples, a computing device may include a controller to generate a key upon boot of the computing device. In some examples, the computing device may include a kernel driver. In some examples, the kernel driver may be to receive the key from a basic input/output system (BIOS) during operating system (OS) boot. In some examples, the kernel driver may be to receive an action request for a BIOS action from an application. In some examples, the kernel driver may be to sign the action request with the key in response to determining that the application is authorized to request the BIOS action. In some examples, the computing device may include the BIOS to perform the BIOS action in response to receiving the signed action request.

Declarations under Rule 4.17:

- *as to the identity of the inventor (Rule 4.17(i))*
- *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))*

Published:

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

BIOS ACTION REQUEST FOR AUTHORIZED APPLICATION

BACKGROUND

[0001] Electronic technology has advanced to become virtually ubiquitous in society and has been used to improve many activities in society. For example, computing devices are used to perform a variety of tasks, including work activities, communication, research, and entertainment. Different varieties of electronic circuits may be utilized to provide different varieties of electronic technology.

BRIEF DESCRIPTION OF THE DRAWINGS

[0002] Various examples will be described below by referring to the following figures.

[0003] FIG. 1 is a block diagram illustrating an example of a computing device that may be utilized for a Basic Input/Output System (BIOS) action request by an authorized application;

[0004] FIG. 2 is a block diagram illustrating another example of a computing device that may be utilized for a BIOS action request by an authorized application;

[0005] FIG. 3 is a block diagram illustrating yet another example of a computing device that may be utilized for a BIOS action request by an authorized application;

[0006] FIG. 4 is a block diagram illustrating yet another example of a computing device that may be utilized for a BIOS action request by an authorized application;

[0007] FIG. 5 is a block diagram illustrating an example of a computer-readable medium for a BIOS action request by an authorized application; and

[0008] FIG. 6 is a flow diagram illustrating an example of a method for requesting a BIOS action by an authorized application.

[0009] Throughout the drawings, identical or similar reference numbers may designate similar, but not necessarily identical, elements. The figures are not necessarily to scale, and the size of some parts may be exaggerated to more clearly illustrate the example shown. Moreover, the drawings provide examples in accordance with the description; however, the description is not limited to the examples provided in the drawings.

DETAILED DESCRIPTION

[0010] A computing device may be a device that includes electronic circuitry. For instance, a computing device may include integrated circuitry (e.g., transistors, digital logic, semiconductor technology, etc.). Examples of computing devices include laptop computers, desktop computers, smartphones, tablet devices, wireless communication devices, game consoles, smart appliances, vehicles with electronic components, aircraft, drones, robots, smart appliances, etc.

[0011] In some examples of the computing devices described herein, a computing device may include a Basic Input/Output System (BIOS). As used herein, a basic input/output system (BIOS) refers to hardware or hardware and instructions to initialize, control, or operate a computing device prior to execution of an operating system (OS) of the computing device. Instructions included within a BIOS may be software, firmware, microcode, or other programming that defines or controls functionality or operation of a BIOS. In one example, a BIOS may be implemented using instructions, such as platform firmware of a computing device, executable by a processor. A BIOS may operate or execute prior to the execution of the OS of a computing device. A BIOS may initialize, control, or operate components such as hardware components of a computing device and may load or boot the OS of computing device.

[0012] In some examples, a BIOS may provide or establish an interface between hardware devices or platform firmware of the computing device and an OS of the computing device, via which the OS of the computing device may control or operate hardware devices or platform firmware of the computing device. In some examples, a BIOS may implement the Unified Extensible Firmware Interface (UEFI) specification or another specification or standard for initializing, controlling, or operating a computing device.

[0013] The BIOS may perform an action (referred to herein as a BIOS action) in response to a request sent from an application of the computing device. In some examples, a BIOS action may include changing settings (referred to as BIOS settings) that are accessible to applications of the computing device. These BIOS settings may include settings for the BIOS and/or settings for the BIOS interface (e.g., UEFI). Examples are described herein for secure access to the BIOS settings by an authorized application at runtime. In other examples, a BIOS action may include use of privileged BIOS capabilities, functions, resources, or other operations that the BIOS may perform in response to the application's request. Therefore, a BIOS action may include an operation that is to be performed by the BIOS in response to a request sent by an application.

[0014] In the examples described herein, the BIOS action may be performed with or without pre-knowledge of the BIOS administrative (Admin) password. Instead, the BIOS may cryptographically detect and verify a calling application.

[0015] In some examples, a kernel driver of the OS may be coupled to the BIOS through cryptography using keys generated at boot (e.g., start up) of the computing device. As used herein, a kernel driver may be a program that operates in kernel mode on the computing device.

[0016] In some examples, the kernel driver may be an early boot (e.g., boot-start) driver that is closely coupled to the BIOS through cryptography. The keys may be generated by a secure controller or by the BIOS itself. In some examples, the keys may be cryptographic keys or transport keys. In some examples, the keys may be symmetric or asymmetric. In the case of asymmetric keys, the public key may be provided to the BIOS and the kernel driver may

hold the private key. The keys may be shared with the kernel driver during OS boot using a secure key exchange protocol using the BIOS as an intermediary.

[0017] After obtaining the cryptographic key from the BIOS, the kernel driver may expose interfaces for authorized applications wanting to send a request to the BIOS to perform a BIOS action (e.g., make changes to the BIOS settings, and/or use privileged BIOS capabilities, functions, resources, etc.). An application may first pass an action request to the kernel driver. In some examples, the action request may include setting information and a new value for the BIOS setting. In other examples, the kernel driver may then verify if the calling application is an authorized application using various techniques (e.g., digital signatures or other OS-level constructs). The kernel driver may then authorize the action request by signing the action request with the shared key provided by the BIOS.

[0018] While examples that include a kernel driver are described herein, in other examples, an early-launch mechanism may obtain the key from the BIOS and may sign the action request from the calling application. As used herein, an early-launch mechanism may include executable code (e.g., a driver, process, service, etc.) that launches at an early stage during an OS boot. The early-launch mechanism may also be referred to as early-launch code or early-boot code. In some examples, the early-launch mechanism may operate in kernel mode. An example of the early-launch mechanism includes an Early-Launch AntiMalware service. In some examples, an early-launch mechanism may perform the operations of the kernel driver described herein.

[0019] In some examples, the signed action request may be returned back to the calling application, which then sends the newly signed action request using BIOS access methods provided by an operating system (e.g., Windows Management Instrumentation (WMI) for WINDOWS for example). In this case, existing BIOS interfaces may be unmodified, thus keeping compatibility with the legacy tools. In some examples, the signed action request may be communicated to the BIOS using an authenticated communication method. For example, an authenticated communication may include symmetric methods (e.g., hash-based message authentication code (HMAC), Authenticated

Encryption with Associated Data (AEAD) encryption, Advanced Encryption Standard (AES)-Galois/Counter Mode (GCM)) or asymmetric methods (e.g., signatures or integrated encryption schemes using Diffie Hellman (DH) and/or Elliptic-curve Diffie–Hellman (ECDH)).

[0020] Once received, the BIOS may detect the presence of the signature for the action request. The BIOS may then independently, or by using the controller that generated the key, verify the signature on the action request. If the signature check passes, the BIOS recognizes the calling application as an authorized application and performs the requested BIOS action.

[0021] This process of secure BIOS access may be performed by the BIOS even if a BIOS administrative password has been previously set and/or is not provided as part of the action request. In other examples, a policy decision may be made to use the BIOS administrative password for some or all operations. These examples also provide protection against man-in-the middle (MiTM) attacks on OS-BIOS access interfaces (e.g., WMI). In some examples, the payload passed to the kernel driver in the action request may also be encrypted, providing confidentiality in addition to integrity protection.

[0022] FIG. 1 is a block diagram illustrating an example of a computing device 102 that may be utilized for a BIOS action request 116 by an authorized application 114. Examples of the computing device 102 may include computing devices, laptop computers, desktop computers, smartphones, tablet devices, wireless communication devices, game consoles, vehicles with electronic components, aircraft, drones, robots, smart appliances, etc.

[0023] In some examples of access schemes (e.g., WMI) for the BIOS 108, there is no standardized way for the BIOS 108 to know the identity of a calling process. Therefore, in these examples, the BIOS 108 does not know if the calling application 114 is an authorized process or not. It should be noted that in some cases a BIOS action 110 (e.g., reading a BIOS setting) may be accessed without the calling process possessing administrative rights in the OS. In other cases a BIOS action 110 (e.g., reading a BIOS setting) may rely on administrative rights (e.g., by using a Private WMI interface and a namespace that is protected for access by administrative processes). However, in these

cases, this does not mean that the calling process is an authorized one. For example, a malicious process with administrative access may also access the BIOS action 110. Therefore, ensuring that BIOS security policies and configurations are accessed by authorized processes is a challenge for computing devices.

[0024] A malicious actor (e.g., malware) that has access to BIOS settings may take over the computing device 102 with little effort. For example, a malicious actor can replace the serial number of a PC. In another example, a malicious actor can adjust the security settings to a more relaxed policy (e.g., by allowing previously prevented devices such as cameras, WiFi cards, Microphone, USB ports, etc.). The malicious actor can then use these exploits to mount new attacks. The malicious actor could also change settings that could prevent the operating system from starting securely (e.g., turning off Secure Boot), may disable boot time power on password, may disable virtualization or Trusted Platform Module (TPM), and so on.

[0025] As seen by this discussion, some computing device architectures do not offer a method for the BIOS 108 to determine whether a trusted process is trying to access the BIOS configuration or whether a malicious actor gained undue privileges and is calling to alter security configuration of the BIOS 108. The examples described herein provide mechanisms by which the BIOS 108 may grant access to a BIOS action 110 to authorized processes while blocking access to other (unauthorized) processes.

[0026] In some examples, the computing device 102 may include a controller 104. In some examples, the controller 104 may be a microcontroller (e.g., embedded controller) that performs tasks on the computing device 102. In some examples, the controller 104 may perform tasks that the OS does not perform. In some examples, the controller 104 may include memory (e.g., RAM, ROM) that is independent of memory used by the main OS of the computing device 102.

[0027] The controller 104 may generate a key 106 upon boot of the computing device 102. At boot time (e.g., upon start up or power up of the computing device 102), and on every boot, the controller 104 may generate a

cryptographic transport key 106. In some examples, the key 106 may be a symmetric key or may include an asymmetric key pair (e.g., a private and public key pair). The controller 104 may securely store the key 106. For example, the controller 104 may store the key in a storage (e.g., memory) that is exclusively accessible to the controller 104. In other examples, the controller 104 may also encrypt the key 106 (or key pair) with another key embedded inside the controller 104 that is exclusively accessible to the controller 104.

[0028] In some examples, the controller 104 may send the key 106 to the BIOS 108 upon execution of the BIOS 108. For example, the controller 104 may pass the key 106 to the BIOS when BIOS starts executing. In some examples, the controller 104 may pass the key 106 through a private channel or protocol. If the BIOS 108 does extract the key 106 from the controller 104 and stores the key 106, the BIOS 108 may extract the key 106 as early as possible during a BIOS power-on self-test (POST). The BIOS 108 may ensure that the key 106 is protected in the most secure memory available to the BIOS 108 (e.g., System Management Mode (SMM) memory).

[0029] In the example of FIG. 1, the controller 104 may store the key 106 and may verify a signed action request 120b signed with the key 106 as described below. It should be noted that in other examples, the BIOS 108 may generate the key 106. An example of this approach is described in FIG. 3.

[0030] The BIOS 108 may create or accommodate for an interface that can allow a kernel driver 112 to request the key 106. In some examples, the kernel driver 112 may be an OS-based early boot driver. In some examples, the interface may include an Advanced Configuration and Power Interface (ACPI) method call, a WMI call, or other method that allows an OS-BIOS interface.

[0031] During early stages of the OS boot process (e.g., before most third-party code is loaded), the kernel driver 112 may be started. The kernel driver 112 may also be referred to as a kernel boot-start driver, a boot driver or an early-boot driver. There are various examples for how the kernel driver 112 may be installed or injected in the OS. These examples include a regular mode of code installation, using a Windows Platform Binary Table (WPBT), injecting the kernel driver 112 in a guest OS using a Hypervisor, etc. It should be noted that a

high level of security may be achieved by the kernel driver 112 being an early boot driver to ensure that the kernel driver 112 is one of the very first set of trusted processes an operating system loads. For example, the key 106 may be extracted from the BIOS 108 as early as possible in a manner that prevents untrusted or un-authorized applications from acquiring the key 106 before the kernel driver 112. This key exchange may be implemented based on desired security guarantees and cryptographic key procedures (e.g., key agreement, key wrapping, and direct transfer of key material).

[0032] In some examples, with a BIOS 108 and OS that use UEFI Secure Boot, the BIOS 108 may ensure that the OS the BIOS 108 is booting is cryptographically secure. The BIOS 108 may also ensure that the OS loader and the other early parts of the OS ensure that the integrity of the components (e.g., the kernel driver 112) the BIOS 108 is loading is secure. Other OS-specific advance security technologies like Hypervisor Code Integrity (HVCI) may provide additional assurances where the integrity and the functionality of the kernel driver 112 is ensured even more by use of a hardware assisted Hypervisor. This ensures that the kernel driver 112 will be one of the earliest code modules that run and has a high probability of obtaining the key 106 before other processes.

[0033] The kernel driver 112 may receive the key 106 from the BIOS 108 during OS boot. When loaded, the kernel driver 112 may make a call directly to the BIOS 108 to request the cryptographic key 106 (e.g., transport key) generated by the controller 104. In some examples, the BIOS 108 may request the key 106 from the controller 104 in response to receiving the request for the key 106 from the kernel driver 112. The BIOS 108 may pass the key 106 to the kernel driver 112 in response to receiving the key 106 from the controller 104.

[0034] In some examples, the BIOS 108 may share the generated key 106 with the kernel driver 112 during OS boot using a secure key exchange protocol. For example, the kernel driver 112 may make an ACPI method call to the BIOS 108 to request the key 106. It should be noted that the ACPI method call may provide additional security against attackers as applications and/or unprivileged code (e.g., ring 3 code) would not be able to obtain the key 106. Furthermore,

with the kernel driver 112 being an early-boot driver, the chances of malware implemented as a kernel driver to obtain the key 106 are significantly reduced.

[0035] The ACPI method call may allow the kernel driver 112 to extract the key 106 from a BIOS-provided memory location. In some examples, the extraction of the key 106 may be a “Get” type operation. In other examples, the extraction of the key 106 may employ a complex key exchange protocol. In another example, the BIOS 108 may ask the controller 104 to provide the BIOS 108 the key 106 when asked. The BIOS 108 may then pass the key 106 over to the kernel driver 112.

[0036] In some examples, the kernel driver 112 that calls the BIOS 108 may also ensure that the key 106 is erased from the temporary memory slot where the BIOS 108 placed the key 106, preventing another driver that loads after the kernel driver 112 from also accessing the key 106. Therefore, the BIOS 108 may erase memory storing the key 106 to prevent additional reads to the key 106 in response to sending the key 106 to the kernel driver 112. In some examples, the BIOS 108 may ensure that after kernel driver 112 extracts the key 106, the BIOS 108 no longer honors a call (e.g., ACPI method call) for key extraction, thus preventing other rogue entities from extracting the key 106.

[0037] The kernel driver 112 may receive an action request 116 from an application 114 for a BIOS action 110. For example, the kernel driver 112 may expose an interface for OS processes to call in and pass data (e.g., Binary Large Object (BLOB)) to the kernel driver 112. In some examples, a legitimate (e.g., authorized) application 114 that seeks access to a protected BIOS action 110 may call the kernel driver 112 using the driver-exposed interface. The application 114 may pass instructions (or a set of instructions) that is to be performed by the BIOS 108 in an action request 116.

[0038] The kernel driver 112 may perform authorization of the action request 116 so that the BIOS 108 can trust the action request 116 and perform the BIOS action 110. In some examples, the kernel driver 112 may first verify that the calling application 114 is an authorized application in response to receiving the action request 116. In an example, the kernel driver 112 may use a whitelist that includes applications authorized to request the BIOS action 110. In another

example, the kernel driver 112 may verify the on-disk identity of the calling application 114 by verifying the digital signature of the application 114 and ensuring the calling application 114 is a known and/or authorized process. In yet another example, the kernel driver 112 may ensure that the location of the calling application 114 is controlled via a strong access-control list (ACL), thereby preventing other applications from placing rogue code (e.g., a rogue Dynamic-link library (DLL)) next to a legitimate application 114, thus causing unintended code injection. In other examples, the OS may be configured with other pieces of technology (e.g., anti-malware, anti-virus, next-gen anti-virus, endpoint detection and response (EDR) tools, or other tools that can prevent process hallowing to ensure that the caller application 114 has not been overtaken by an imposter.

[0039] The kernel driver 112 may sign the action request 116 with the key 106 in response to determining that the application 114 is authorized to request the BIOS action 110 to generate a signed action request 120a. For example, if the calling application 114 is verified to be legitimate and/or authorized by the kernel driver 112 to request changes to the BIOS 108, the kernel driver 112 may sign the incoming action request 116 with the key 106 that the kernel driver 112 acquired during boot. In some examples, the action request 116 may be signed by the kernel driver 112 with the key 106 using a digital signature, hash-based message authentication code (HMAC), or an authenticated encryption (e.g., AES-GCM) scheme. In some examples, the kernel driver 112 may also add additional metadata to the signed action request 120a that adds protections (e.g., integrity, confidentiality, and/or replay-protection) using the key 106.

[0040] In the example of FIG. 1, the kernel driver 112 may return the signed action request 120a to the application 114. In this case, the application 114 sends the signed action request 120b to the BIOS 108. For example, once the calling application 114 receives the signed action request 120a from the kernel driver 112, the application 114 can then use the available methods (e.g., WMI methods) for passing the signed action request 120b to the BIOS 108.

[0041] In other examples, the kernel driver 112 may send the signed action request 120a to the BIOS 108 to perform the BIOS action 110. In this case,

instead of sending the signed action request 120a back to the application 114, the kernel driver 112 may send the signed action request 120a directly to the BIOS 108. This may be accomplished as described in FIG. 2.

[0042] Upon receiving the signed action request 120b, the BIOS 108 may detect the signed action request 120a is signed. If the BIOS 108 detects the signed action request 120b, then the BIOS 108 may perform a signature check to verify the signed action request 120b. In some examples, the BIOS 108 may authenticate the action request 116 using the key 106. In some examples where the key 106 was generated and stored by the controller 104, the BIOS 108 may delegate the task of validating the integrity of the signed action request 120b (and/or removing the confidentiality envelope) to the controller 104 using a transport protocol available on the computing device 102.

[0043] If the signature check passes (e.g., is successful), then the BIOS 108 may process the signed action request 120b. For example, the BIOS 108 may change a BIOS setting or perform an operation according to the signed action request 120a in response to determining that the signature check passes. In some examples, the BIOS 108 may return the result (e.g., success or failure) of the signed action request 120b to the calling application 114.

[0044] If the BIOS 108 finds that an incoming action request was not properly protected or authenticated with the key 106 (e.g., if the signature check fails, an action request does not include the key 106), then the BIOS 108 may refuse to act on the action request. In some examples, the BIOS 108 may treat this as an un-authorized attempt to change a BIOS setting by a rogue application. In some examples where a BIOS administrative access password is set, the BIOS 108 may treat this failed signature check as a password failure situation where multiple failures lock out further requests, even if the password provided in the action request was correct.

[0045] These examples provide controlled access to a BIOS action 110, preventing malicious or un-authorized processes to change or to access BIOS configurations. The access to the BIOS action 110 may be prevented even if the malicious or un-authorized processes somehow obtained access to the BIOS administrative password (if one is set).

[0046] It should be noted that while the above examples were described using a BIOS action 110, these examples also apply to other situations where the BIOS 108 is to trust any incoming data or commands. Similarly, while Windows and x86 terminology was used, these examples also apply to other OS and computer architectures.

[0047] It should also be noted that these examples may add several levels of protection and flexibility for performing a BIOS action 110. For example, the BIOS 108 may use the signed action request 120a described herein to provide access to sensitive BIOS settings while allowing and honoring other action requests to come in un-signed. In other examples, the BIOS 108 may allow Get requests to be honored without signing but may expect some or all Set requests to be authorized. Again, these examples may be extended to allow for a flexible security model for accessing a BIOS action 110.

[0048] The described examples provide authenticity, integrity, and confidentiality for calls to the BIOS 108 (e.g., WMI calls). Enforcement of access control policies for the calls (e.g., WMI calls) may be enforced in the kernel driver 112. Once the BIOS 108 knows that the calling application 114 is authorized, the BIOS 108 may then perform the BIOS action 110 immediately without depending on authorization from a physically present user, thus simplifying user experiences while increasing security. The described examples may also eliminate potential MiTM attacks between the BIOS 108 and calling applications for data returned from BIOS.

[0049] Using a kernel driver 112 that is an early boot-start driver to retrieve the key 106 from the BIOS 108 may reduce the attack surface to a millisecond-scale timeframe. This is because the kernel driver 112 is one of the first processes that starts during the OS boot process. After the first request to get the key 106, the BIOS 108 may erase the key 106 from the temporary memory of the BIOS 108, which prevents any other requests from retrieving the key 106, ensuring that the kernel driver 112, and not other processes, will receive the key 106.

[0050] Furthermore, because of the kernel driver 112 may run in a privileged state (e.g., privilege level 0, or ring 0), no other user mode process can read or

replace the memory structures of the kernel driver 112. Therefore, the key 106 is protected against attacks from user mode processes. In some examples, the kernel driver 112 may rely on the OS to prevent non-signed drivers to be installed in the kernel mode. This means that the kernel driver 112 may be protected from being replaced or corrupted by another kernel driver.

[0051] In some examples, the computing device 102 may include additional components and/or some of the components described herein may be removed and/or modified without departing from the scope of this disclosure. In some examples, the computing device 102 and/or the controller 104 may include a processor and/or a memory. The processor may be any of a central processing unit (CPU), a semiconductor-based microprocessor, graphics processing unit (GPU), field-programmable gate array (FPGA), an application-specific integrated circuit (ASIC), and/or other hardware device suitable for retrieval and execution of instructions stored in the memory. The processor may fetch, decode, and/or execute instructions stored in the memory. The memory may be any electronic, magnetic, optical, or other physical storage device that contains or stores electronic information (e.g., instructions and/or data). Thus, the memory may be, for example, Random Access Memory (RAM), Electrically Erasable Programmable Read-Only Memory (EEPROM), Dynamic Random Access Memory (DRAM), magnetoresistive random-access memory (MRAM), phase change RAM (PCRAM), memristor, flash memory, a storage device, and/or an optical disc, etc. In some examples, the memory may be a non-transitory tangible computer-readable storage medium, where the term “non-transitory” does not encompass transitory propagating signals. The processor may be in electronic communication with the memory. In some examples, a processor and/or memory of the controller 104 may be separate from an applications processor (e.g., CPU) of the computing device 102.

[0052] In some examples, the computing device 102 may include an input/output interface (not shown) through which the computing device 102 may communicate with an external device or devices (not shown). The input/output interface may include hardware and/or computer-readable instructions to enable the processor to communicate with the external device or devices. The

input/output interface may enable a wired or wireless connection to the external device or devices. In some examples, the input/output interface may further include a network interface card and/or may also include hardware and/or computer-readable instructions to enable the processor to communicate with various input and/or output devices, such as a keyboard, a mouse, a display, another apparatus, computing device, electronic device, etc., through which a user may input instructions and/or indications into the computing device 102.

[0053] FIG. 2 is a block diagram illustrating another example of a computing device 202 that may be utilized for a BIOS action request 216 by an authorized application 214. The computing device 202 may be implemented in accordance with the computing device 102 described in FIG. 1. For example, the controller 204 may generate a key 206. This key 206 may be provided to the kernel driver 212 via the BIOS 208 as described in FIG. 1.

[0054] The kernel driver 212 may receive an action request 216 from an application 214 for a BIOS action 210. The kernel driver 212 may perform authorization of the action request 216 so that the BIOS 208 can trust the action request 216 and perform the BIOS action 210. The kernel driver 212 may sign the action request 216 with the key 206 in response to determining that the application 214 is authorized to request the BIOS action 210. This may be accomplished as described in FIG. 1.

[0055] In the example of FIG. 2, the kernel driver 212 may send the signed action request 220 to the BIOS 208 to perform the BIOS action 210. In this case, instead of sending the signed action request 220 back to the application 214, the kernel driver 212 may send the signed action request 220 directly to the BIOS 208.

[0056] Upon receiving the signed action request 220, the BIOS 208 may verify the signed action request 220 based in the key 206. This may be accomplished as described in FIG. 1. If the signed action request 220 is verified, the BIOS 208 may perform the BIOS action 210.

[0057] FIG. 3 is a block diagram illustrating yet another example of a computing device 302 that may be utilized for a BIOS action request 316 by an

authorized application 314. The computing device 302 may be implemented in accordance with the computing device 102 described in FIG. 1.

[0058] In this example, the BIOS 308 may generate the key 306 used to sign an action request 316 sent by an application 314. For example, upon boot of the computing device 302, the BIOS 308 may generate the key 306. The BIOS 308 may store the key 306 in a secure internal structure (e.g., a UEFI variable, SMM or private storage on shared or private flash memory).

[0059] The BIOS 308 may create an interface to allow the kernel driver 312 to request the key during OS boot. This may be accomplished as described in FIG. 1. However, in this example, the BIOS 308 may retain the key 306 in storage for use in verifying a signed action request 320.

[0060] In this example, a processor 311 of the computing device 302 may implement the kernel driver 312 and the calling application 314. For example, the kernel driver 312 and the calling application 314 may be implemented as code executed by the processor 311. In some examples, the processor 311 may be any of a CPU, a semiconductor-based microprocessor, GPU, FPGA, an ASIC, and/or other hardware device suitable for retrieval and execution of instructions stored in memory. The kernel driver 312 may use the key 306 to sign an action request 316 in response to determining that the application 314 is authorized to request the BIOS action 310. This may be accomplished as described in FIG. 1.

[0061] The processor 311 may send the signed action request 320 to the BIOS 308. In an example, the application 314 may send the signed action request 320 to the BIOS 308, as described in FIG. 1. In another example, the kernel driver 312 may send the signed action request 320 to the BIOS 308, as described in FIG. 2.

[0062] Upon receiving the signed action request 320, the BIOS 308 may perform a signature check to verify the signed action request 320. In this case, the BIOS 308 may validate the integrity of the signed action request 320 (and/or remove the confidentiality envelope) using the stored key 306. The BIOS 308 may perform the BIOS action 310 in response to determining that the signature check passes.

[0063] FIG. 4 is a block diagram illustrating yet another example of a computing device 402 that may be utilized for a BIOS action request by an authorized application. At the booting of the computing device 402, the controller 404 may generate a key 406. In some examples, the key 406 may be a transport key. In some examples, the controller 404 may generate an asymmetric (e.g., public-private) key pair using RSA, ECC, or other asymmetric cryptographic procedure. In other examples, the controller 404 may generate a symmetric key from a random number generator (RNG) of suitable security.

[0064] The controller 404 may securely store the key 406 (or key pair) into a secure storage accessible exclusively by the controller 404. In some examples, this secure storage may be a private spy chip accessible exclusively by the controller 404 by hardware design. In another example, the secure storage may be integrated flash storage inside the controller 404. In some examples, if the key 406 is unique and accessible exclusively by the controller 404, the controller 404 may use that key 406 to further protect the confidentiality of a stored key pair to prevent un-authorized snooping of the secrets through offline attacks.

[0065] The controller 404 may send the key 406 to the BIOS 408. In the case of an asymmetric key pair, the controller 404 may send the public part of the key (e.g., the public key) to the BIOS 408. For a symmetric key, the controller 404 may send the entire key 406 to the BIOS 408.

[0066] The BIOS 408 may store the key 406. For example, the BIOS 408 may store the key 406 in an internal structure such as a UEFI variable, SMM or private storage on a shared (or private) flash. The BIOS 408 may hand control to the operating system, which then starts to boot.

[0067] During the operating system boot stages, the very basic (e.g., core) OS components start to load and boot. Next, early-launch drivers may load followed by early boot drivers (also referred to as boot-start drivers or boot drivers) may start to load. The kernel driver 412 may be started as part of the kernel 418 of the OS. It should be noted that a specific boot time is not integral to this example, but the kernel driver 412 may be started before any “untrusted” code can run.

[0068] In some examples, the kernel driver 412 may be an Early Launch of Anti Malware (ELAM)-type driver, a boot driver, or a boot driver that is launched and/or loaded by ELAM. These types of drivers may be trusted code that launch before any untrusted code runs.

[0069] The kernel driver 412 may request the key 406 from the BIOS 408. In response to this request, the BIOS 408 may return the key 406 to the kernel driver 412. The BIOS 408 may then erase the memory structure storing the key 406 for the BIOS 408 to prevent additional reads to the key 406. This may prevent other malicious programs from accessing the key 406 from the BIOS 408, which would allow the malicious programs to then become a malicious middleware driver. By ensuring that the kernel driver 412 is an early-boot driver, a large set of malicious malware is eliminated because the OS ensures that boot drivers are signed by legitimate organizations. For example, ELAM uses an Extended Validation (EV) certificate and provides a high bar for organizations to meet before an EV certificate is issued. It should be noted that the controller 404 in this example may retain the key 406 in secure storage.

[0070] The kernel driver 412 may store the key 406 internally in memory. In some examples, an operating system may ensure that accessing internal memory belonging to early-boot drivers is restricted. However, the kernel driver 412 may also use additional techniques to enhance protection of memory holding the key. For example, the kernel driver 412 may use enclave technologies provided by hardware and/or chipset schemes or via a hypervisor, etc. After the kernel driver 412 completes initialization, the rest of the operating system may continue to boot normally.

[0071] At a later stage, an authorized application 414 may start. The application 414 may be a process executed by the OS. In some examples, the application 414 may request a specific BIOS action 410. The kernel driver 412 may verify that the application is authorized to request the BIOS action 410.

[0072] In some examples, the application 414 may include code signed by a key pair that is trustworthy to the BIOS 408. For example, the public key belonging to the signing key may be embedded in the BIOS secure memory.

The name of the application 414 and/or the signer company name may also be ingrained in the BIOS secure memory.

[0073] In some examples, a whitelist may be defined for the kernel driver 412 to use to authenticate applications 414. The whitelist may include authorized processes and/or code signing certificate authorities (CAs) and corresponding WMIs that are allowed to be called by the application 414. The whitelist may be hardcoded in the kernel driver 412 or provided as a boot-time argument. In some examples, the granularity of the whitelist may be at the WMI level or even parameter level.

[0074] The application 414 may open a handle to the kernel driver 412 in preparation for sending an action request 416. The kernel driver 412 may determine the full path to the file of the application 414 attempting to get the handle from the kernel driver 412. The kernel driver 412 may verify the signature on the file. If the signature fails to verify for any reason (e.g., no certificate, no signature, bad signature, etc.), then the kernel driver 412 may fail the request for a handle.

[0075] In some examples, the kernel driver 412 may check the publisher information against a predefined list of valid publishers. If the publisher is not on the predefined list, then the kernel driver 412 may fail the request for a handle.

[0076] In some examples, the kernel driver 412 may verify that the original filename specified in the resources of the handle request matches the filename of the application 414. This may prevent swapping of other binaries from the same publisher. If the filenames are different, then the kernel driver 412 may fail the request for a handle.

[0077] If the kernel driver 412 verifies that the application 414 is authorized to request the BIOS action 410, then the application 414 may proceed to build the action request 416 to be sent to the BIOS 408 as a data buffer (e.g., a BLOB used in WMI). The application 414 may then send the action request 416 to the kernel driver 412 asking the kernel driver 412 to sign the action request 416. For example, the kernel driver 412 may expose an interface allowing the application 414 (and other drivers) to call.

[0078] The kernel driver 412 may then protect the action request with the key 406 to generate a signed action request 420a. In some examples, the kernel driver 412 may add anti-replay or targeting information to the signed action request 420a. Protection of the signed action request 420a may include authentication, encryption, and integrity protection. For symmetric keys, the signed action request 420a may be encoded using HMAC or encrypted (using AES-GCM, for instance). For asymmetric keys, the signed action request 420a may be protected using a symmetric session key. Depending on the procedure, the session secret may be derived through a key agreement or a key encapsulation method between the kernel driver 412 and the BIOS 408 or controller 404.

[0079] The kernel driver 412 may return the signed action request 420a back to the application 414. With the protected data returned from the kernel driver 412, the application 414 may call the BIOS 408 (e.g., using a WMI) and may send the signed action request 420b to the BIOS 408.

[0080] Upon receiving the signed action request 420b, the BIOS 408 may send the signed action request 420c to the controller 404 to verify the signed action request 420c using the key 406. The controller 404 may also process any anti-replay or targeting mechanisms that were included in the signed action request 420c. The targeting mechanisms may allow for unique identification of the computing device 402. The anti-replay mechanisms may restrict the scope of command execution on the computing device 402, such as a time duration during which the command can be executed, the number of times a command can execute on the given computing device 402, etc.

[0081] The controller 404 may decrypt the data in the signed action request 420c (if needed). The controller 404 may verify the data integrity and authenticity of the signed action request 420c using the key 406. The controller 404 may return the resulting data back to the BIOS 408 in a verification response 424. If the data verification fails, then the controller 404 may return an error in the verification response 424.

[0082] If the controller 404 returns a successfully verified action request, then the BIOS 408 may analyze the signed action request 420c to determine what

action to perform. The BIOS 408 may then perform the BIOS action 410 based on the signed action request 420c.

[0083] In some examples, the BIOS 408 may return a response 426 to the calling application 414. For example, if the signed action request 420b passes the signature check, then the BIOS 408 may return the response 426 as a success response. Otherwise, the BIOS 408 may return the response 426 as a failure response.

[0084] FIG. 5 is a block diagram illustrating an example of a computer-readable medium 530 for a BIOS action request by an authorized application. The computer-readable medium 530 may be a non-transitory, tangible computer-readable medium 530. The computer-readable medium 530 may be, for example, RAM, EEPROM, a storage device, an optical disc, and the like. In some examples, the computer-readable medium 530 may be volatile and/or non-volatile memory, such as DRAM, EEPROM, MRAM, PCRAM, memristor, flash memory, and the like. In some examples, the computer-readable medium 530 described in FIG. 5 may be an example of memory for a computing device 102 described in FIG. 1. In some examples, code (e.g., data and/or executable code or instructions) of the computer-readable medium 530 may be transferred and/or loaded to memory or memories of the computing device 102.

[0085] The computer-readable medium 530 may include code (e.g., data and/or executable code or instructions). For example, the computer-readable medium 530 may include key request instructions 532, action request reception instructions 534, and/or action request signing instructions 536.

[0086] In some examples, the key request instructions 532 may be instructions that when executed cause a processor of the computing device to send a request from a kernel driver during OS boot to a BIOS for a key. The key may be generated by a controller or the BIOS upon boot of the computing device. In some examples, this may be accomplished as described in FIGS. 1–4.

[0087] In some examples, the action request reception instructions 534 may be instructions that when executed cause the processor of the computing device to receive an action request for a BIOS action from an application. For

example, the processor may expose an interface for receiving the action request for the BIOS action from the application. In some examples, this may be accomplished as described in FIGS. 1–4.

[0088] In some examples, the action request signing instructions 536 may be instructions that when executed cause the processor of the computing device to sign the action request with the key in response to determining that the application is authorized to request the BIOS action. For example, the processor may verify that the application is an authorized application in response to receiving the action request. The processor may authorize the action request by signing the action request with the key. The signed action request may allow the BIOS to perform the BIOS action. In some examples, this may be accomplished as described in FIGS. 1–4.

[0089] FIG. 6 is a flow diagram illustrating an example of a method 600 for requesting a BIOS action by an authorized application. The method 600 and/or an element or elements of the method 600 may be performed by a computing device. For example, an element or elements of the method 600 may be performed by the computing device 102 described in FIG. 1, the controller 104 described in FIG. 1, the computing device 202 described in FIG. 2, the computing device 302 described in FIG. 3, the computing device 402 and/or the controller 404 described in FIG. 4, any of which may be referred to generally as a “computing device” in FIG. 6. In some examples, the method 600 may be performed using an instruction or instructions described in FIG. 5.

[0090] A computing device may generate 602 a key upon boot of the computing device. In some examples, a controller or the BIOS may generate 602 the key when the computing device powers on as described in FIGS. 1–4.

[0091] The computing device may send 604 the key from the BIOS to a kernel driver during OS boot. For example, the BIOS may create an interface to allow the kernel driver to request the key. The BIOS may receive a request for the key from the kernel driver. The BIOS may then pass the key to the kernel driver. The BIOS may share the generated key with the kernel driver during OS boot using a secure key exchange protocol. In some examples, the BIOS may

erase memory storing the key to prevent additional reads to the key in response to sending the key to the kernel driver.

[0092] The computing device (e.g., the kernel driver) may receive 606 an action request from an application for a BIOS action. For example, the kernel driver may expose an interface for receiving the action request. The application may send the action request using the interface provided by the kernel driver.

[0093] The computing device (e.g., the kernel driver) may determine 608 that the application is authorized to request to the BIOS action. For example, the kernel driver may use a whitelist of authorized applications to authenticate the calling application. In other examples, the kernel driver may verify a signature of the calling application. In yet other examples, the kernel driver may verify that the original filename specified in the resources of the handle request matches the filename of the application.

[0094] The computing device (e.g., the kernel driver) may sign 610 the action request with the key. For example, the kernel driver may protect the action request with the key to generate a signed action request.

[0095] The computing device may send 612 the signed action request to the BIOS. In some examples, the kernel driver may send the signed action request to the calling application, which forwards the signed action request to the BIOS. In other examples, the kernel driver may send the signed action request to the BIOS directly.

[0096] The computing device (e.g., the BIOS) may perform 614 a signature check to verify the signed action request. For example, the BIOS may send the signed action request to the controller to verify the data integrity and authenticity of the signed action request using the key. In other examples, the BIOS may verify the signature of the signed action request. The computing device (e.g., the BIOS) may perform 616 the BIOS action in response to determining that the signature check passes.

[0097] As used herein, the term “and/or” may mean an item or items. For example, the phrase “A, B, and/or C” may mean any of: A (without B and C), B (without A and C), C (without A and B), A and B (but not C), B and C (but not A), A and C (but not B), or all of A, B, and C.

[0098] While various examples are described herein, the disclosure is not limited to the examples. Variations of the examples described herein may be within the scope of the disclosure. For example, operations, functions, aspects, or elements of the examples described herein may be omitted or combined.

CLAIMS

1. A computing device, comprising:
a controller to generate a key upon boot of the computing device;
a kernel driver to:
 receive the key from a basic input/output system (BIOS) during
 operating system (OS) boot;
 receive an action request for a BIOS action from an application;
 and
 sign the action request with the key in response to determining
 that the application is authorized to request the BIOS
 action; and
the BIOS to perform the BIOS action in response to receiving the signed
action request.
2. The computing device of claim 1, wherein the controller is to store the
key and verify the signed action request.
3. The computing device of claim 1, wherein the controller is to send the key
to the BIOS upon execution of the BIOS.
4. The computing device of claim 1, wherein the BIOS is to share the key
with the kernel driver during OS boot using a secure key exchange protocol.
5. The computing device of claim 1, wherein the BIOS is to erase memory
storing the key to prevent additional reads to the key in response to sending the
key to the kernel driver.
6. The computing device of claim 1, wherein the BIOS is to:
 receive a request for the key from the kernel driver;
 request the key from the controller in response to the request for the key
 from the kernel driver; and

pass the key to the kernel driver in response to receiving the key from the controller.

7. The computing device of claim 1, wherein the kernel driver is to return the signed action request to the application, and wherein the application sends the signed action request to the BIOS.

8. The computing device of claim 1, wherein the kernel driver is to send the signed action request to the BIOS to perform the BIOS action.

9. A computing device, comprising:

a basic input/output system (BIOS) to generate a key upon boot of the computing device;

a kernel driver to:

receive the key from the BIOS during operating system (OS) boot;

receive an action request for a BIOS action from an application;

and

sign the action request with the key in response to determining

that the application is authorized to request the BIOS

action; and

the BIOS to perform the BIOS action in response to receiving the signed action request.

10. The computing device of claim 9, wherein the BIOS is to create an interface to allow the kernel driver to request the key.

11. The computing device of claim 9, wherein the BIOS is to:

detect the signed action request; and

perform a signature check to verify the signed action request.

12. The computing device of claim 11, wherein the BIOS is to perform the BIOS action in response to determining that the signature check passes.

13. A non-transitory tangible computer-readable medium comprising instructions when executed cause a processor of a computing device to:
- send a request from a kernel driver during operating system (OS) boot to a basic input/output system (BIOS) for a key, the key being generated by a controller or the BIOS upon boot of the computing device;
 - receive an action request for a BIOS action from an application; and
 - sign the action request with the key in response to determining that the application is authorized to request the BIOS action, the signed action request to allow the BIOS to perform the BIOS action.
14. The computer-readable medium of claim 13, wherein the instructions when executed further cause the processor to:
- verify that the application is an authorized application in response to receiving the action request; and
 - authorize the action request by signing the action request with the key.
15. The computer-readable medium of claim 13, wherein the instructions when executed further cause the processor to expose an interface for receiving the action request for the BIOS action from the application.

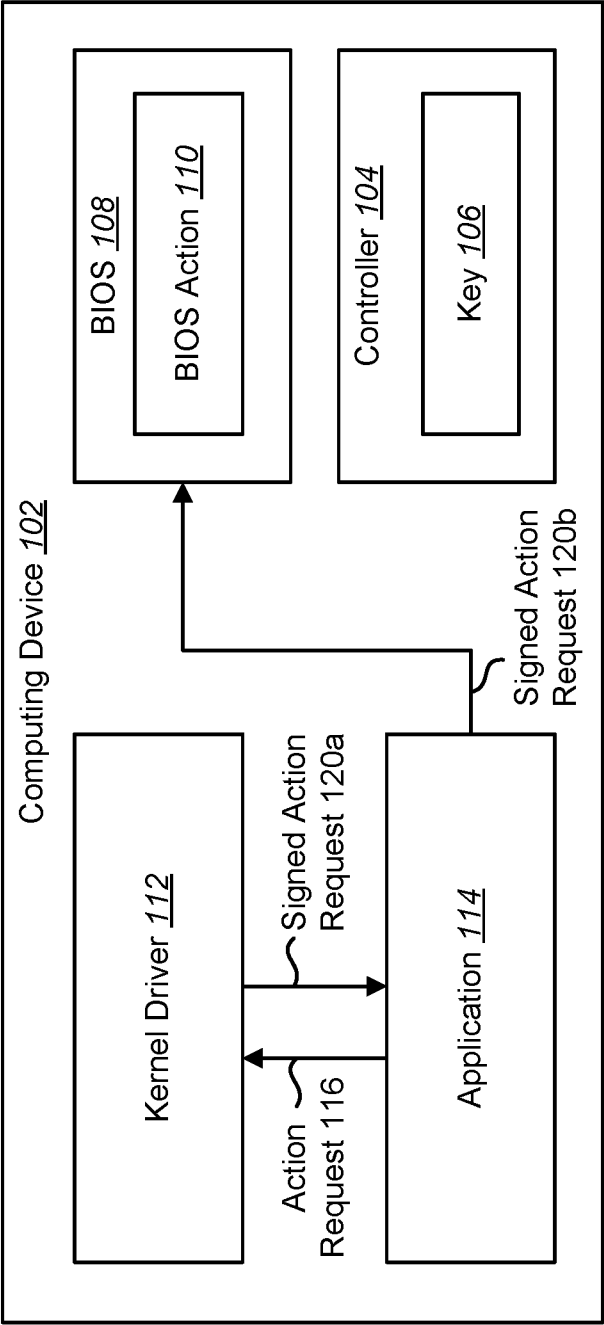


FIG. 1

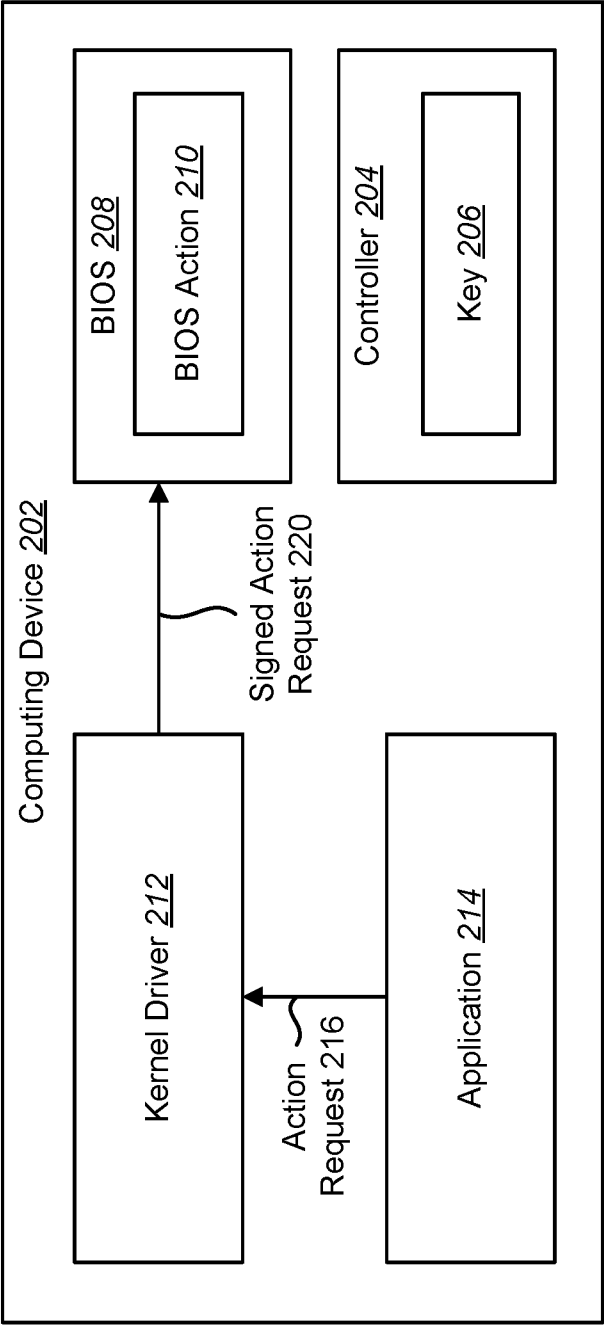


FIG. 2

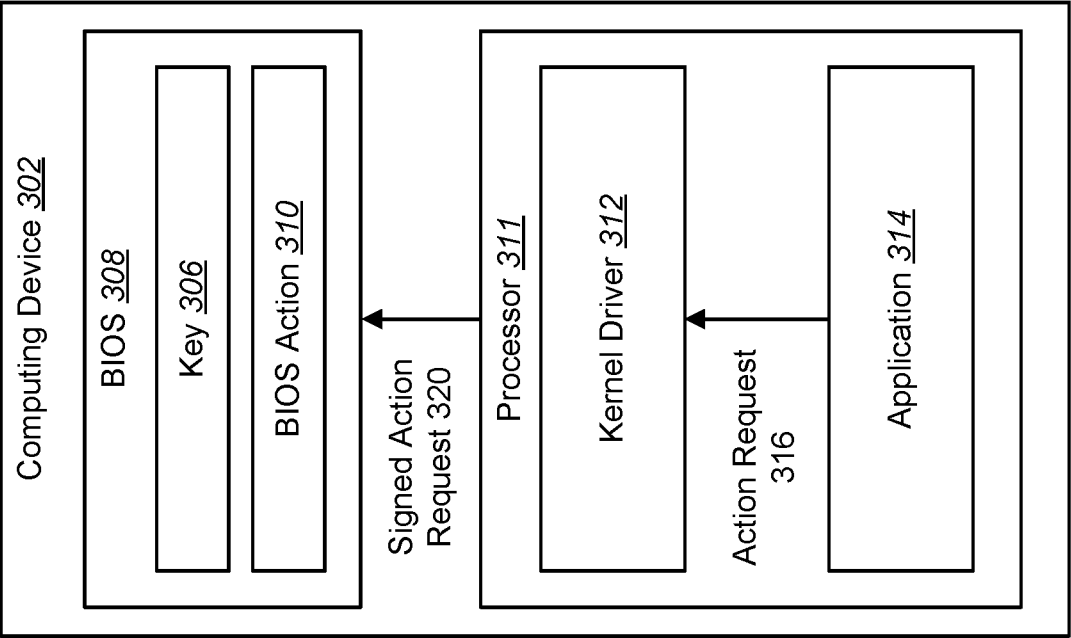


FIG. 3

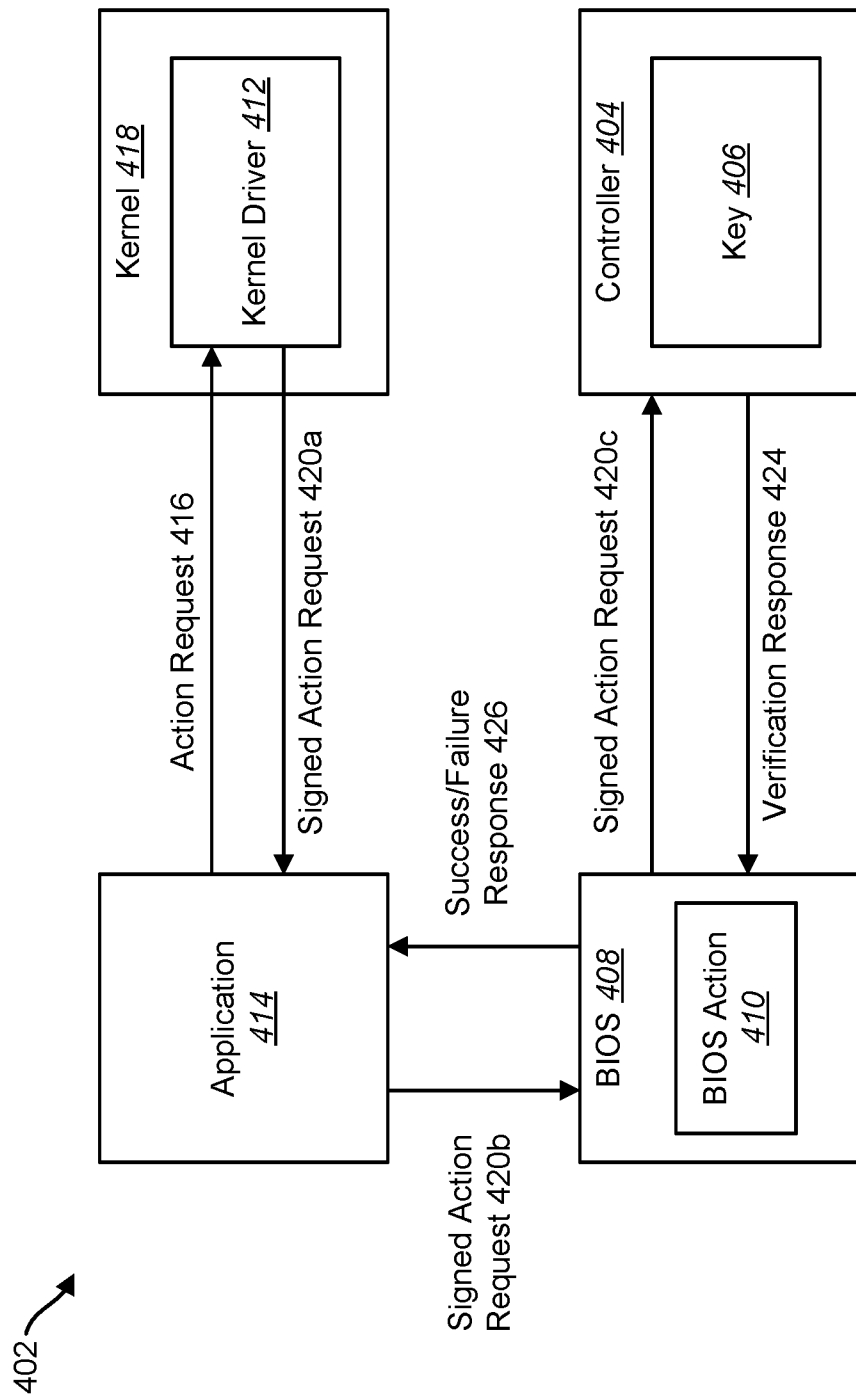


FIG. 4

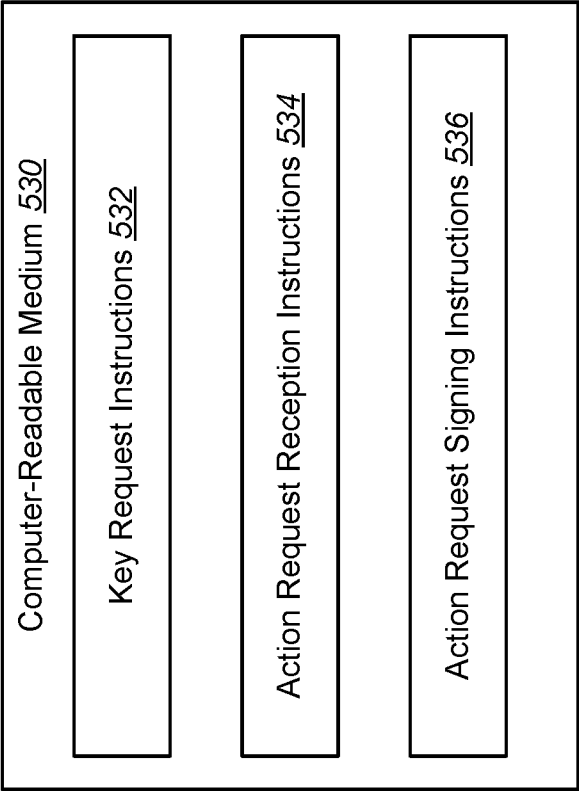
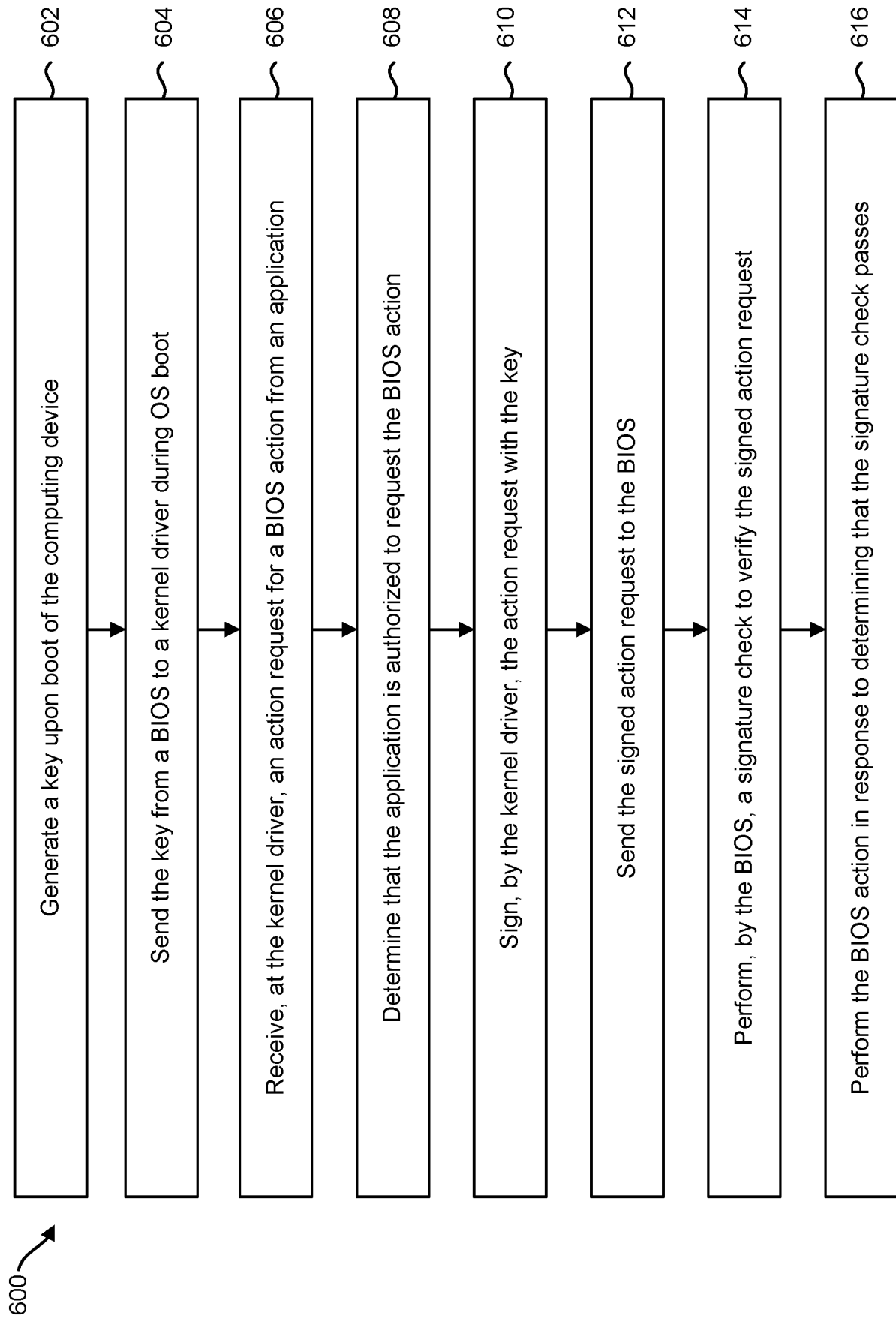


FIG. 5

6/6

**FIG. 6**

INTERNATIONAL SEARCH REPORT

International application No.

PCT/US 2020/044307

A. CLASSIFICATION OF SUBJECT MATTER		
G06F 9/50 (2006.01) G06F 12/00 (2006.01) 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 9/50, 12/00, 21/44		
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)		
EAPATIS, ESPACENET, PatSearch (RUPTO internal), USPTO, PATENTSCOPE, Google		
C. DOCUMENTS CONSIDERED TO BE RELEVANT		
Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	WO 2019/147360 A1 (DIEBOLD NIXDORF, INCORPORATED) 01.08.2019, abstract, paragraphs [0021], [0023], [0033], [0038], claim 2	1-12
X	US 2019/0236278 A1 (DELL PRODUCTS L.P.) 01.08.2019, abstract, paragraphs [0012], [0015], [0019], [0025], [0031]-[0033], [0040], [0042], claims 2, 3	13-15
Y		1-12
Y	US 8156298 B1 (ADAM STUBBLEFIED) 10.04.2012, abstract, column 2, lines 30-52, column 10, lines 30-36	4, 6
A	US 2012/0191965 A1 (PING-YING CHU) 26.07.2012	1-15
☐ 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 “D” document cited by the applicant in the international application “E” earlier document 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
01 April 2021 (01.04.2021)		15 April 2021 (15.04.2021)
Name and mailing address of the ISA/RU: Federal Institute of Industrial Property, Berezhkovskaya nab., 30-1, Moscow, G-59, GSP-3, Russia, 125993 Facsimile No: (8-495) 531-63-18, (8-499) 243-33-37		Authorized officer V. Zakharov Telephone No. +7 (495) 531-64-81