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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
26 January 2017 (26.01.2017)

WIPO | PCT

(10) International Publication Number
WO 2017/014889 A1

(51) International Patent Classification:
G06F 21/60 (2013.01)

(21) International Application Number:
PCT/US2016/038396

(22) International Filing Date:
20 June 2016 (20.06.2016)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

62/194,763	20 July 2015 (20.07.2015)	US
62/195,148	21 July 2015 (21.07.2015)	US
62/195,595	22 July 2015 (22.07.2015)	US
62/195,600	22 July 2015 (22.07.2015)	US
14/979,002	22 December 2015 (22.12.2015)	US

(71) Applicant: **INTEL CORPORATION** [US/US]; 2200 Mission College Boulevard, Santa Clara, California 95054 (US).

(72) Inventors: **CHHABRA, Siddhartha**; 1221 NE 51st Avenue, Apt. 30, Hillsboro, Oregon 97124 (US). **GERZON, Gideon**; 40 B Wingate Street, 30900 Zichron Yaakov (IL). **LAL, Reshma**; 2111 NE 25th. Avenue, MS: JF2-65, Hillsboro, Oregon 97124 (US). **XING, Bin**; 2756 NE Aurora Drive, Hillsboro, Oregon 97124 (US). **PAPPACHAN, Pradeep M.**; 2111 NE 25th Avenue, Mailstop: JF2-55, Hillsboro, Oregon 97124 (US). **MCGOWAN, Steven B.**; 2565 SW 112th Place, Portland, Oregon 97225 (US).

(74) Agent: **KELLETT, Glen M.**; Barnes & Thornburg LLP, c/o CPA Global, P.O. Box 52050, Minneapolis, Minnesota 55402 (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, 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,

[Continued on next page]

(54) Title: TECHNOLOGIES FOR SECURE PROGRAMMING OF A CRYPTOGRAPHIC ENGINE FOR SECURE I/O

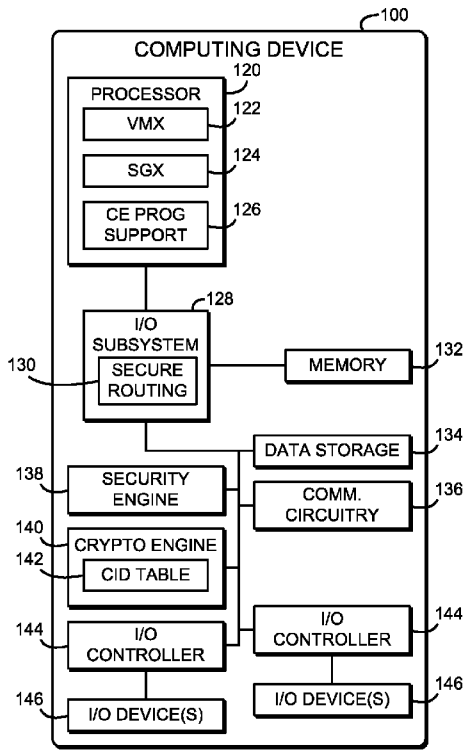

FIG. 1

(57) Abstract: Technologies for secure programming of a cryptographic engine include a computing device with a cryptographic engine and one or more I/O controllers. The computing device establishes, an invoking secure enclave using secure enclave support of a processor. The invoking enclave configures channel programming information, including a channel key, and invokes a processor instruction with the channel programming information as a parameter. The processor generates wrapped programming information including an encrypted channel key and a message authentication code. The encrypted channel key is protected with a key known only to the processor. The invoking enclave provides the wrapped programming information to untrusted software, which invokes a processor instruction with the wrapped programming information as a parameter. The processor unwraps and verifies the wrapped programming information and then programs the cryptographic engine. The processor generates an authenticated response that may be verified by the invoking enclave. Other embodiments are described and claimed.

WO 2017/014889 A1

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, 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))*

TECHNOLOGIES FOR SECURE PROGRAMMING OF A CRYPTOGRAPHIC ENGINE FOR SECURE I/O

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] The present application claims priority to U.S. Utility Patent Application Serial No. 14/979,002, entitled "TECHNOLOGIES FOR SECURE PROGRAMMING OF A CRYPTOGRAPHIC ENGINE FOR TRUSED I/O," which was filed on December 22, 2015 and which claims priority under 35 U.S.C. § 119(e) to U.S. Provisional Patent Application Serial No. 62/194,763, entitled "CRYPTOGRAPHIC PROTECTION OF I/O DATA FOR DMA CAPABLE I/O CONTROLLERS," which was filed on July 20, 2015, to U.S. Provisional Patent Application Serial No. 62/195,148, entitled "CRYPTOGRAPHIC PROTECTION OF I/O DATA FOR DMA CAPABLE I/O CONTROLLERS," which was filed on July 21, 2015, to U.S. Provisional Patent Application Serial No. 62/195,595, entitled "TECHNOLOGIES FOR SECURE PROGRAMMING OF A COMMON CRYPTO ENGINE FOR TRUSTED I/O," which was filed on July 22, 2015, and to U.S. Provisional Patent Application Serial No. 62/195,600, entitled "TECHNOLOGIES FOR SECURE COMMAND UNWRAPPING AND ERROR REPORTING FOR TRUSTED I/O," which was filed on July 22, 2015.

BACKGROUND

[0002] Typical computing devices may rely on software agents, such as anti-malware agents, for security. However, it is difficult to keep up with the increasing number of malware attacks on users' devices. To combat the malware threat, there is a trend to protect security sensitive software by running it inside a Trusted Execution Environment (TEE). TEEs provide a sterile environment that can protect secrets even when other parts of the system are compromised. Examples of TEEs include Intel® Software Guard Extensions (Intel® SGX), secure virtual machines (VMs), and a converged security engine (CSE). The TEE, while useful to protect secrets within the TEE, may not protect I/O data such as user and sensor data that is communicated into and/or out of the secure "container." The security requirements for trusted I/O vary per use case and device, and involve flavors and combinations of confidentiality, integrity, liveliness, and replay protection.

[0003] On a personal computer platform, securing I/O has several complexities. To protect I/O for a given usage, many input devices may need to be secured because the platform often has multiple devices of the same category connected via different I/O controllers, and a user may dynamically select any one of the connected devices during use. For example, when inputting text, the user may choose to use an embedded keyboard, a USB keyboard, or a

Bluetooth (BT) keyboard. The user may also use a touch screen to input data. This means all keyboards and touch input may need to be secured for a usage that requires secure text input. Additionally, I/O devices may be used by secure applications and by regular applications, which means that those devices may be required to switch dynamically from being protected to being in the clear and vice versa.

BRIEF DESCRIPTION OF THE DRAWINGS

[0004] The concepts described herein are illustrated by way of example and not by way of limitation in the accompanying figures. For simplicity and clarity of illustration, elements illustrated in the figures are not necessarily drawn to scale. Where considered appropriate, reference labels have been repeated among the figures to indicate corresponding or analogous elements.

[0005] FIG. 1 is a simplified block diagram of at least one embodiment of a computing device for secure programming of a central cryptographic engine;

[0006] FIG. 2 is a simplified block diagram of at least one embodiment of the cryptographic engine of the computing device of FIG. 1;

[0007] FIG. 3 is a simplified block diagram of at least one embodiment of an environment that may be established by the computing device of FIG. 1

[0008] FIG. 4 is a simplified block diagram of at least one embodiment of a system architecture that may be established by the computing device of FIGS. 1-3;

[0009] FIG. 5 is a simplified flow diagram of at least one embodiment of a method for secure programming of a central cryptographic engine that may be executed by the computing device of FIGS. 1-4;

[0010] FIG. 6 is a simplified flow diagram of at least one embodiment of a method for secure programming information binding that may be executed by a processor of the computing device of FIGS. 1-4;

[0011] FIG. 7 is pseudocode illustrating at least one embodiment of the method of FIG. 6;

[0012] FIGS. 8A and 8B are a simplified flow diagram of at least one embodiment of a method for secure programming information unwrapping that may be executed by the processor of the computing device of FIGS. 1-4; and

[0013] FIG. 9 is pseudocode illustrating at least one embodiment of the method of FIGS. 8A and 8B.

DETAILED DESCRIPTION OF THE DRAWINGS

[0014] While the concepts of the present disclosure are susceptible to various modifications and alternative forms, specific embodiments thereof have been shown by way of example in the drawings and will be described herein in detail. It should be understood, however, that there is no intent to limit the concepts of the present disclosure to the particular forms disclosed, but on the contrary, the intention is to cover all modifications, equivalents, and alternatives consistent with the present disclosure and the appended claims.

[0015] References in the specification to "one embodiment," "an embodiment," "an illustrative embodiment," etc., indicate that the embodiment described may include a particular feature, structure, or characteristic, but every embodiment may or may not necessarily include that particular feature, structure, or characteristic. Moreover, such phrases are not necessarily referring to the same embodiment. Further, when a particular feature, structure, or characteristic is described in connection with an embodiment, it is submitted that it is within the knowledge of one skilled in the art to effect such feature, structure, or characteristic in connection with other embodiments whether or not explicitly described. Additionally, it should be appreciated that items included in a list in the form of "at least one of A, B, and C" can mean (A); (B); (C); (A and B); (A and C); (B and C); or (A, B, and C). Similarly, items listed in the form of "at least one of A, B, or C" can mean (A); (B); (C); (A and B); (A and C); (B and C); or (A, B, and C).

[0016] The disclosed embodiments may be implemented, in some cases, in hardware, firmware, software, or any combination thereof. The disclosed embodiments may also be implemented as instructions carried by or stored on one or more transitory or non-transitory machine-readable (e.g., computer-readable) storage media, which may be read and executed by one or more processors. A machine-readable storage medium may be embodied as any storage device, mechanism, or other physical structure for storing or transmitting information in a form readable by a machine (e.g., a volatile or non-volatile memory, a media disc, or other media device).

[0017] In the drawings, some structural or method features may be shown in specific arrangements and/or orderings. However, it should be appreciated that such specific arrangements and/or orderings may not be required. Rather, in some embodiments, such features may be arranged in a different manner and/or order than shown in the illustrative figures. Additionally, the inclusion of a structural or method feature in a particular figure is not meant to imply that such feature is required in all embodiments and, in some embodiments, may not be included or may be combined with other features.

[0018] Referring now to FIG. 1, in an illustrative embodiment, a computing device 100 for secure I/O includes, among other components, a processor 120, main memory 132, a hardware cryptographic engine 140, and one or more I/O controllers 144 in communication with one or more I/O devices 146. In use, the cryptographic engine 140 provides on-the-fly encryption and decryption of data transferred via direct memory access (DMA) transactions between the platform I/O controllers 144 and the memory 132. Each DMA transaction is tagged with a channel ID (CID) representing a flow of data associated with a particular I/O device 146 or set of I/O devices 146. The cryptographic engine 140 uses the CID to reliably identify transactions that must be protected, retrieve the corresponding encryption keys, and perform appropriate cryptographic operations on the DMA data. The cryptographic engine 140 is programmed, for example with channel information and associated encryption keys, by trusted software using one or more specialized instructions of the processor 120 to generate wrapped programming information. The trusted software provides the wrapped programming information to untrusted software such as a kernel-mode driver. The untrusted software invokes an unwrapping engine using one or more specialized instructions of the processor 120 to unwrap the programming information and program the cryptographic engine 140. By using specialized processor 120 instructions to wrap the programming information, the computing device 100 may ensure that only trusted software programs the cryptographic engine 140. By allowing untrusted software to invoke the unwrapping engine, the computing device 100 allows system software (e.g., an operating system and/or VMM) to manage programming of the cryptographic engine 140. Additionally, by performing wrapping and/or unwrapping with the processor 120, the computing device 100 may reduce complexity of the cryptographic engine 140.

[0019] The computing device 100 may be embodied as any type of computation or computer device capable of performing the functions described herein, including, without limitation, a computer, a desktop computer, a workstation, a server, a laptop computer, a notebook computer, a tablet computer, a mobile computing device, a wearable computing device, a network appliance, a web appliance, a distributed computing system, a processor-based system, and/or a consumer electronic device. As shown in FIG. 1, the computing device 100 illustratively includes a processor 120, an input/output subsystem 128, a memory 132, a data storage device 134, and communication circuitry 136. Of course, the computing device 100 may include other or additional components, such as those commonly found in a desktop computer (e.g., various input/output devices), in other embodiments. Additionally, in some embodiments, one or more of the illustrative components may be incorporated in, or otherwise

form a portion of, another component. For example, the memory 132, or portions thereof, may be incorporated in the processor 120 in some embodiments.

[0020] The processor 120 may be embodied as any type of processor capable of performing the functions described herein. The processor 120 may be embodied as a single or multi-core processor(s), digital signal processor, microcontroller, or other processor or processing/controlling circuit. As shown, the processor 120 may include hardware virtualization support 122, secure enclave support 124, and crypto engine programming support 126.

[0021] The hardware virtualization support 122 supports virtualized execution of operating systems, applications, and other software by the computing device 100. The hardware virtualization support 122 may include virtual machine extensions (VMX) support by providing two modes of execution: VMX-root mode and VMX non-root mode. The VMX-root mode allows executing software to have broad control of the computing device 100 and its hardware resources. Conversely, a hypervisor, virtual machine monitor (VMM), or host operating system (OS) may execute in VMX-root mode. The VMX non-root mode restricts access to certain hardware instructions while still implementing the ordinary ring/privilege system of the processor 120. One or more guest OSs may execute in the VMX non-root mode. Those guest OSs may execute in ring zero, similar to being executed without virtualization. The hardware virtualization support 122 may also support extended page tables (EPT), which may be embodied as hardware-assisted second-level page address translation. The hardware virtualization support 122 may be embodied as, for example, Intel® VT-x technology.

[0022] The secure enclave support 124 allows the processor 120 to establish a trusted execution environment known as a secure enclave, in which executing code may be measured, verified, and/or otherwise determined to be authentic. Additionally, code and data included in the secure enclave may be encrypted or otherwise protected from being accessed by code executing outside of the secure enclave. For example, code and data included in the secure enclave may be protected by hardware protection mechanisms of the processor 120 while being executed or while being stored in certain protected cache memory of the processor 120. The code and data included in the secure enclave may be encrypted when stored in a shared cache or the main memory 132. The secure enclave support 124 may be embodied as a set of processor instruction extensions that allows the processor 120 to establish one or more secure enclaves in the memory 132. For example, the secure enclave support 124 may be embodied as Intel® Software Guard Extensions (SGX) technology.

[0023] The crypto engine programming support 126 allows the processor 120 to program the cryptographic engine 140 to provide cryptographic protection of I/O data. In particular, the processor 120 may enable or disable encryption for certain I/O channels, and may securely provide encryption keys to the cryptographic engine 140. The crypto engine programming support 126 may be embodied as one or more specialized processor instructions (e.g., the instructions EBINDTIO, UNWRAP, or other instructions) and associated hardware, microcode, firmware, or other components of the processor 120.

[0024] The memory 132 may be embodied as any type of volatile or non-volatile memory or data storage capable of performing the functions described herein. In operation, the memory 132 may store various data and software used during operation of the computing device 100 such as operating systems, applications, programs, libraries, and drivers. The memory 132 is communicatively coupled to the processor 120 via the I/O subsystem 128, which may be embodied as circuitry and/or components to facilitate input/output operations with the processor 120, the memory 132, and other components of the computing device 100. For example, the I/O subsystem 128 may be embodied as, or otherwise include, memory controller hubs, input/output control hubs, platform controller hubs, integrated control circuitry, firmware devices, communication links (i.e., point-to-point links, bus links, wires, cables, light guides, printed circuit board traces, etc.) and/or other components and subsystems to facilitate the input/output operations. The I/O subsystem 128 may further include secure routing support 130. The secure routing support 130 includes hardware support to ensure I/O data cannot be misrouted in the fabric 128 under the influence of rogue software. The secure routing support 130 may be used with the cryptographic engine 140 to provide cryptographic protection of I/O data. In some embodiments, the I/O subsystem 128 may form a portion of a system-on-a-chip (SoC) and be incorporated, along with the processor 120, the memory 132, and other components of the computing device 100, on a single integrated circuit chip.

[0025] The data storage device 134 may be embodied as any type of device or devices configured for short-term or long-term storage of data such as, for example, memory devices and circuits, memory cards, hard disk drives, solid-state drives, or other data storage devices. In some embodiments, the data storage device 134 may be used to store the contents of one or more secure enclaves. When stored by the data storage device 134, the contents of the secure enclave may be encrypted to prevent unauthorized access.

[0026] The communication circuitry 136 of the computing device 100 may be embodied as any communication circuit, device, or collection thereof, capable of enabling communications between the computing device 100 and other remote devices over a network.

The communication circuitry 136 may be configured to use any one or more communication technology (e.g., wired or wireless communications) and associated protocols (e.g., Ethernet, Bluetooth®, Wi-Fi®, WiMAX, etc.) to effect such communication.

[0027] In some embodiments, the computing device 100 may include a security engine 138, which may be embodied as any hardware component(s) or circuitry capable of providing security-related services to the computing device 100. In particular, the security engine 138 may include a microprocessor, microcontroller, or other embedded controller capable of executing firmware and/or other code independently and securely from the processor 120. Thus, the security engine 138 may be used to establish a trusted execution environment separate from code executed by the processor 120. The security engine 138 may communicate with the processor 120 and/or other components of the computing device 100 over a dedicated bus, such as a host embedded controller interface (HECI). The security engine 138 may also provide remote configuration, control, or management of the computing device 100. In the illustrative embodiment, the security engine 138 is embodied as a converged security and manageability engine (CSME) incorporated in a system-on-a-chip (SoC) of the computing device 100. In some embodiments, the security engine 138 may be embodied as a manageability engine, an out-of-band processor, a Trusted Platform Module (TPM), or other security engine device or collection of devices. Further, in some embodiments, the security engine 138 is also capable of communicating using the communication circuitry 136 or a dedicated communication circuit independently of the state of the computing device 100 (e.g., independently of the state of the main processor 120), also known as "out-of-band" communication.

[0028] The cryptographic engine 140 may be embodied as any microcontroller, microprocessor, functional block, logic, or other circuit or collection of circuits capable of performing the functions described herein. As further described below, the cryptographic engine 140 may encrypt and/or decrypt I/O data read or written by the I/O controllers 144 in one or more direct memory access (DMA) operations to the memory 132. The cryptographic engine 140 includes an internal channel identifier (CID) table 142, which the cryptographic engine 140 uses to dynamically identify DMA channel(s) to be protected. The CID table 142 may be controlled and/or programmed by trusted software, for example using the crypto engine programming support 126 of the processor 120. The encryption keys and/or other secret information of the CID table 142 are not available to untrusted software. In some embodiments, the cryptographic engine 140 may be incorporated along with the I/O subsystem 128 and/or the processor 120 in a system-on-a-chip (SoC) of the computing device 100.

[0029] Similarly, the I/O controllers 144 may be embodied as any embedded controller, microcontroller, microprocessor, functional block, logic, or other circuit or collection of circuits capable of performing the functions described herein. In some embodiments, one or more of the I/O controllers 144 may be embedded in another component of the computing device 100 such as the I/O subsystem 128 and/or the processor 120. Additionally or alternatively, one or more of the I/O controllers 144 may be connected to the I/O subsystem 128 and/or the processor 120 via an expansion bus such as PCI Express (PCIe) or other I/O connection. As further described below, the I/O controllers 144 communicate with one or more I/O devices 146, for example over a peripheral communications bus (e.g., USB, Bluetooth, etc.). The I/O devices 146 may be embodied as any I/O device, such as human interface devices, keyboards, mice, touch screens, microphones, cameras, and other input devices, as well as displays and other output devices. As described above, the I/O controllers 144 and associated DMA channels are uniquely identified using identifiers called channel identifiers (CIDs). Each I/O controller 144 may assert an appropriate CID with every DMA transaction, for example as part of a transaction layer packet (TLP) prefix, to uniquely identify the source of the DMA transaction and provide liveness protections. The CID also enables the isolation of I/O from different devices 146.

[0030] Referring now to FIG. 2, diagram 200 illustrates hardware of the cryptographic engine 140 at a high level. As shown, the CE hardware 140 implements a CID table 142 and a request bank 202. In some embodiments, the CID table 142 may be embodied as content addressable memory (CAM) 142. The CID table 142 is loaded with channel information associated with a trusted channel such as CID, direction of the trusted channel (e.g., input or output), skip length, cryptographic mode, and the associated channel key. The CIDs are platform implementation defined values. For example, the CID may include a controller identifier field and a channel number field. The size of the respective fields and their assigned values may be platform 100 and/or controller 144 dependent. Thus, it should be understood that the fields shown in the CID table 142 are illustrative, and that in some embodiments, additional fields may be stored with each CID entry in the CID table 142, which may allow the cryptographic engine 140 to manage DMA transactions.

[0031] In use, the cryptographic engine 140 snoops all DMA transactions generated by the I/O controllers 144 to the memory 132. On each transaction to or from a device 146 capable of participating in trusted I/O, the cryptographic engine 140 references the CID table 142 to find the CID corresponding to the DMA channel in the CID table 142. A match indicates that the channel is currently protected and that the cryptographic engine 140 should use the channel

key associated with the channel to protect the data written to and/or the data read from memory 132 (depending on the direction of the channel). The request bank 202 represents a set of registers that receive the unwrapped channel programming information for programming a channel from the untrusted software. As described below, the request bank 202 may receive the unwrapped channel programming from the processor 120 via a sideband interface that it inaccessible to software executed by the computing device 100.

[0032] Referring now to FIG. 3, in an illustrative embodiment, the computing device 100 establishes an environment 300 during operation. The illustrative environment 300 includes a secure programming module 302, a binding module 304, an unsecure programming module 306, and an unwrapping engine module 308. The various modules of the environment 300 may be embodied as hardware, firmware, microcode, software, or a combination thereof. As such, in some embodiments, one or more of the modules of the environment 300 may be embodied as circuitry or collection of electrical devices (e.g., secure programming circuitry 302, binding circuitry 304, unsecure programming circuitry 306, and/or unwrapping engine circuitry 308). It should be appreciated that, in such embodiments, one or more of the secure programming circuitry 302, the binding circuitry 304, the unsecure programming circuitry 306, and/or the unwrapping engine circuitry 308 may form a portion of one or more of the processor 120, the I/O subsystem 128, the cryptographic engine 140, and/or other components of the computing device 100. Additionally, in some embodiments, one or more of the illustrative modules may form a portion of another module and/or one or more of the illustrative modules may be independent of one another.

[0033] The secure programming module 302 is configured to establish, with the secure enclave support 124 of the processor 120, a secure enclave called an invoking enclave. The secure programming module 302 is further configured to configure, by the invoking enclave, channel programming information. The channel programming information includes a channel identifier and a channel key that are to be programmed to the cryptographic engine 140. The secure programming module 302 is further configured to invoke, by the invoking enclave, a processor instruction of the processor 120 with the channel programming information as a parameter. The secure programming module 302 may be further configured to provide, by the invoking enclave, wrapped programming information to an untrusted kernel mode component of the computing device 100. In some embodiments, the secure programming module 302 may be further configured to receive, by the invoking enclave, an authenticated response from the untrusted kernel mode component in response to providing the wrapped programming

information to the untrusted kernel mode component, and verify the authenticated response with the channel key and a random nonce of the channel programming information.

[0034] The binding module 304 is configured to generate, by the processor 120, the wrapped programming information based on the channel programming information in response to invocation of the processor instruction by the invoking enclave. The wrapped programming information includes an encrypted channel key and a message authentication code generated over the channel programming information.

[0035] The unsecure programming module 306 is configured to receive, by the untrusted kernel mode component, the wrapped programming information from the invoking enclave and to invoke, by the untrusted kernel mode component, a processor instruction of the processor 120 with the wrapped programming information as a parameter. As described above, the wrapped programming information includes an encrypted channel key to be programmed to the cryptographic engine 140. The unsecure programming module 306 may be further configured to read, by the untrusted kernel mode component, an authenticated response generated by the processor 120 and provide the authenticated response to the invoking enclave.

[0036] The unwrapping engine module 308 is configured to unwrap, by the processor 120, the wrapped programming information to generate the channel programming information in response to invocation of the processor instruction by the untrusted kernel mode component. As described the channel programming information includes an unencrypted channel key. The unwrapping engine module 308 is further configured to verify the channel programming information and program the unencrypted channel key to the cryptographic engine 140 in response to verifying the channel programming information. The unwrapping engine module 308 may be further configured to determine whether the wrapped programming information is potentially replayed and if so, indicate an unwrapping error. The unwrapping engine module 308 is further configured to determine, by the processor 120, whether one or more target-specific programming checks are satisfied and, if not, indicate an error. The unwrapping engine module 308 is further configured to generate, by the processor 120, an authenticated response based on programming status in response to verifying the channel programming information.

[0037] Referring now to FIG. 4, diagram 400 illustrates a system architecture that may be established by the computing device 100. The system architecture may include an untrusted I/O stack including an application 402, a device driver 404, a filter driver 406, and a bus driver 408. The untrusted I/O stack may receive unprotected (i.e., plaintext) I/O data from the I/O controllers 144 via the cryptographic engine 140 and process the I/O data as normal. The system architecture may also include a trusted I/O stack including an application enclave 410

and a device driver enclave 412. Each of the enclaves 410, 412 may be established using the secure enclave support 124 of the processor 120 and thus may be trusted. As shown, each of the enclaves 410, 412 may be provisioned with encryption keys associated with one or more DMA channels. Thus, the application enclave 410 and/or the device driver enclave 412 may securely decrypt and process secure I/O data generated from the I/O devices 146 via the cryptographic engine 140. As shown, each secure enclave 410, 412 may receive secure I/O data via components of the untrusted I/O stack, such as the bus driver 408 and/or the filter driver 406. In particular, I/O control and routing may be performed by the untrusted I/O stack, and because the payload of the secure I/O data is encrypted, the secure I/O data remains protected. Accordingly, the untrusted I/O stack need not be included in the trusted code base of the computing device 100. In some embodiments, the untrusted I/O stack, including the filter driver 406, the bus driver 408, and/or other untrusted I/O components may be re-used or otherwise shared with an ordinary operating system of the computing device 100.

[0038] As shown, the system architecture 400 further includes a crypto engine enclave 414, a crypto engine driver 416, and an unwrapping engine 418, which may be used to program the cryptographic engine 140. The crypto engine enclave 414 may be embodied as user-level (e.g., ring-3) code protected with the secure enclave support 124 of the processor 120 and thus may be trusted. The crypto engine enclave 414 maintains or otherwise has access to encryption keys associated with one or more DMA channels. The crypto engine enclave 414 may provision the trusted I/O stack with the encryption keys. The crypto engine enclave 414 may also program the cryptographic engine 140 using the crypto engine programming support 126 of the processor 120. In particular, the crypto engine enclave 414 may execute one or more specialized processor instruction to prepare a binary blob including wrapped channel programming information, including wrapped encryption keys that may be used to program the cryptographic engine 140. The crypto engine enclave 414 may provide the binary blob to the crypto engine driver 416, which may be embodied as a kernel-level untrusted software component. The crypto engine driver 416 provides the binary blob to the unwrapping engine 418, which may unwrap and verify the binary blob and, if verified, program the channel programming information to the cryptographic engine 140. Thus, the crypto engine driver 416 may allow an operating system or other control software of the computing device 100 to manage programming of the cryptographic engine 140 without requiring the operating system to have access to the plaintext encryption keys for the DMA channels. In the illustrative embodiment, the unwrapping engine 418 is provided by hardware and/or microcode resources of the processor 120; however, in some embodiments the functions of the unwrapping engine

418 may be performed by non-core components of the processor 120 (i.e., the processor uncore), the cryptographic engine 140, and/or other components of the computing device 100.

[0039] Referring now to FIG. 5, in use, the computing device 100 may execute a method 500 for secure programming of the cryptographic engine 140. The method 500 may be executed by hardware, firmware, processor microcode, software, or other execution resources of the computing device 100. The method 500 begins with block 502, in which a secure enclave generates a key to program the DMA channel. For example, in some embodiments the crypto engine enclave (CEE) 414 may generate the key used to program the DMA channel. The key may include a channel key used to protect I/O data transmitted over the DMA channel. Therefore, the secure enclave may also provide the channel key to other trusted components of the computing device 100 that may access the protected I/O data, such as the application enclave 410 and/or the device driver enclave 412. The secure enclave may retain the channel programming key to verify an authenticated response generated by the cryptographic engine 140, as described further below.

[0040] In block 504, the invoking secure enclave (e.g., the CEE 414) prepares channel programming information to be used to program the DMA channel. The channel programming information may include the encryption key as determined in block 502 as well as other programming information, such as the channel identifier (CID) of the DMA channel to be programmed, a programming command, a random nonce that may be used for authentication and replay protection, and other programming information. To prepare the programming information, the invoking enclave may, for example, allocate a structure in memory including the programming information, also called a "binary blob." In some embodiments, the programming information may be stored in a structure known as a BIND_STRUCT, which may include fields as described below in Table 1.

[0041]

Name of Offset	Offset	Size (B)	Description	Set by
BTID	0	4	Target device	Software
BTSVN	4	4	Target security version number	Software
BTPOLICY	8	16	Target device policy	Software
CID_IO	24	4	Channel ID to be programmed. MSB indicates input or output	Software
TKEY	28	16	Channel key for the target	Software
NONCE	44	8	Nonce for authenticated responses	Software
COMMAND	52	4	Programming commands to the target device	Software
SEQID	56	8	Seed for generating initialization vector (IV)	Hardware
MAC	64	16	MAC on encrypted keys, policy, target ID, SVN, CID_IO, NONCE, and INVOCATN_CTR	Hardware
RSVD	80	48	Reserved	Hardware

Table 1. Bind key structure (BIND_STRUCT).

[0042] The bind target ID (BTID) field in BIND_STRUCT is set up by the invoking enclave, which will eventually invoke a processor instruction to program to a target device. The target device is illustratively the cryptographic engine 140, although is some embodiments the computing device 100 may support alternative target devices. The BTID field is set to the identifier of the target device (e.g., the cryptographic engine 140) to enable the unwrapping engine 418 to direct the programming to the desired target device. In the illustrative embodiment, the CEE 414 sets up the BTID field to include the target ID of the cryptographic engine 140. It should be understood that in some embodiments, the programming information may be bound to an endpoint other than the cryptographic engine 140.

[0043] The bind target security version number (BTSVN) field is set up by the invoking entity and contains the security version number (SVN) for any firmware running on the endpoint device (e.g., the cryptographic engine 140). In some embodiments, the cryptographic engine 140 of the computing device 100 may not include any firmware and thus the BTSVN field must be zero (MBZ). The bind target policy (BTPOLICY) field is set up by the invoking enclave and contains the policy that must be applied to the data being processed by the target device (e.g., the cryptographic engine 140). In some embodiments, the cryptographic engine 140 may not support any defined policy and thus this field must be zero (MBZ).

[0044] The channel ID (CID_IO) field is set up by the invoking enclave and contains the channel identifier and the direction of the channel (i.e., input or output) that the invoking enclave is attempting to program. The CID_IO field is used by the invoking enclave to program the channel identifier of the DMA channel being protected for trusted I/O. In some embodiments, CID_IO may include more than four bytes. The TKEY field is set up by the invoking enclave and contains the encryption key to be programmed to the target device (e.g., the cryptographic engine 140). This key may be used to protect I/O data communicated over the DMA channel.

[0045] The random nonce field is set up by the invoking enclave and contains a random nonce which is used by the cryptographic engine 140 for generating an authenticated response as described further below. The authenticated response may be used by the invoking enclave to verify that the programming attempt to the cryptographic engine 140 was successful.

[0046] The command (COMMAND) field is set up by the invoking enclave and indicates the type of programming to be done for the target device (e.g., the cryptographic engine 140). The cryptographic engine 140 may support one or more commands, including protecting a channel (CH_PROTECT), unprotecting a channel (CH_UNPROTECT), reprogramming a channel with a different key (CH_REPROGRAM), or querying for the key associated with a channel (QUERY_KEY).

[0047] As shown, the BIND_STRUCT structure may also include fields that are set by hardware of the processor 120, including a sequence number (SEQID) and a message authentication code (MAC). Generation of those fields by the processor 120 is described further below. Of course, the BIND_STRUCT structure illustrates one potential embodiment of the channel programming information, and the programming information may be stored in different formats in other embodiments. For example, in some embodiments, the programming information may include variable amounts of target-specific data and/or wrapped data, as well as associated size fields that may be interpreted by the processor 120.

[0048] Still referring to FIG. 5, in block 506, the invoking enclave invokes a processor instruction of the processor 120 to generate wrapped programming information. The invoking enclave may pass the BIND_STRUCT including the channel programming key as a parameter to the processor instruction. The processor 120 encrypts the key of the channel programming information to generate an encrypted key. The processor 120 may encrypt the key using a key wrapping key (KWK) known only to the processor 120 and the unwrapping engine 418. The processor 120 may generate and include a sequence number in the wrapped programming information for replay protection, and the processor 120 may generate a MAC over the channel

programming information for integrity protection. The processor 120 may modify the BIND_STRUCT to contain the wrapped programming information. In some embodiments, in block 508 the invoking enclave may invoke an EBINDTIO instruction. The EBINDTIO instruction may be embodied as a user-level (e.g., ring 3) instruction. One potential embodiment of a method for executing the EBINDTIO instruction is described below in connection with FIG. 6.

[0049] In block 510, the invoking enclave provides the wrapped programming information to untrusted, kernel-mode software of the computing device, such as the crypto engine driver 416. Because the wrapped programming information has been encrypted and bound to the cryptographic engine 140, sensitive data in the channel programming information (e.g., the channel programming key) may not be accessed by the untrusted software. The untrusted software may inspect unprotected fields of the wrapped programming information (e.g., the CID_IO and COMMAND fields) to determine whether to allow the programming attempt. Thus, kernel-mode software such as the crypto engine driver 416 may manage programming of the cryptographic engine 140 without being trusted or otherwise capable of accessing the protected I/O data.

[0050] In block 512, the untrusted software (e.g., the crypto engine driver 416) invokes a processor instruction of the processor 120 to unwrap the programming information and securely program the DMA channel. For example, the crypto engine driver 416 may invoke a processor instruction that causes the unwrapping engine 418 (e.g., the processor 120 and/or the cryptographic engine 140) to decrypt the channel programming key, verify the channel programming information, and otherwise program the DMA channel. To program the cryptographic engine 140, the unwrapping engine 418 may copy the channel programming information into an appropriate entry of the CID table 142, clear an entry of the CID table 142 identified by the channel programming information, or otherwise modify the CID table 142. After programming, the unwrapping engine 418 and/or cryptographic engine 140 generates an authenticated response indicating the programming status and/or the unwrapping status. In some embodiments, in block 514 the untrusted software invokes an UNWRAP instruction of the processor 120. The UNWRAP instruction may be embodied as a kernel-level (e.g., ring 0) instruction. In some embodiments, the UNWRAP instruction may generate a virtual machine exit (VMExit), allowing a VMM and/or hypervisor to manage virtualization of the UNWRAP instruction. One potential embodiment of a method for executing the UNWRAP instruction is described below in connection with FIGS. 8A and 8B.

[0051] In block 516, the untrusted software (e.g., the crypto engine driver 416) reads the authenticated response from the memory 132 and returns the authenticated response to the invoking enclave (e.g., to the CEE 414). The untrusted software may also evaluate unencrypted fields of the authenticated response, such as a programming status code and/or an unwrapping status code. In block 518, the invoking enclave verifies the authenticated response. Verifying the authenticated response allows the invoking enclave to determine whether the cryptographic engine 140 successfully completed the programming request. Thus, after verifying the authenticated response the invoking enclave (e.g., the CEE 414) may, for example, allow the application enclave 410 and/or the device driver enclave 412 to use the secure DMA channel. In some embodiments, in block 520 the invoking enclave may verify the authenticated response using the channel programming key and the random nonce that were included in the original channel programming information. For example, the authenticated response may include a message authentication code over the programming status that can be verifying using the channel programming key and the random nonce. After verifying the authenticated response, the method 500 loops back to block 502, in which the computing device 100 may program additional DMA channels.

[0052] Referring now to FIG. 6, in use, the computing device 100 may execute a method 600 for secure programming information binding. The method 600 may be executed by hardware, firmware, processor microcode, or other execution resources of the processor 120. Thus, the method 600 may have a hardware root of trust (i.e., the processor 120). The method 600 begins with block 602, in which the computing device 100 invokes the ENBINDTIO processor instruction. As described above in connection with block 506 of FIG. 5, a secure enclave such as the cryptographic engine enclave (CEE) 414 may invoke the EBINDTIO instruction. The invoking enclave provides channel programming information as a parameter to the EBINDTIO instruction. For example, a pointer to a BIND_STRUCT including the channel programming information may be passed in a register of the processor 120 such as RCX. In some embodiments, in response to invocation of the EBINDTIO instruction, the processor 120 may verify that a particular enclave (e.g., the CEE 414) has invoked the processor, for example by determining whether one or more attributes of the invoking enclave are set.

[0053] In block 604, the processor 120 wraps the encryption key and binds the encryption key to the unwrapping engine 418. In particular, the processor 120 encrypts the channel programming key and generates a message authentication code (MAC) to integrity-protect the channel programming information. In block 606, the processor 120 creates an initialization vector (IV) for the wrapping and binding process. The processor 120 may

generate a sequence ID on each EBINDTIO invocation by using an internally maintained monotonic counter. The sequence ID is used to construct the initialization vector for the cryptographic wrapping. Constructing the initialization vector may be performed as described by the AES-GCM standard. The processor 120 also stores the sequence ID in the SEQID field of the BIND_STRUCT. The SEQID field may be used as an invocation counter to indicate the count for the EBINDTIO invocations on the computing device 100, which may be used for replay protection as described further below.

[0054] In block 608, the processor 120 encrypts the channel programming key using a key wrapping key (KWK) to create the encrypted channel key. As described above, the KWK is known only to the processor 120 and the unwrapping engine 418. In the illustrative embodiment, the processor 120 encrypts the channel key using the AES-GCM algorithm. Of course, the processor 120 may use any appropriate cryptographic algorithm to encrypt the channel key. The processor 120 may store the encrypted channel key in the TKEY field of the BIND_STRUCT object. In block 610, processor 120 generates a message authentication code (MAC) over the encrypted channel key and other BIND_STRUCT fields. For example, in the illustrative embodiment, the MAC is generated over the BIND_STRUCT fields BTID, BTSVN, BTPOLICY, CID_IO, NONCE, SEQID, COMMAND, and the encrypted channel programming key. The MAC is stored in the MAC field of the BIND_STRUCT and allows the unwrapping engine 418 to verify that the wrapped programming information was not modified while transitioning through untrusted software of the computing device 100.

[0055] In block 612, the processor 120 returns from executing the EBINDTIO instruction. The processor 120 may, for example, resume executing the next instruction of invoking enclave (e.g., the CEE 414). After executing the EBINDTIO instruction, the memory 132 includes the wrapped programming information. For example, the SEQID, TKEY, and MAC fields of the BIND_STRUCT may include values stored by the processor 120 during execution of the EBINDTIO instruction. After returning, the method 600 is completed.

[0056] Referring now to FIG. 7, pseudocode 700 illustrates one potential embodiment of the EBINDTIO instruction. The pseudocode 700 may illustrate, for example, microcode of the processor 120 associated with the EBINDTIO instruction. As shown, the processor 120 generates a sequence ID by sampling a 64-bit monotonic counter, generates the initialization vector from the sequence ID, and stores the sequence ID in the BIND_STRUCT. The processor 120 generates an authentication header over several fields of the BIND_STRUCT, and then performs an AES-GCM authenticated encryption operation using the KWK, the initialization vector, the authentication header, and the channel programming key. The processor 120 next

stores the cipher text and the message authentication code produced by the authenticated encryption operation into the BIND_STRUCT.

[0057] Referring now to FIGS. 8A and 8B, in use, the computing device 100 may execute a method 800 for secure programming information unwrapping. The method 800 may be executed by hardware, firmware, processor microcode, or other execution resources of the processor 120. The method 800 begins with block 802, in which the computing device 100 invokes the UNWRAP processor instruction. As described above in connection with block 512 of FIG. 5, an untrusted kernel-mode (e.g., ring-0) software entity such as the crypto engine driver 416 may invoke the UNWRAP instruction. The calling untrusted software provides the wrapped programming information as a parameter to the UNWRAP instruction. For example, a pointer to a BIND_STRUCT including the wrapped programming information may be passed in a register of the processor 120 such as RCX.

[0058] In block 804, the processor 120 compares an invocation counter included in the wrapped programming information to an internal invocation counter. For example, the processor 120 may compare the SEQID field of a BIND_STRUCT that includes the wrapped programming information to the internal invocation counter. The internal invocation counter is initialized to zero and may be incremented in response to a successful invocation of the UNWRAP instruction as described further below. In block 806, the processor 120 determines whether the invocation counter received with the wrapped programming information is greater than the internal invocation counter. If not, the unwrapping attempt may be a replay attack using the wrapped programming information. If the invocation counter received with the wrapped programming information is greater than the internal invocation counter, the method 800 advances to block 810, described below. If the invocation counter received with the wrapped programming information is not greater than the internal invocation counter, the method 800 branches to block 808.

[0059] In block 808, the processor 120 indicates an unwrapping failure. For example, the processor 120 may write an appropriate error code in a response structure in the memory 132. In some embodiments, the response may be stored in a structure known as a UNWRAP_RESPONSE_STRUCT, which may include fields as described below in Table 2. The untrusted software (e.g., the crypto engine driver 416) may allocate space for the UNWRAP_RESPONSE_STRUCT in memory, and may pass a pointer to the UNWRAP_RESPONSE_STRUCT as a parameter to the UNWRAP instruction. To indicate the unwrapping code due to a suspected replay attack, the UNWRAP_STATUS field may be set to '10.' After indicating the unwrap failure, the method 800 branches ahead to block 830, in

which the processor 120 returns from executing the UNWRAP instruction and the method 800 is completed.

Name of Offset	Offset	Size (B)	Description
UNWRAP_STATUS	0	1	Bit 0: Unwrap Status (0: Success, 1: Failure), Bit 1 and 2: Failure reason (00: MAC failure, 01: Device Busy, 10: Replayed blob)
AUTHENTICATED_RSP	1	25	Authenticated Response – valid only if there were no unwrap errors, indicated by the unwrap status
RSVD	26	38	Reserved (must be zero)

Table 2. UNWRAP_RESPONSE_STRUCT

[0060] Referring back to block 806, if the invocation counter of the wrapped programming information is greater than the internal invocation counter, the method 800 advances to block 810. In block 810, the processor 120 unwraps the wrapped programming information and recovers the channel programming key. The processor 120 may decrypt the TKEY field of the BIND_STRUCT using the key wrapping key (KWK), which is known only to the processor 120 and the unwrapping engine 418 (which, in the illustrative embodiment, is also the processor 120). In the illustrative embodiment, the processor 120 decrypts the channel key using the AES-GCM algorithm. Of course, the processor 120 may use any appropriate cryptographic algorithm to decrypt the channel key

[0061] In block 812, the processor 120 verifies the MAC over the wrapped programming information. For example, the processor 120 may verify the MAC over the channel key and other BIND_STRUCT fields using an authenticated encryption algorithm such as AES-GCM. In some embodiments, the processor 120 may verify the MAC over the BIND_STRUCT fields BTID, BTSVN, BTPOLICY, CID_IO, NONCE, SEQID, COMMAND, and the channel key. Of course, the processor 120 may use any appropriate cryptographic algorithm to verify that the wrapped programming information has not been modified while transitioning through untrusted software. In block 814, the processor 120 determines whether the MAC was verified. If not, the method 800 branches to block 808, in which the processor 120 indicates an unwrapping failure as described above. In particular, the processor 120 may indicate MAC failure by setting the UNWRAP_STATUS field of the UNWRAP_RESPONSE_STRUCT to '00.' In some embodiments, the processor 120 may also

indicate unwrap failure if the UNWRAP instruction is invoked while another unwrap request is in progress by setting the UNWRAP_STATUS field of the UNWRAP_RESPONSE_STRUCT to '01' to indicate that the unwrapping engine 418 is busy. For example, the UNWRAP instruction may maintain a lock to protect shared state used by the UNWRAP instruction. After indicating the unwrap failure, the method 800 branches ahead to block 830, in which the processor 120 returns from executing the UNWRAP instruction and the method 800 is completed.

[0062] Referring back to block 814, if the MAC was verified, then the method 800 advances to block 816, in which the processor 120 updates its internal invocation counter with the invocation counter of the wrapped programming information. For example, the processor 120 may update the internal invocation counter to match the value of the SEQID field of the BIND_STRUCT object. As described above, updating the internal invocation counter may allow the processor 120 to detect attempted replay attacks.

[0063] In block 818, the processor 120 evaluates one or more target-specific checks associated with the requested programming channel programming command. The target-specific checks may verify protect the computing device 100 from malicious attack, for example by checking the consistency and/or correctness of the channel programming information. Performing the target-specific checks by the processor 120 may reduce the complexity and/or cost of the programming target. In the illustrative embodiment, the checks performed are specific to the cryptographic engine 140; however, in other embodiments the checks may be appropriate for any other component that is the target of the programming attempt.

[0064] In some embodiments, in block 820, the processor 120 may perform checks specific to the cryptographic engine 140 for commands to program a DMA channel to secure. The processor 120 may check that the channel ID received with the wrapped programming information (e.g., the BIND_STRUCT object) is not currently present in the CID table 142 of the cryptographic engine 140. In other words, the processor 120 may check that the DMA channel being programmed is not already secure. Additionally or alternatively, the processor 120 may check if there is an available entry in the CID table 142 for the programming request; that is, the processor 120 may check whether the CID table 142 is full. In order to perform those checks, the processor 120 may model the state of the CID table 142 based on the channel programming requests received. Modeling the state on the processor 120 may reduce the complexity of the cryptographic engine 140 hardware.

[0065] In some embodiments, in block 822, the processor 120 may perform checks specific to the cryptographic engine 140 for commands to program a DMA channel out of secure. The processor 120 may check that the channel ID received with the wrapped programming information (e.g., the BIND_STRUCT object) is currently programmed to secure, for example by ensuring that the received CID and direction (e.g., input and/or output) match a CID and direction combination in the CID table 142. Additionally or alternatively, the processor 120 may check that the channel programming key passed with the wrapped programming information is the same as the current channel key. By requiring the same channel programming key, the processor 120 may ensure that the same enclave that programmed the DMA channel to secure (e.g., the CEE 414) also programs the DMA channel out of secure. As described above, in order to perform those checks, the processor 120 may model the state of the CID table 142 based on the channel programming requests received.

[0066] In block 824, the processor 120 generates an authenticated response based on the programming status of the cryptographic engine 140. The authenticated response generated may have two elements, the unwrap status to indicate the status of unwrap of the EBIND blob, and a cryptographic response which is an encrypted/authenticated response representing the programming status of the CID table 142. The cryptographic response allows the invoking enclave (e.g., the CEE 414) to verify that the untrusted software actually initiated the cryptographic engine 140 programming by calling UNWRAP. For example, as described above, the authenticated response may be represented by the UNWRAP_RESPONSE_STRUCT, with the UNWRAP_STATUS field representing the unwrapping status and the AUTHENTICATED_RSP field representing the programming status. The AUTHENTICATED_RSP field may include one or more status codes or other predefined values. For example, AUTHENTICATED_RSP may include CH_PROG_SUCCESS to indicate programming was successful, CH_PROG_CAMID_UNAVAILABLE to indicate no CID table 142 entries are available for programming, CH_ALREADY_PROG to indicate that a DMA channel requested to be secure is already secure, CH_PROG_CID_UNAVAILABLE to indicate a channel ID requested to be out of secure is not in use, and/or CH_PROG_CUR_KEY_MISMATCH to indicate a channel key of the request does not match the current channel key.

[0067] In order to construct the encrypted response, each programming attempt may generate a status and optional data. In the illustrative embodiment, none of the commands for the cryptographic engine 140 return data; however it should be understood that in some embodiments data such as secret keys may be returned. The status indicates the status of the

programming and can be used by both the untrusted software as well as the invoking enclave. Thus, in in order to allow both the untrusted software and the invoking enclave to inspect the programming status, the programming status may be integrity-protected but not encrypted. Of course, the optional data may be both integrity-protected and encrypted, preventing untrusted software from accessing the data.

[0068] In some embodiments, in block 826, a message authentication code for the authenticated response may be generated over the programming status using the random nonce provided by the invoking enclave in the BIND_STRUCT structure as the initialization vector, and using the encryption key included in the programming attempt. Thus, the invoking enclave includes a key (e.g., in the TKEY field of the BIND_STRUCT) with each programming attempt, even when programming a DMA channel out of secure. The response generated also includes a MAC over the encrypted response data and the status. As described above, the invoking enclave, on receiving this response, verifies the MAC using the random nonce and the key that it included with the original programming attempt. A successful verification of the MAC indicates that the programming attempt was sent to the cryptographic engine 140 through the UNWRAP instruction because only the unwrapping engine 418 may unwrap the EBIND_STRUCT, recover the key included with UNWRAP, and generate the MAC using this key included in the original programming attempt. The mechanism used also guarantees that the response cannot be modified by the untrusted software without detection, as the MAC verification by the invoking enclave will fail if there was an attempt to modify the authenticated response.

[0069] In block 828, the processor 120 determines whether channel programming was successful. The processor 120 may, for example, determine whether the BIND_STRUCT object was successfully unwrapped and whether the target-specific checks for the requested command were satisfied. If the channel programming was not successful, the method 800 advances to block 830, in which the processor 120 returns from executing the UNWRAP instruction and the method 800 is completed. As described above, the UNWRAP_RESPONSE_STRUCT object includes an authenticated response indicating that programming was unsuccessful.

[0070] Referring back to block 828, if the processor 120 determines that channel programming was successful, the method 800 branches to block 832, shown in FIG. 8B. In block 832, the processor 120 finds an appropriate entry in the CID table 142 for programming. As described above, the processor 120 may model the state of the CID table 142 based on the channel programming requests received. For example, the processor 120 may maintain a copy

of the contents of the CID table 142 in memory and update the copy in response to successful programming attempts. For a request to program a channel to secure, the processor 120 may search the model of the CID table 142 for an available entry. For a request to program a channel out of secure, the processor 120 may search the model of the CID table 142 for the entry corresponding to the specified CID.

[0071] In block 834, the processor 120 programs the unwrapped programming information to the selected entry of the CID table 142 of the cryptographic engine 140. The processor 120 may use any technique to program the cryptographic engine 140. For example, the processor 120 may set one or more registers of the request bank 202 maintained by the cryptographic engine 140 using a sideband interface that is unavailable to software executed by the processor 120. After programming, the CID table 142 of the cryptographic engine 140 is updated based on the requested command. In block 836, the processor 120 returns from executing the UNWRAP instruction and the method 800 is completed. As described above, when returning after a successful programming, the UNWRAP_RESPONSE_STRUCT object includes an authenticated response indicating that programming was successful.

[0072] Referring now to FIG. 9, pseudocode 900 illustrates one potential embodiment of the UNWRAP instruction. The pseudocode 900 may illustrate, for example, microcode of the processor 120 associated with the UNWRAP instruction. As shown, the processor 120 generates a temporary initialization vector based on the sequence ID field of the BIND_STRUCT object. The processor 120 generates an authentication header over several fields of the BIND_STRUCT, and then performs an AES-GCM authenticated encryption operation using the KWK, the temporary initialization vector, the authentication header, and the channel key. The processor 120 next compares the reference MAC generated by the AES-GCM authenticated encryption operation with the MAC provided in the BIND_STRUCT object. If those values match, the processor 120 proceeds to evaluate target-specific checks and program the cryptographic engine 140 as described in block 816. If those values do not match, the processor 120 indicates the unwrap failure in an error structure, such as the UNWRAP_RESPONSE_STRUCT.

[0073] It should be appreciated that, in some embodiments, the methods 500, 600, and/or 800 may be embodied as various instructions stored on a computer-readable media, which may be executed by the processor 120, the cryptographic engine 140, and/or other components of the computing device 100 to cause the computing device 100 to perform the corresponding method 500, 600, and/or 800. The computer-readable media may be embodied as any type of media capable of being read by the computing device 100 including, but not

limited to, the memory 132, the data storage device 134, microcode of the processor 120, memory of the cryptographic engine 140, firmware of the cryptographic engine 140, and/or other media.

EXAMPLES

[0074] Illustrative examples of the technologies disclosed herein are provided below. An embodiment of the technologies may include any one or more, and any combination of, the examples described below.

[0075] Example 1 includes a computing device for secure cryptographic engine programming, the computing device comprising: a secure programming module to (i) establish, by a processor of a computing device with secure enclave support, an invoking enclave, (ii) configure, by the invoking enclave, channel programming information, wherein the channel programming information includes a channel identifier and a channel key to be programmed to a cryptographic engine of the computing device, and (ii) invoke, by the invoking enclave, a processor instruction with the channel programming information as a parameter; and a binding module to generate, by the processor, wrapped programming information based on the channel programming information in response to invocation of the processor instruction, wherein the wrapped programming information comprises an encrypted channel key and a message authentication code.

[0076] Example 2 includes the subject matter of Example 1, and wherein the processor instruction comprises an EBINDTIO instruction.

[0077] Example 3 includes the subject matter of any of Examples 1 and 2, and wherein: the channel programming information comprises a binary structure indicative of the channel programming information; and to generate the wrapped programming information comprises to modify the binary structure to generate the wrapped programming information.

[0078] Example 4 includes the subject matter of any of Examples 1-3, and wherein to generate the wrapped programming information comprises to encrypt the channel key with a key wrapping key to generate the encrypted channel key, wherein the key wrapping key is a secret of the processor.

[0079] Example 5 includes the subject matter of any of Examples 1-4, and wherein to generate the wrapped programming information further comprises to generate the message authentication code with the key wrapping key.

[0080] Example 6 includes the subject matter of any of any of Examples 1-5, and wherein the secure programming module is further to provide, by the invoking enclave, the

wrapped programming information to an untrusted kernel mode component of the computing device.

[0081] Example 7 includes the subject matter of any of Examples 1-6, and wherein the untrusted kernel mode component comprises a crypto engine driver of the computing device.

[0082] Example 8 includes the subject matter of any of Examples 1-7, and wherein: the secure programming module is further to (i) receive, by the invoking enclave, an authenticated response from the untrusted kernel mode component in response to provision of the wrapped programming information to the untrusted kernel mode component, and (ii) verify, by the invoking enclave, the authenticated response with the channel key and a random nonce of the channel programming information; and to configure the channel programming information comprises to generate the random nonce.

[0083] Example 9 includes the subject matter of any of Examples 1-8, and further comprising: an unsecure programming module to invoke, by the untrusted kernel mode component, a second processor instruction with the wrapped programming information as a parameter in response to provision of the wrapped programming instruction to the untrusted kernel mode component; and an unwrapping engine module to (i) unwrap, by the processor, the wrapped programming information to generate the channel programming information in response to invocation of the second processor instruction, (ii) verify the channel programming information in response to the invocation of the second processor instruction, and (iii) program the channel key to the cryptographic engine in response to verification of the channel programming information.

[0084] Example 10 includes a computing device for secure cryptographic engine programming, the computing device comprising: an unsecure programming module to (i) receive, by an untrusted kernel mode component of a computing device, wrapped programming information, wherein the wrapped programming information includes an encrypted channel key to be programmed to a cryptographic engine of the computing device, and (ii) invoke, by the untrusted kernel mode component, a processor instruction with the wrapped programming information as a parameter; and an unwrapping engine module to (i) unwrap, by a processor of the computing device, the wrapped programming information to generate channel programming information in response to invocation of the processor instruction, wherein the channel programming information includes an unencrypted channel key, (ii) verify, by the processor, the channel programming information, and (iii) program, by the processor, the unencrypted channel key to the cryptographic engine in response to verification of the channel programming information.

[0085] Example 11 includes the subject matter of Example 10, and wherein the processor instruction comprises an UNWRAP instruction.

[0086] Example 12 includes the subject matter of any of Examples 10 and 11, and wherein the untrusted kernel mode component comprises a crypto engine driver of the computing device.

[0087] Example 13 includes the subject matter of any of Examples 10-12, and wherein to unwrap the wrapped programming information to generate the channel programming information comprises to decrypt the encrypted channel key with a key wrapping key to generate the unencrypted channel key, wherein the key wrapping key is a secret of the processor.

[0088] Example 14 includes the subject matter of any of Examples 10-13, and wherein: the unwrapping engine module is further to (i) determine whether the wrapped programming information is potentially replayed, and (ii) indicate, by the processor, an unwrapping error in response to a determination that the wrapped programming information is potentially replayed; wherein to unwrap the wrapped programming information comprises to unwrap the wrapped programming information in response to a determination that the wrapped programming information is not potentially replayed.

[0089] Example 15 includes the subject matter of any of Examples 10-14, and wherein to determine whether the wrapped programming information is potentially replayed comprises to: determine whether an invocation counter of the wrapped programming information has a predefined relationship to an internal invocation counter of the processor; and update the internal invocation counter with the invocation counter of the wrapped programming information in response to the verification of the channel programming information.

[0090] Example 16 includes the subject matter of any of Examples 10-15, and wherein to verify the channel programming information comprises to verify a message authentication code of the wrapped programming information with the channel programming information.

[0091] Example 17 includes the subject matter of any of Examples 10-16, and wherein: the unwrapping engine module is further to (i) determine, by the processor, whether a target-specific programming check is satisfied in response to the verification of the channel programming information, and (ii) indicate, by the processor, an error in response to determining that the target-specific programming check is not satisfied; wherein to program the unencrypted channel key to the cryptographic engine further comprising to program the unencrypted channel key to the cryptographic engine in response to a determination that the target-specific programming check is satisfied.

[0092] Example 18 includes the subject matter of any of Examples 10-17, and wherein: the channel programming information comprises a command to program a DMA channel to secure; and to determine whether the target-specific programming check is satisfied comprises to determine whether the DMA channel indicated by the channel programming information is already programmed, or to determine whether a channel identifier table of the crypto engine is full.

[0093] Example 19 includes the subject matter of any of Examples 10-18, and wherein: the channel programming information comprises a command to program a DMA channel out of secure; and to determine whether the target-specific programming check is satisfied comprises to determine whether the DMA channel is already programmed or to determine whether the unencrypted channel key matches a current channel key associated with the DMA channel.

[0094] Example 20 includes the subject matter of any of Examples 10-19, and wherein to program the unencrypted channel key to the cryptographic engine comprises to: determine an index of an available channel identifier table entry of the crypto engine with a crypto engine model maintained by the processor; and write the channel programming information at the index of the available channel identifier table entry.

[0095] Example 21 includes the subject matter of any of Examples 10-20, and wherein the unwrapping engine module is further to generate, by the processor, an authenticated response based on a programming status in response to the verification of the channel programming information.

[0096] Example 22 includes the subject matter of any of Examples 10-21, and wherein to generate the authenticated response comprises to generate a message authentication code over the programming status with the unencrypted channel key and a random nonce of the channel programming information.

[0097] Example 23 includes the subject matter of any of Examples 10-22, and wherein: the unsecure programming module is further to read, by the untrusted kernel mode component, the authenticated response in response to generation of the authenticated response; and the computing device further comprises a secure programming module to (i) establish, by the processor, an invoking enclave with secure enclave support of the processor, and (ii) verify, by the invoking enclave, the authenticated response in response to reading of the authenticated response.

[0098] Example 24 includes the subject matter of any of Examples 10-23, and wherein: the secure programming module is further to configure, by the invoking enclave, the channel programming information, wherein the channel programming information includes the

unencrypted channel key and a random nonce; wherein to verify the authenticated response comprises to verify the authenticated response with the channel programming key and the random nonce.

[0099] Example 25 includes the subject matter of any of Examples 10-24, and wherein: the secure programming module is further to invoke, by the invoking enclave, a second processor instruction with the channel programming information as a parameter; and the computing device further comprises a binding module to generate, by the processor, the wrapped programming information based on the channel programming information in response to invocation of the second processor instruction, wherein the wrapped programming information comprises the encrypted channel key and a message authentication code.

[00100] Example 26 includes a method for secure cryptographic engine programming, the method comprising: establishing, by a processor of a computing device having secure enclave support, an invoking enclave with the secure enclave support of the processor; configuring, by the invoking enclave, channel programming information, wherein the channel programming information includes a channel identifier and a channel key to be programmed to a cryptographic engine of the computing device; invoking, by the invoking enclave, a processor instruction with the channel programming information as a parameter; generating, by the processor, wrapped programming information based on the channel programming information in response to invoking the processor instruction, wherein the wrapped programming information comprises an encrypted channel key and a message authentication code.

[0100] Example 27 includes the subject matter of Example 26, and wherein invoking the processor instruction comprises invoking an EBINDTIO instruction.

[0101] Example 28 includes the subject matter of any of Examples 26 and 27, and wherein: configuring the channel programming information comprises configuring a binary structure indicative of the channel programming information; and generating the wrapped programming information comprises modifying the binary structure to generate the wrapped programming information.

[0102] Example 29 includes the subject matter of any of Examples 26-28, and wherein generating the wrapped programming information comprises encrypting the channel key with a key wrapping key to generate the encrypted channel key, wherein the key wrapping key is a secret of the processor.

[0103] Example 30 includes the subject matter of any of Examples 26-29, and wherein generating the wrapped programming information further comprises generating the message authentication code with the key wrapping key.

[0104] Example 31 includes the subject matter of any of Examples 26-30, and further comprising providing, by the invoking enclave, the wrapped programming information to an untrusted kernel mode component of the computing device.

[0105] Example 32 includes the subject matter of any of Examples 26-31, and wherein providing the wrapped programming information to the untrusted kernel mode component comprises providing the wrapped programming information to a crypto engine driver of the computing device.

[0106] Example 33 includes the subject matter of any of Examples 26-32, and further comprising: receiving, by the invoking enclave, an authenticated response from the untrusted kernel mode component in response to providing the wrapped programming information to the untrusted kernel mode component; and verifying, by the invoking enclave, the authenticated response using the channel key and a random nonce of the channel programming information; wherein configuring the channel programming information comprises generating the random nonce.

[0107] Example 34 includes the subject matter of any of Examples 26-33, and further comprising: invoking, by the untrusted kernel mode component, a second processor instruction with the wrapped programming information as a parameter in response to providing the wrapped programming instruction to the untrusted kernel mode component; unwrapping, by the processor, the wrapped programming information to generate the channel programming information in response to invoking the second processor instruction; verifying, by the processor, the channel programming information in response to invoking the second processor instruction; and programming, by the processor, the channel key to the cryptographic engine in response to verifying the channel programming information.

[0108] Example 35 includes a method for secure cryptographic engine programming, the method comprising: receiving, by an untrusted kernel mode component of a computing device, wrapped programming information, wherein the wrapped programming information includes an encrypted channel key to be programmed to a cryptographic engine of the computing device; invoking, by the untrusted kernel mode component, a processor instruction with the wrapped programming information as a parameter; unwrapping, by a processor of the computing device, the wrapped programming information to generate channel programming information in response to invoking the processor instruction, wherein the channel programming information includes an unencrypted channel key; verifying, by the processor, the channel programming information; and programming, by the processor, the unencrypted

channel key to the cryptographic engine in response to verifying the channel programming information.

[0109] Example 36 includes the subject matter of Example 35, and wherein invoking the processor instruction comprises invoking an UNWRAP instruction.

[0110] Example 37 includes the subject matter of any of Examples 35 and 36, and wherein the untrusted kernel mode component comprises a crypto engine driver of the computing device.

[0111] Example 38 includes the subject matter of any of Examples 35-37, and wherein unwrapping the wrapped programming information to generate the channel programming information comprises decrypting the encrypted channel key with a key wrapping key to generate the unencrypted channel key, wherein the key wrapping key is a secret of the processor.

[0112] Example 39 includes the subject matter of any of Examples 35-38, and further comprising: determining, by the processor, whether the wrapped programming information is potentially replayed; and indicating, by the processor, an unwrapping error in response to determining that the wrapped programming information is potentially replayed; wherein unwrapping the wrapped programming information comprises unwrapping the wrapped programming information in response to determining that the wrapped programming information is not potentially replayed.

[0113] Example 40 includes the subject matter of any of Examples 35-39, and wherein determining whether the wrapped programming information is potentially replayed comprises: determining whether an invocation counter of the wrapped programming information has a predefined relationship to an internal invocation counter of the processor; and updating the internal invocation counter with the invocation counter of the wrapped programming information in response to verifying the channel programming information.

[0114] Example 41 includes the subject matter of any of Examples 35-40, and wherein verifying the channel programming information comprises verifying a message authentication code of the wrapped programming information with the channel programming information.

[0115] Example 42 includes the subject matter of any of Examples 35-41, and further comprising: determining, by the processor, whether a target-specific programming check is satisfied in response to verifying the channel programming information; and indicating, by the processor, an error in response to determining that the target-specific programming check is not satisfied; wherein programming the unencrypted channel key to the cryptographic engine

further comprising programming the unencrypted channel key to the cryptographic engine in response to determining that the target-specific programming check is satisfied.

[0116] Example 43 includes the subject matter of any of Examples 35-42, and wherein: the channel programming information comprises a command to program a DMA channel to secure; and determining whether the target-specific programming check is satisfied comprises determining whether the DMA channel indicated by the channel programming information is already programmed or determining whether a channel identifier table of the crypto engine is full.

[0117] Example 44 includes the subject matter of any of Examples 35-43, and wherein: the channel programming information comprises a command to program a DMA channel out of secure; and determining whether the target-specific programming check is satisfied comprises determining whether the DMA channel is already programmed or determining whether the unencrypted channel key matches a current channel key associated with the DMA channel.

[0118] Example 45 includes the subject matter of any of Examples 35-44, and wherein programming the unencrypted channel key to the cryptographic engine comprises: determining an index of an available channel identifier table entry of the crypto engine using a crypto engine model maintained by the processor; and writing the channel programming information at the index of the available channel identifier table entry.

[0119] Example 46 includes the subject matter of any of Examples 35-45, and further comprising generating, by the processor, an authenticated response based on a programming status in response to verifying the channel programming information.

[0120] Example 47 includes the subject matter of any of Examples 35-46, and wherein generating the authenticated response comprises generating a message authentication code over the programming status using the unencrypted channel key and a random nonce of the channel programming information.

[0121] Example 48 includes the subject matter of any of Examples 35-47, and further comprising: establishing, by the processor, an invoking enclave using secure enclave support of the processor; reading, by the untrusted kernel mode component, the authenticated response in response to generating the authenticated response; and verifying, by the invoking enclave, the authenticated response in response to reading the authenticated response.

[0122] Example 49 includes the subject matter of any of Examples 35-48, and further comprising: configuring, by the invoking enclave, the channel programming information, wherein the channel programming information includes the unencrypted channel key and a

random nonce; wherein verifying the authenticated response comprises verifying the authenticated response using the channel programming key and the random nonce.

[0123] Example 50 includes the subject matter of any of Examples 35-49, and further comprising: invoking, by the invoking enclave, a second processor instruction with the channel programming information as a parameter; and generating, by the processor, the wrapped programming information based on the channel programming information in response to invoking the second processor instruction, wherein the wrapped programming information comprises the encrypted channel key and a message authentication code.

[0124] Example 51 includes a computing device comprising: a processor; and a memory having stored therein a plurality of instructions that when executed by the processor cause the computing device to perform the method of any of Examples 26-50.

[0125] Example 52 includes one or more machine readable storage media comprising a plurality of instructions stored thereon that in response to being executed result in a computing device performing the method of any of Examples 26-50.

[0126] Example 53 includes a computing device comprising means for performing the method of any of Examples 26-50.

[0127] Example 54 includes a computing device for secure cryptographic engine programming, the computing device comprising: means for establishing, by a processor of a computing device having secure enclave support, an invoking enclave with the secure enclave support of the processor; means for configuring, by the invoking enclave, channel programming information, wherein the channel programming information includes a channel identifier and a channel key to be programmed to a cryptographic engine of the computing device; means for invoking, by the invoking enclave, a processor instruction with the channel programming information as a parameter; means for generating, by the processor, wrapped programming information based on the channel programming information in response to invoking the processor instruction, wherein the wrapped programming information comprises an encrypted channel key and a message authentication code.

[0128] Example 55 includes the subject matter of Example 54, and wherein the means for invoking the processor instruction comprises means for invoking an EBINDTIO instruction.

[0129] Example 56 includes the subject matter of any of Examples 54 and 55, and wherein: the means for configuring the channel programming information comprises means for configuring a binary structure indicative of the channel programming information; and the means for generating the wrapped programming information comprises means for modifying the binary structure to generate the wrapped programming information.

[0130] Example 57 includes the subject matter of any of Examples 54-56, and wherein the means for generating the wrapped programming information comprises means for encrypting the channel key with a key wrapping key to generate the encrypted channel key, wherein the key wrapping key is a secret of the processor.

[0131] Example 58 includes the subject matter of any of Examples 54-57, and wherein the means for generating the wrapped programming information further comprises means for generating the message authentication code with the key wrapping key.

[0132] Example 59 includes the subject matter of any of Examples 54-58, and further comprising means for providing, by the invoking enclave, the wrapped programming information to an untrusted kernel mode component of the computing device.

[0133] Example 60 includes the subject matter of any of Examples 54-59, and wherein the means for providing the wrapped programming information to the untrusted kernel mode component comprises means for providing the wrapped programming information to a crypto engine driver of the computing device.

[0134] Example 61 includes the subject matter of any of Examples 54-60, and further comprising: means for receiving, by the invoking enclave, an authenticated response from the untrusted kernel mode component in response to providing the wrapped programming information to the untrusted kernel mode component; and means for verifying, by the invoking enclave, the authenticated response using the channel key and a random nonce of the channel programming information; wherein the means for configuring the channel programming information comprises means for generating the random nonce.

[0135] Example 62 includes the subject matter of any of Examples 54-61, and further comprising: means for invoking, by the untrusted kernel mode component, a second processor instruction with the wrapped programming information as a parameter in response to providing the wrapped programming instruction to the untrusted kernel mode component; means for unwrapping, by the processor, the wrapped programming information to generate the channel programming information in response to invoking the second processor instruction; means for verifying, by the processor, the channel programming information in response to invoking the second processor instruction; and means for programming, by the processor, the channel key to the cryptographic engine in response to verifying the channel programming information.

[0136] Example 63 includes a computing device for secure cryptographic engine programming, the computing device comprising: means for receiving, by an untrusted kernel mode component of a computing device, wrapped programming information, wherein the wrapped programming information includes an encrypted channel key to be programmed to a

cryptographic engine of the computing device; means for invoking, by the untrusted kernel mode component, a processor instruction with the wrapped programming information as a parameter; means for unwrapping, by a processor of the computing device, the wrapped programming information to generate channel programming information in response to invoking the processor instruction, wherein the channel programming information includes an unencrypted channel key; means for verifying, by the processor, the channel programming information; and means for programming, by the processor, the unencrypted channel key to the cryptographic engine in response to verifying the channel programming information.

[0137] Example 64 includes the subject matter of Example 63, and wherein the means for invoking the processor instruction comprises means for invoking an UNWRAP instruction.

[0138] Example 65 includes the subject matter of any of Examples 63 and 64, and wherein the untrusted kernel mode component comprises a crypto engine driver of the computing device.

[0139] Example 66 includes the subject matter of any of Examples 63-65, and wherein the means for unwrapping the wrapped programming information to generate the channel programming information comprises means for decrypting the encrypted channel key with a key wrapping key to generate the unencrypted channel key, wherein the key wrapping key is a secret of the processor.

[0140] Example 67 includes the subject matter of any of Examples 63-66, and further comprising: means for determining, by the processor, whether the wrapped programming information is potentially replayed; and means for indicating, by the processor, an unwrapping error in response to determining that the wrapped programming information is potentially replayed; wherein the means for unwrapping the wrapped programming information comprises means for unwrapping the wrapped programming information in response to determining that the wrapped programming information is not potentially replayed.

[0141] Example 68 includes the subject matter of any of Examples 63-67, and wherein the means for determining whether the wrapped programming information is potentially replayed comprises: means for determining whether an invocation counter of the wrapped programming information has a predefined relationship to an internal invocation counter of the processor; and means for updating the internal invocation counter with the invocation counter of the wrapped programming information in response to verifying the channel programming information.

[0142] Example 69 includes the subject matter of any of Examples 63-68, and wherein the means for verifying the channel programming information comprises means for verifying a

message authentication code of the wrapped programming information with the channel programming information.

[0143] Example 70 includes the subject matter of any of Examples 63-69, and further comprising: means for determining, by the processor, whether a target-specific programming check is satisfied in response to verifying the channel programming information; and means for indicating, by the processor, an error in response to determining that the target-specific programming check is not satisfied; wherein the means for programming the unencrypted channel key to the cryptographic engine further comprising means for programming the unencrypted channel key to the cryptographic engine in response to determining that the target-specific programming check is satisfied.

[0144] Example 71 includes the subject matter of any of Examples 63-70, and wherein: the channel programming information comprises a command to program a DMA channel to secure; and the means for determining whether the target-specific programming check is satisfied comprises means for determining whether the DMA channel indicated by the channel programming information is already programmed or determining whether a channel identifier table of the crypto engine is full.

[0145] Example 72 includes the subject matter of any of Examples 63-71, and wherein: the channel programming information comprises a command to program a DMA channel out of secure; and the means for determining whether the target-specific programming check is satisfied comprises means for determining whether the DMA channel is already programmed or determining whether the unencrypted channel key matches a current channel key associated with the DMA channel.

[0146] Example 73 includes the subject matter of any of Examples 63-72, and wherein the means for programming the unencrypted channel key to the cryptographic engine comprises: means for determining an index of an available channel identifier table entry of the crypto engine using a crypto engine model maintained by the processor; and means for writing the channel programming information at the index of the available channel identifier table entry.

[0147] Example 74 includes the subject matter of any of Examples 63-73, and further comprising means for generating, by the processor, an authenticated response based on a programming status in response to verifying the channel programming information.

[0148] Example 75 includes the subject matter of any of Examples 63-74, and wherein the means for generating the authenticated response comprises means for generating a message

authentication code over the programming status using the unencrypted channel key and a random nonce of the channel programming information.

[0149] Example 76 includes the subject matter of any of Examples 63-75, and further comprising: means for establishing, by the processor, an invoking enclave using secure enclave support of the processor; means for reading, by the untrusted kernel mode component, the authenticated response in response to generating the authenticated response; and means for verifying, by the invoking enclave, the authenticated response in response to reading the authenticated response.

[0150] Example 77 includes the subject matter of any of Examples 63-76, and further comprising: means for configuring, by the invoking enclave, the channel programming information, wherein the channel programming information includes the unencrypted channel key and a random nonce; wherein the means for verifying the authenticated response comprises means for verifying the authenticated response using the channel programming key and the random nonce.

[0151] Example 78 includes the subject matter of any of Examples 63-77, and further comprising: means for invoking, by the invoking enclave, a second processor instruction with the channel programming information as a parameter; and means for generating, by the processor, the wrapped programming information based on the channel programming information in response to invoking the second processor instruction, wherein the wrapped programming information comprises the encrypted channel key and a message authentication code.

WHAT IS CLAIMED IS:

1. A computing device for secure cryptographic engine programming, the computing device comprising:

a secure programming module to (i) establish, by a processor of a computing device with secure enclave support, an invoking enclave, (ii) configure, by the invoking enclave, channel programming information, wherein the channel programming information includes a channel identifier and a channel key to be programmed to a cryptographic engine of the computing device, and (ii) invoke, by the invoking enclave, a processor instruction with the channel programming information as a parameter; and

a binding module to generate, by the processor, wrapped programming information based on the channel programming information in response to invocation of the processor instruction, wherein the wrapped programming information comprises an encrypted channel key and a message authentication code.

2. The computing device of claim 1, wherein the processor instruction comprises an EBINDTIO instruction.

3. The computing device of claim 1, wherein to generate the wrapped programming information comprises to encrypt the channel key with a key wrapping key to generate the encrypted channel key, wherein the key wrapping key is a secret of the processor.

4. The computing device of claim 1, wherein to generate the wrapped programming information further comprises to generate the message authentication code with the key wrapping key.

5. The computing device of any of claims 1-4, wherein the secure programming module is further to provide, by the invoking enclave, the wrapped programming information to an untrusted kernel mode component of the computing device.

6. The computing device of claim 5, wherein:

the secure programming module is further to (i) receive, by the invoking enclave, an authenticated response from the untrusted kernel mode component in response to provision of the wrapped programming information to the untrusted kernel mode component,

and (ii) verify, by the invoking enclave, the authenticated response with the channel key and a random nonce of the channel programming information; and

to configure the channel programming information comprises to generate the random nonce.

7. A computing device for secure cryptographic engine programming, the computing device comprising:

an unsecure programming module to (i) receive, by an untrusted kernel mode component of a computing device, wrapped programming information, wherein the wrapped programming information includes an encrypted channel key to be programmed to a cryptographic engine of the computing device, and (ii) invoke, by the untrusted kernel mode component, a processor instruction with the wrapped programming information as a parameter; and

an unwrapping engine module to (i) unwrap, by a processor of the computing device, the wrapped programming information to generate channel programming information in response to invocation of the processor instruction, wherein the channel programming information includes an unencrypted channel key, (ii) verify, by the processor, the channel programming information, and (iii) program, by the processor, the unencrypted channel key to the cryptographic engine in response to verification of the channel programming information.

8. The computing device of claim 7, wherein the processor instruction comprises an UNWRAP instruction.

9. The computing device of claim 7, wherein to unwrap the wrapped programming information to generate the channel programming information comprises to decrypt the encrypted channel key with a key wrapping key to generate the unencrypted channel key, wherein the key wrapping key is a secret of the processor.

10. The computing device of claim 7, wherein:

the unwrapping engine module is further to (i) determine whether the wrapped programming information is potentially replayed, and (ii) indicate, by the processor, an unwrapping error in response to a determination that the wrapped programming information is potentially replayed;

wherein to unwrap the wrapped programming information comprises to unwrap the wrapped programming information in response to a determination that the wrapped programming information is not potentially replayed.

11. The computing device of claim 10, wherein to determine whether the wrapped programming information is potentially replayed comprises to:

determine whether an invocation counter of the wrapped programming information has a predefined relationship to an internal invocation counter of the processor; and

update the internal invocation counter with the invocation counter of the wrapped programming information in response to the verification of the channel programming information.

12. The computing device of any of claims 7-11, wherein:

the unwrapping engine module is further to (i) determine, by the processor, whether a target-specific programming check is satisfied in response to the verification of the channel programming information, and (ii) indicate, by the processor, an error in response to determining that the target-specific programming check is not satisfied;

wherein to program the unencrypted channel key to the cryptographic engine further comprising to program the unencrypted channel key to the cryptographic engine in response to a determination that the target-specific programming check is satisfied.

13. The computing device of any of claims 7-11, wherein to program the unencrypted channel key to the cryptographic engine comprises to:

determine an index of an available channel identifier table entry of the crypto engine with a crypto engine model maintained by the processor; and

write the channel programming information at the index of the available channel identifier table entry.

14. The computing device of any of claims 7-11, wherein the unwrapping engine module is further to generate, by the processor, an authenticated response based on a programming status in response to the verification of the channel programming information.

15. The computing device of claim 14, wherein:

the unsecure programming module is further to read, by the untrusted kernel mode component, the authenticated response in response to generation of the authenticated response; and

the computing device further comprises a secure programming module to (i) establish, by the processor, an invoking enclave with secure enclave support of the processor, and (ii) verify, by the invoking enclave, the authenticated response in response to reading of the authenticated response.

16. A method for secure cryptographic engine programming, the method comprising:

establishing, by a processor of a computing device having secure enclave support, an invoking enclave with the secure enclave support of the processor;

configuring, by the invoking enclave, channel programming information, wherein the channel programming information includes a channel identifier and a channel key to be programmed to a cryptographic engine of the computing device;

invoking, by the invoking enclave, a processor instruction with the channel programming information as a parameter;

generating, by the processor, wrapped programming information based on the channel programming information in response to invoking the processor instruction, wherein the wrapped programming information comprises an encrypted channel key and a message authentication code.

17. The method of claim 16, wherein generating the wrapped programming information comprises encrypting the channel key with a key wrapping key to generate the encrypted channel key, wherein the key wrapping key is a secret of the processor.

18. The method of claim 16, wherein generating the wrapped programming information further comprises generating the message authentication code with the key wrapping key.

19. A method for secure cryptographic engine programming, the method comprising:

receiving, by an untrusted kernel mode component of a computing device, wrapped programming information, wherein the wrapped programming information includes an encrypted channel key to be programmed to a cryptographic engine of the computing device;

invoking, by the untrusted kernel mode component, a processor instruction with the wrapped programming information as a parameter;

unwrapping, by a processor of the computing device, the wrapped programming information to generate channel programming information in response to invoking the processor instruction, wherein the channel programming information includes an unencrypted channel key;

verifying, by the processor, the channel programming information; and

programming, by the processor, the unencrypted channel key to the cryptographic engine in response to verifying the channel programming information.

20. The method of claim 19, wherein unwrapping the wrapped programming information to generate the channel programming information comprises decrypting the encrypted channel key with a key wrapping key to generate the unencrypted channel key, wherein the key wrapping key is a secret of the processor.

21. The method of claim 19, further comprising:

determining, by the processor, whether the wrapped programming information is potentially replayed; and

indicating, by the processor, an unwrapping error in response to determining that the wrapped programming information is potentially replayed;

wherein unwrapping the wrapped programming information comprises unwrapping the wrapped programming information in response to determining that the wrapped programming information is not potentially replayed.

22. The method of claim 21, wherein determining whether the wrapped programming information is potentially replayed comprises:

determining whether an invocation counter of the wrapped programming information has a predefined relationship to an internal invocation counter of the processor; and

updating the internal invocation counter with the invocation counter of the wrapped programming information in response to verifying the channel programming information.

23. A computing device comprising:

a processor; and

a memory having stored therein a plurality of instructions that when executed by the processor cause the computing device to perform the method of any of claims 16-22.

24. One or more machine readable storage media comprising a plurality of instructions stored thereon that in response to being executed result in a computing device performing the method of any of claims 16-22.

25. A computing device comprising means for performing the method of any of claims 16-22.

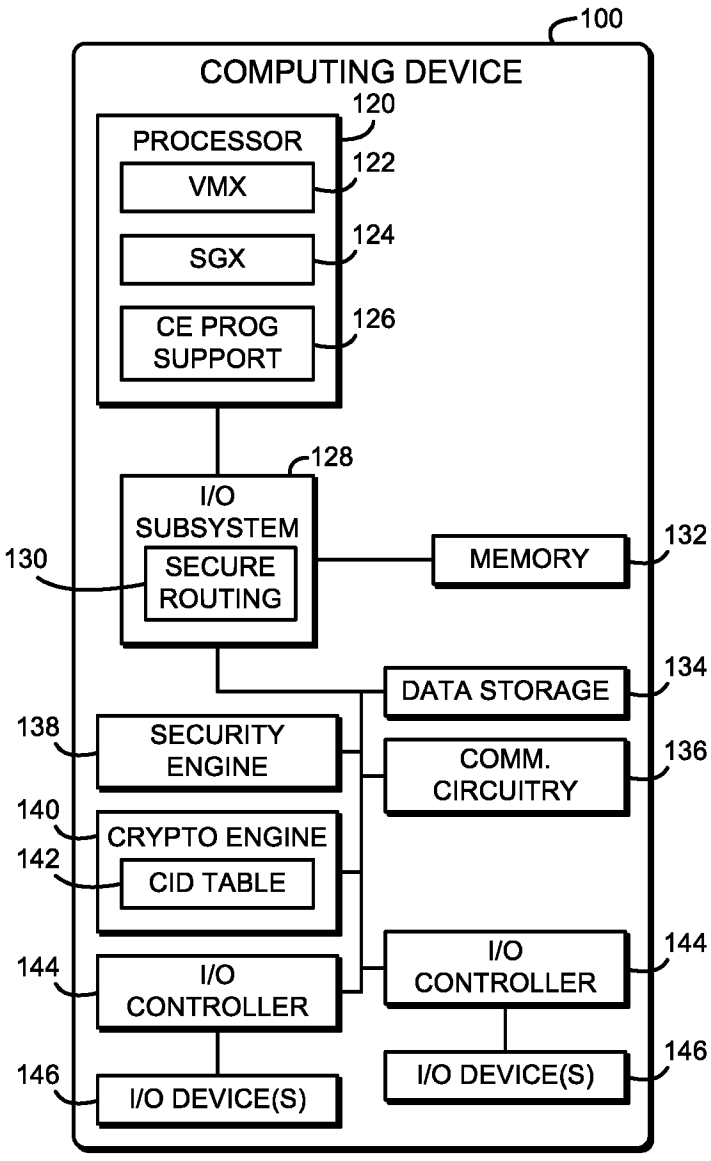
100
COMPUTING DEVICE
120
PROCESSOR
122
VMX
124
SGX
126
CE PROG SUPPORT
128
I/O SUBSYSTEM
130
SECURE ROUTING
MEMORY
132
DATA STORAGE
134
138
SECURITY ENGINE
COMM. CIRCUITRY
136
140
CRYPTO ENGINE
142
CID TABLE
144
I/O CONTROLLER
144
I/O CONTROLLER
146
I/O DEVICE(S)
146
I/O DEVICE(S)

FIG. 1

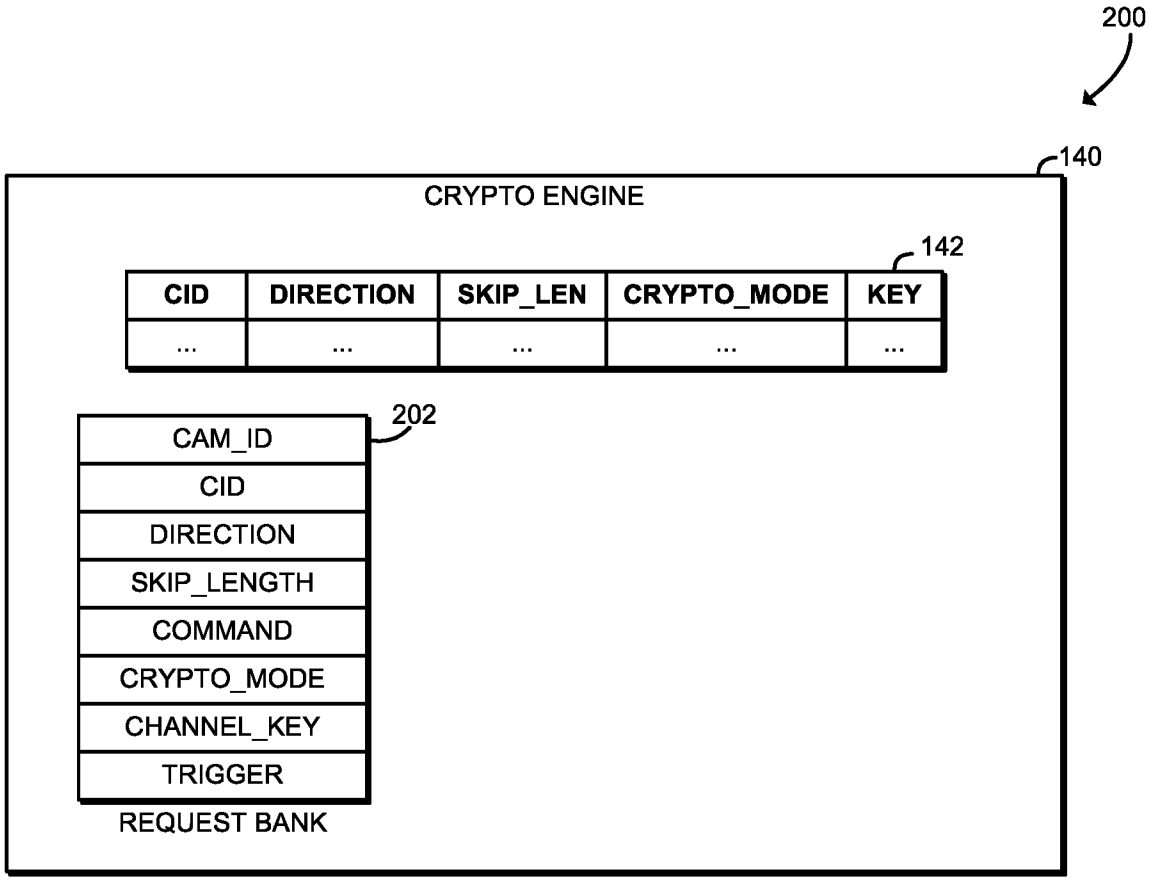
200
140
CRYPTO ENGINE
142
CID
DIRECTION
SKIP_LEN
CRYPTO_MODE
KEY
...
...
...
...
...
202
CAM_ID
CID
DIRECTION
SKIP_LENGTH
COMMAND
CRYPTO_MODE
CHANNEL_KEY
TRIGGER
REQUEST BANK

FIG. 2

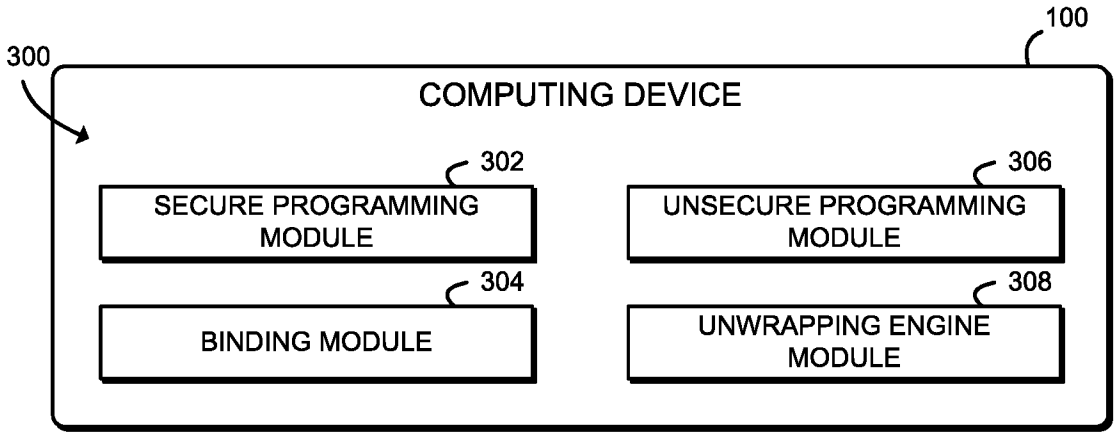
100
300
COMPUTING DEVICE
302
SECURE PROGRAMMING MODULE
306
UNSECURE PROGRAMMING MODULE
304
BINDING MODULE
308
UNWRAPPING ENGINE MODULE

FIG. 3

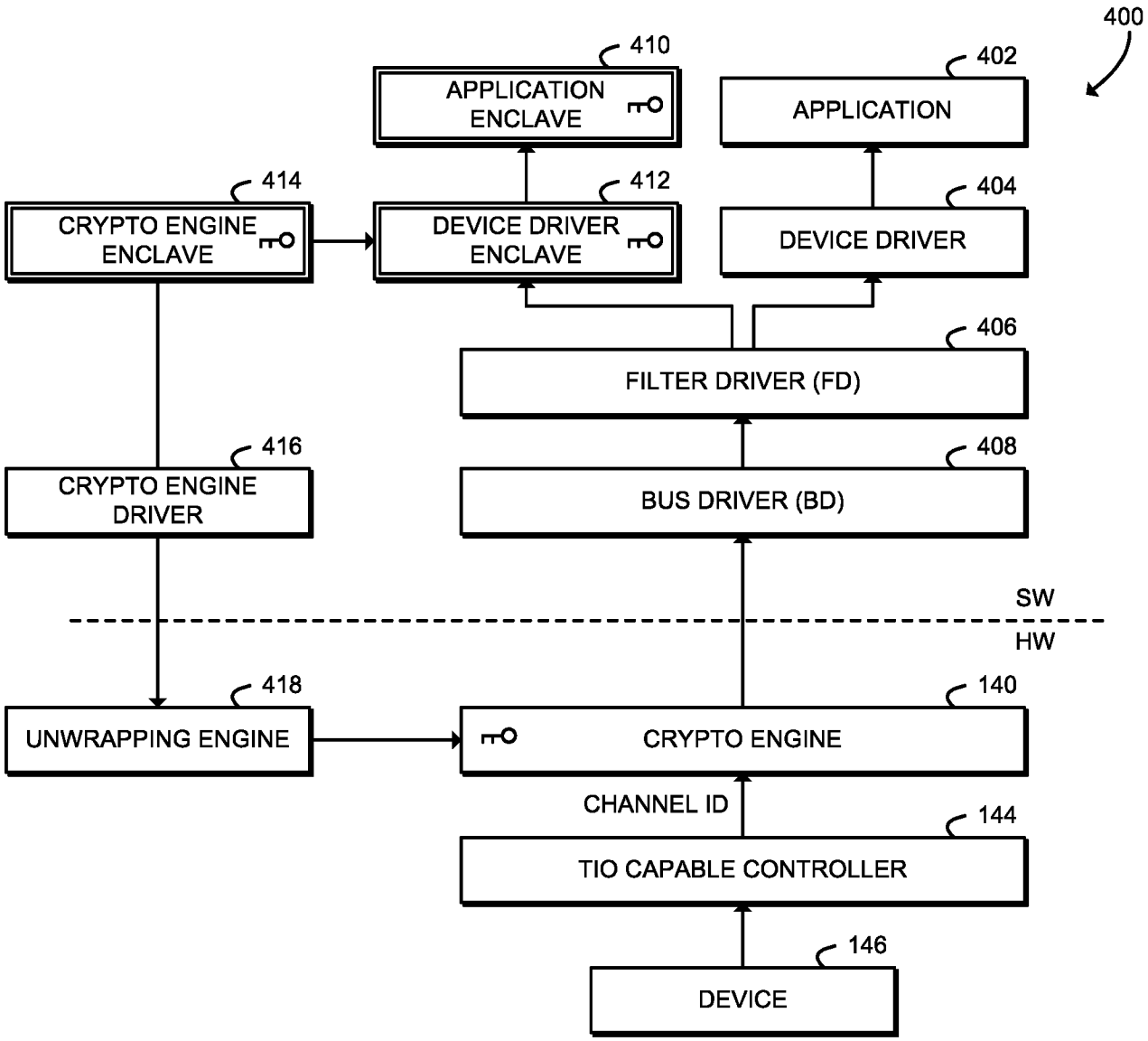
400
410
APPLICATION ENCLAVE
402
APPLICATION
414
CRYPTO ENGINE ENCLAVE
412
DEVICE DRIVER ENCLAVE
404
DEVICE DRIVER
406
FILTER DRIVER (FD)
416
CRYPTO ENGINE DRIVER
408
BUS DRIVER (BD)
SW
HW
418
UNWRAPPING ENGINE
140
CRYPTO ENGINE
CHANNEL ID
144
TIO CAPABLE CONTROLLER
146
DEVICE

FIG. 4

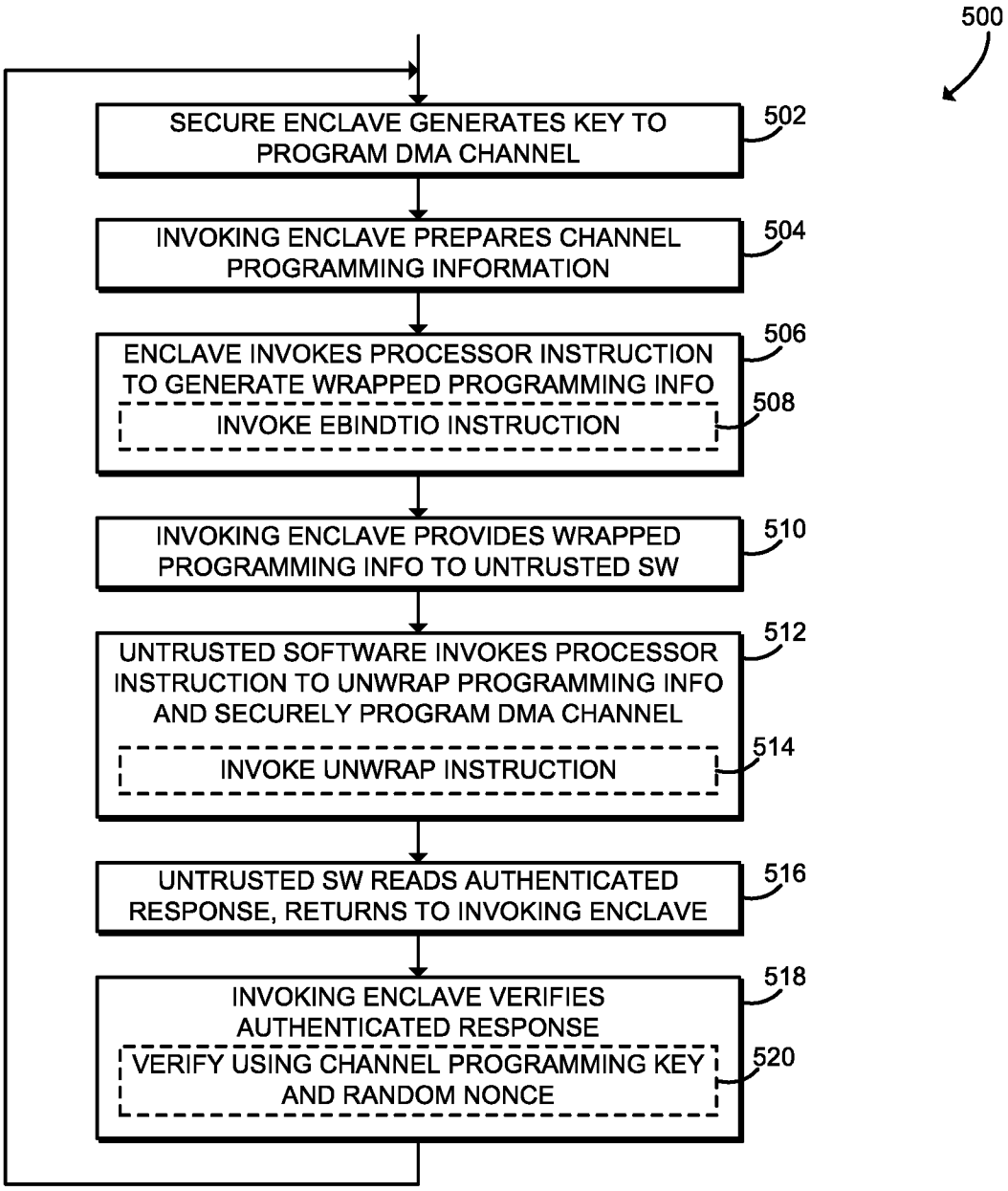
500
SECURE ENCLAVE GENERATES KEY TO PROGRAM DMA CHANNEL
502
INVOKING ENCLAVE PREPARES CHANNEL PROGRAMMING INFORMATION
504
ENCLAVE INVOKES PROCESSOR INSTRUCTION TO GENERATE WRAPPED PROGRAMMING INFO
506
INVOKE EBINDTIO INSTRUCTION
508
INVOKING ENCLAVE PROVIDES WRAPPED PROGRAMMING INFO TO UNTRUSTED SW
510
UNTRUSTED SOFTWARE INVOKES PROCESSOR INSTRUCTION TO UNWRAP PROGRAMMING INFO AND SECURELY PROGRAM DMA CHANNEL
512
INVOKE UNWRAP INSTRUCTION
514
UNTRUSTED SW READS AUTHENTICATED RESPONSE, RETURNS TO INVOKING ENCLAVE
516
INVOKING ENCLAVE VERIFIES AUTHENTICATED RESPONSE
518
VERIFY USING CHANNEL PROGRAMMING KEY AND RANDOM NONCE
520

FIG. 5

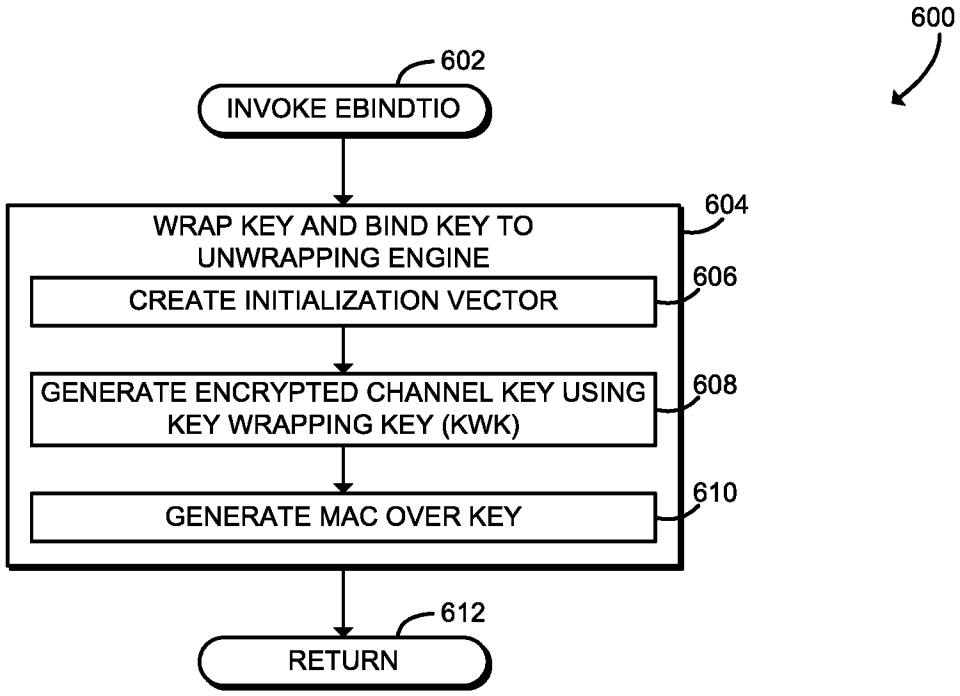

FIG. 6

700

```
TMP_SEQID := Sample from a 64-bit monotonic counter
        maintained by hardware
TMP_IV := TMP_SEQID || {63'b0} || 1'b1
BIND_STRUCT.SEQID := TMP_SEQID
AUTH_HEADER := BIND_STRUCT.BTID || BIND_STRUCT.BTSVN ||
          BIND_STRUCT.BTPOLICY || BIND_STRUCT.CID_IP ||
          BIND_STRUCT.NONCE || BIND_STRUCT.SEQID ||
          BIND_STRUCTUE.COMMAND
<CT, MAC>:= AES_GCM(KWK, TMP_IV, AUTH_HEADER, TKEY)
BIND_STRUCT.TKEY := CT
BIND_STRUCT.MAC := MAC
```

FIG. 7

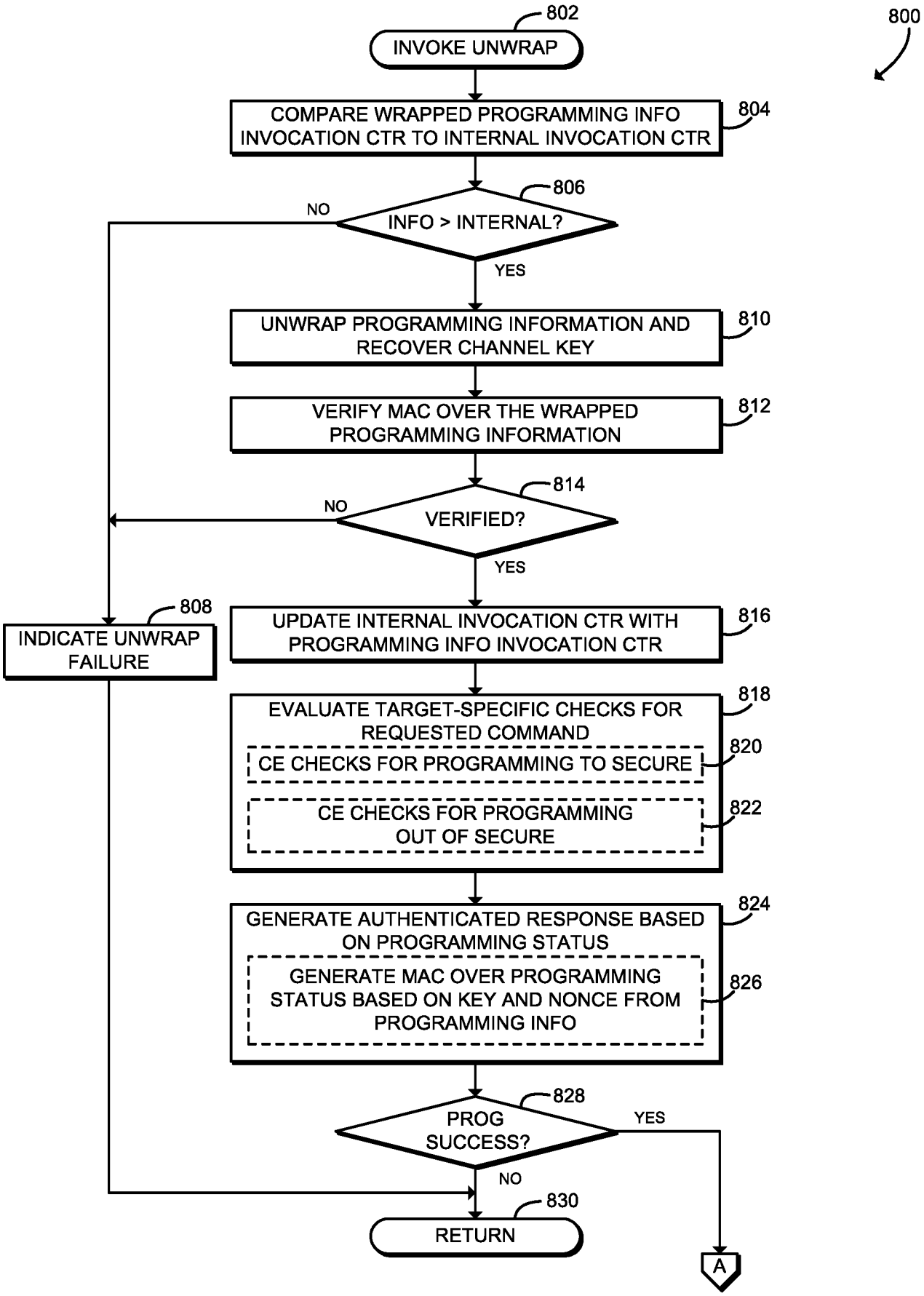
800
802
INVOKE UNWRAP
COMPARE WRAPPED PROGRAMMING INFO INVOCATION CTR TO INTERNAL INVOCATION CTR
804
806
INFO > INTERNAL?
NO
YES
UNWRAP PROGRAMMING INFORMATION AND RECOVER CHANNEL KEY
810
VERIFY MAC OVER THE WRAPPED PROGRAMMING INFORMATION
812
814
VERIFIED?
NO
YES
808
INDICATE UNWRAP FAILURE
UPDATE INTERNAL INVOCATION CTR WITH PROGRAMMING INFO INVOCATION CTR
816
EVALUATE TARGET-SPECIFIC CHECKS FOR REQUESTED COMMAND
818
CE CHECKS FOR PROGRAMMING TO SECURE
820
CE CHECKS FOR PROGRAMMING OUT OF SECURE
822
GENERATE AUTHENTICATED RESPONSE BASED ON PROGRAMMING STATUS
824
GENERATE MAC OVER PROGRAMMING STATUS BASED ON KEY AND NONCE FROM PROGRAMMING INFO
826
828
PROG SUCCESS?
YES
NO
830
RETURN
A

FIG. 8A

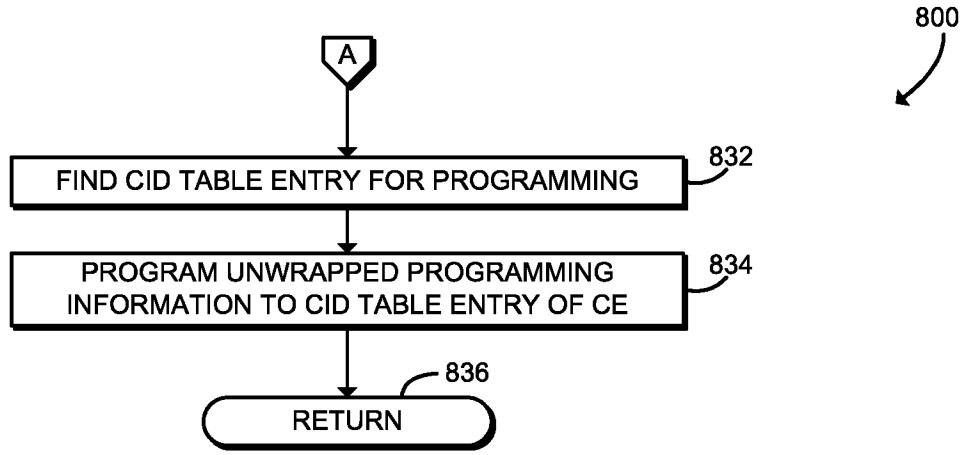

FIG. 8B

900

```
TMP_IV := BIND_STRUCT.SEQID || {63'b0} || 1'b1
AUTH_HEADER := BIND_STRUCT.BTID || BIND_STRUCT.BTSVN ||
          EBIND_STRUCT.BTDATA || EBIND_STRUCT.NONCE ||
          EBIND_STRUCT.SEQID ||
<PT, REF_MAC>:= AES_GCM(KWK, TMP_IV, AUTH_HEADER,
           EBIND_STRUCT.TKEY)
Compare REF_MAC with EBIND_STRUCT.MAC. If the two match,
proceed to block 816, else indicate the unwrap failure in
ERROR_STRUCT
```

FIG. 9

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2016/038396

A. CLASSIFICATION OF SUBJECT MATTER

G06F 21/60(2013.01)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 21/60; G06F 3/00; H04L 9/32; H04L 29/06; G06Q 20/38; G06F 21/22; G06F 21/00; G06F 13/00

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
Korean utility models and applications for utility models
Japanese utility models and applications for utility models

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
eKOMPASS(KIPO internal) & keywords: trusted I/O, secure programming module, binding module, secure enclave support, channel identifier, channel key, programming information

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 9059855 B2 (INTEL CORPORATION) 16 June 2015 See column 4, line 18 - column 6, line 48; claims 2, 10; and figures 2, 4.	1-25
A	US 2011-0225431 A1 (KENNETH W. STUFFLEBEAM, JR. et al.) 15 September 2011 See paragraphs [0032]-[0033], [0036], [0040]-[0045]; and claims 1, 6.	1-25
A	US 9058494 B2 (INTEL CORPORATION) 16 June 2015 See claims 1-6; and figures 3B-4B.	1-25
A	US 2014-0188732 A1 (NCR CORPORATION) 03 July 2014 See paragraphs [0029]-[0042]; claims 1-3, 20; and figures 2A, 3.	1-25
A	US 7716389 B1 (BRUCE REY et al.) 11 May 2010 See column 3, line 33 - column 4, line 46; and figures 2-3.	1-25

☐ 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
22 September 2016 (22.09.2016)	**23 September 2016 (23.09.2016)**

Name and mailing address of the ISA/KR
International Application Division
Korean Intellectual Property Office
189 Cheongsa-ro, Seo-gu, Daejeon, 35208, Republic of Korea
Facsimile No. +82-42-481-8578

Authorized officer
CHIN, Sang Bum
Telephone No. +82-42-481-8398

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US2016/038396

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 9059855 B2	16/06/2015	US 2012-0163589 A1	28/06/2012
		US 2013-0232345 A1	05/09/2013
		US 2015-0186680 A1	02/07/2015
		US 8832452 B2	09/09/2014
		WO 2012-087562 A2	28/06/2012
		WO 2012-087562 A3	16/08/2012
US 2011-0225431 A1	15/09/2011	US 2015-0106633 A1	16/04/2015
		US 8930713 B2	06/01/2015
		US 9298938 B2	29/03/2016
US 9058494 B2	16/06/2015	CN 105009134 A	28/10/2015
		EP 2973154 A1	20/01/2016
		KR 10-2015-0105983 A	18/09/2015
		US 2014-0283093 A1	18/09/2014
		WO 2014-143671 A1	18/09/2014
US 2014-0188732 A1	03/07/2014	None	
US 7716389 B1	11/05/2010	None	