WHITE PAPER

Intel® Software Guard Extensions (Intel® SGX)

Trusted Time and Monotonic Counters with Intel® Software Guard Extensions Platform Services

Shanwei Cen, Bo Zhang

Intel Corporation

1 Introduction

Intel® Software Guard Extensions (Intel® SGX) is an Intel® CPU based Trusted Execution Environment (TEE) technology. It consists of a set of processor extensions that allow a user-space application to create a Trusted Computing Base (TCB) called an enclave in its address space [6] [7]. An enclave has the CPU package boundary as its security perimeter and provides confidentiality and integrity protection, even in the presence of privileged malware or external memory bus snoops. Intel SGX provides support of enclave attestation to a 3rd party service, so the latter can verify the security properties of the Intel CPU and the enclave software before provisioning secrets. Intel SGX allows an enclave to seal its secrets using a hardware-derived sealing key that is unique to the CPU and the enclave identities.

Many security usages can be implemented using Intel SGX technology to protect their secrets. Examples include digital rights management, online banking and e-commerce, protection of private keys for secure communication, protection of symmetric keys for disk encryption, etc.

However, some security usages require other security capabilities in addition to protection of their secrets. For example, digital rights management not only requires protection of its encryption keys in its media playback licenses, but also requires trusted time and monotonic counters in order to enforce expiration date and playback count limits in its license policies.

Trusted services such as timer and monotonic counters are not supported by the Intel SGX technology itself. They need to be provided by a separate TEE, and securely made available to Intel SGX enclaves. The Intel SGX software provides an implementation of these services, called Intel SGX Platform Services, leveraging the security capabilities of the Intel Converged Security and Management Engine (CSME). The CSME is an embedded engine running its dedicated firmware in the Platform Control Hub (PCH) on Intel platforms, and is separate from the CPU [5], though on some Intel platforms the PCH and the CPU are physically in the same package. The Intel SGX software consists of two packages: the Platform Software (PSW) with architectural enclaves, drivers, and user-space service applications; and the SDK with libraries and software development tools. For convenience, in this paper, when there is no ambiguity, the PSW and the SDK are collectively referred to as the Intel SGX SDK.

This paper explains the Intel SGX Platform Services implemented with the CSME. It covers the Intel SGX SDK version 1.8 for Linux* OS and Microsoft* Windows* OS on the 6th and 7th generations of Intel SGX-enabled processors. Section 2 provides an overview of the supported platform services, architecture,

and software / firmware stack. Section 3 introduces the APIs in the Intel SGX SDK for setting up and using the trusted platform services. Section 4 presents an in-depth discussion on architecture topics including how the components establish their identities and how they verify mutual trust and establishes secure sessions between them. Finally Section 5 summarizes and wraps up the paper.

2 Overview

2.1 Trusted Platform Services

The trusted platform services implemented by the Intel SGX Platform Services software and made available to application enclaves through the Intel SGX SDK include the following two features:

- Trusted time – for timer-based policy enforcement on offline platforms that don't have access to remote time servers for trusted calendar time.
- Monotonic counter – for replay protection of offline storage, enforcement of counter-based policies, etc.

The Intel SGX Platform Services architecture and the Intel SGX SDK implementation protect the integrity of the platform services, and provide mechanisms for application enclaves to detect in case the integrity of the services is disrupted, for example, when the battery powered real-time clock is reset.

2.2 Architecture Overview

Figure 1 shows a high-level architecture diagram of the Intel SGX Platform Services, including the TCB components and the relationship between them.

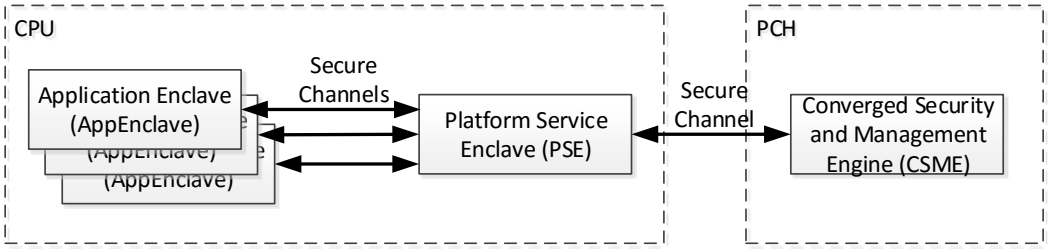

Figure 1 Platform Services Overall Architecture

The CSME in the PCH provides a set of security capabilities including battery backed real-time-clock (RTC) and replay-protected storage, and exposes an interface for trusted software running in the CPU to securely access these capabilities.

On the CPU side, the Platform Service Enclave (PSE) implements the trusted time and monotonic counter services by leveraging the CSME security capabilities. It enables a large number of application enclaves to access the platform services, in spite of limited resources in the CSME.

The PSE and the CSME use a key-exchange protocol called SIGMA to pair with each other (to verify mutual trust and establish shared secrets), and to create an ephemeral secure session between them. The SIGMA protocol requires the PSE to be provisioned a certificate of a type called Elliptic Curve Digital Signature Algorithm (ECDSA), and the CSME to be provisioned a certificate of a scheme called Intel

Enhanced Privacy ID (EPID). Further discussions on EPID, SIGMA, and PSE – CSME pairing and ephemeral session establishment are covered later in the architecture deep dive section (Section 4).

To access trusted platform services, an Application Enclave (AppEnclave) establishes a secure session with the PSE using an SGX report based key exchange protocol. More details about AppEnclave – PSE secure session establishment is presented in the architecture deep dive section (Section 4). Typically, an AppEnclave relies on a remote server to provision secrets to it, and needs to attest to the server its security properties. To allow attestation by the remote server of the trust-worthiness of platform services that the AppEnclave uses, the PSE provides its security information as well as that of the paired CSME to the AppEnclave, to be forwarded to the remote server for verification.

2.3 Software / Firmware Stack

The security-oriented architecture diagram in Figure 1 only shows the TCB components involved and their relationship. In implementation, each TCB requires support of untrusted software to provide non-security related functionalities and to expose interfaces for interaction with other components. Figure 2 shows an implementation-oriented software / firmware stack diagram.

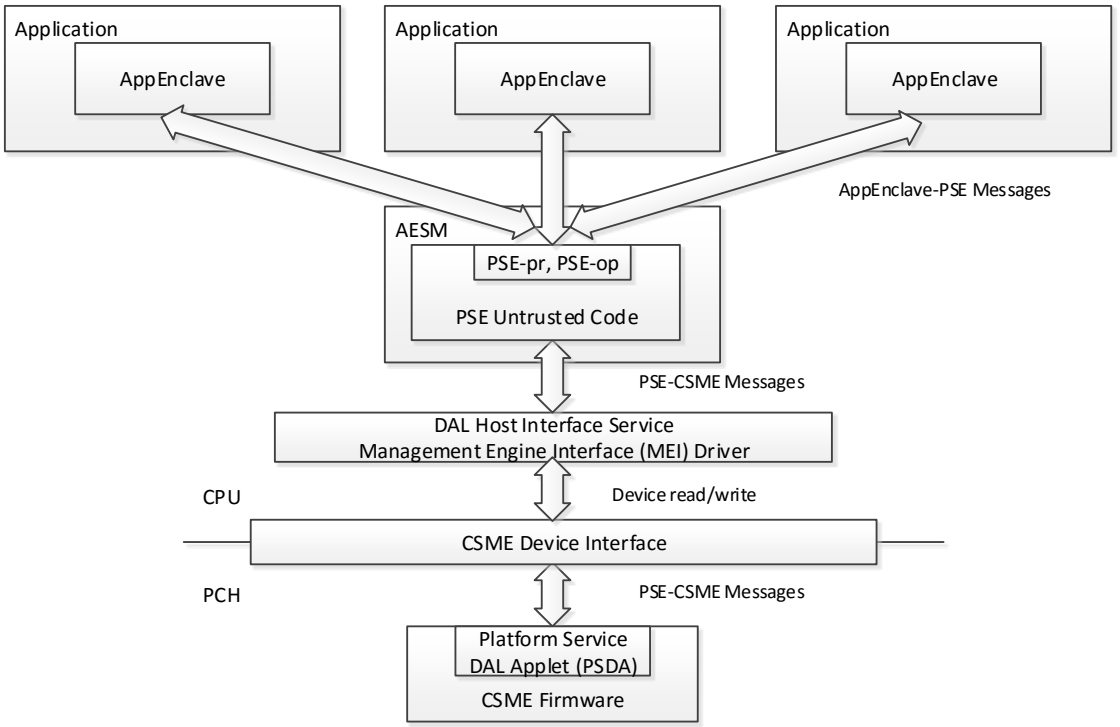

Figure 2 Platform Services Software / Firmware Stack

The CSME consists of the embedded CSME hardware engine in the PCH and the firmware running in it. To support flexibility for adding new features without having to modify the firmware image itself, the CSME firmware has a module called the Dynamic Application Loader (DAL). This DAL module allows the host software to dynamically load and execute a Java applet (called DAL applet) in the CSME at runtime, and to establish a secure session with it. To support flexibility without compromising security, all DAL applets must be signed properly [5].

For communication between DAL applets and host software, two components in the host software stack are involved: a kernel-space Management Engine Interface (MEI) driver, and a user-space DAL host interface service. These components are typically pre-installed along with other Intel software on an Intel platform [5].

The Intel SGX Platform Services related features in the CSME are provided by a DAL applet called the Platform Services DAL Applet (PSDA). The PSDA securely exposes the CSME security capabilities including battery backed RTC and replay-protected storage to the PSE.

A background service process called the Intel SGX Architectural Enclave Service Manager (AESM) hosts the PSE enclave. During runtime, when the AESM starts, it automatically loads and starts the PSDA applet. It also reloads the PSDA upon platform events such as resume from standby or hibernation.

For modular software development, optimized use of limited Intel SGX resources, and improved security by reducing complexity of individual enclaves, the PSE is implemented as two separate enclaves: an enclave for provisioning of a PSE certificate and pairing with the CSME (called PSE-pr), and another enclave for platform services operations (called PSE-op). These two enclaves are designed so that sealed secrets can be shared between them, and there is no need to run both enclaves at the same time. The AESM invokes the PSE-pr for PSE certificate provisioning and PSE – CSME pairing only if a pairing between the PSE and the CSME does not exist yet.

An application enclave (AppEnclave) is hosted in a user-space application. The enclave and the hosting application can use the libraries provided in the Intel SGX SDK to establish secure session with the PSE through inter-process communication, to retrieve Intel SGX Platform Services security information, and to access the supported services.

3 Programming with Platform Services

3.1 Secure Session Establishment between Application Enclave and the PSE

Before an application enclave can access any trusted platform services, it needs to establish a secure session with the PSE. For the session establishment process to be successful, the PSE needs to be provisioned with a valid certificate, successfully paired with the CSME and established an ephemeral secure session with the latter. The Intel SGX Platform Services software is designed so that the application enclave – PSE secure session establishment flow automatically triggers these pre-requisite flows if they are not already completed.

The Intel SGX SDK provides several API functions in its trusted libraries for application enclaves to create and close secure sessions with the PSE, and retrieve security information for remote attestation. The functions in the trusted libraries of the SDK are intended to be invoked from within an enclave. This section provides a brief summary of these functions. More detailed descriptions can be found in [2] and [3]. The functions are:

- `sgx_create_pse_session()` – creates a secure session with the PSE.
  - Upon receiving a request from an application enclave, the PSE checks if the pre-requisite flows are completed. The AESM initiates these flows automatically if needed.

- `sgx_close_pse_session()` – closes the existing secure session with the PSE.
- `sgx_get_pse_sec_prop()` – retrieves security information for the PSE and the paired CSME (including the PSDA), for remote attestation.
  - The retrieved security information is returned as an opaque byte-stream, to be interpreted by the remote attestation server

3.2 Trusted Time Service

For trusted time service, the PSE provides a CSME Protected Real-Time Clock (PRTC) based timer to application enclaves. This trusted time service can be used by an application enclave running on an offline platform (that does not have access to trusted calendar time from a remote time server) to track the amount of time passed since a previous read of the timer. Note that the trusted time service provides coarse-grain timer values in units of seconds relative to a reference point. It does not provide trusted calendar (or wall clock) time. An application enclave that requires trusted wall-clock time needs to retrieve it from a trusted remote time server, which is outside of the scope of the Intel SGX SDK.

The trusted time service uses an epoch value (called timer source epoch) to enable an application enclave to detect if there is discontinuity between different timer reads. A change of the timer source epoch value between two reads indicates that either the PRTC in the paired CSME has been reset due to events like battery replacement, or the PSE has paired with a different CSME due to unexpected event such as software attack. In this case, the application should not trust the calculated duration between the two reads, and handle the error condition properly. More discussion on the architecture for the trusted time service is presented in the architecture deep dive section (Section 4.6).

The Intel SGX SDK provides a trusted library API function, `sgx_get_trusted_time()`, for application enclaves to retrieve the current timer value and the timer source epoch.

3.3 Monotonic Counter Service

The PSE is capable to simultaneously support multiple monotonic counters for multiple application enclaves. Each monotonic counter is assigned a cryptographically unique identifier (called monotonic counter ID), and includes an access control policy based on the signing key and / or measurement of the requesting application enclave.

Internally the PSE maintains a secure database to manage all the active monotonic counters, and uses the replay-protected storage in the CSME for integrity and replay protection of the database. In case the PSE detects unexpected events – such as loss of pairing with the CSME or corruption of the database – that void the integrity and replay protection of the database, it will re-initialize the database, causing all existing monotonic counters to be lost. More discussion on the architecture for the monotonic counter service is presented in the architecture deep dive section (Section 4.7).

An application enclave can request the PSE to create a new monotonic counter, increment an existing one, read its value, or delete it. The application enclave should be designed to detect unexpected loss of a monotonic counter and handle the error condition properly.

The Intel SGX SDK provides a set of trusted library API functions for application enclaves to access the monotonic counter service provided by the PSE, as follows:

- `sgx_create_monotonic_counter()`, `sgx_create_monotonic_counter_ex()` – creates a new monotonic counter with a default initial counter value and returns its ID. The extended version function allows specification of a custom access control policy.
- `sgx_increment_monotonic_counter()` – increments the value of the monotonic counter with the provided ID.
- `sgx_read_monotonic_counter()` – reads the current value of the monotonic counter with the provided ID.
- `sgx_destroy_monotonic_counter()` – deletes the monotonic counter with the provided ID.

3.4 Sample Code - SealedData

To demonstrate how the Intel SGX Platform Services can be used, the Intel SGX SDK includes a sample project called SealedData. This sample project shows how an application can use the monotonic counter and trusted time to enforce policies based on number of uses (called replay protected policy) or expiration time (called time-based policy). These types of policies are enforced in applications such as digital rights management (DRM).

The sample project includes an enclave and a hosting application. For the replay protected policy, the sample project shows how to initialize, update, and verify a policy, as well as how to detect when the number of uses has reached the pre-defined limit, and how to detect when integrity of the policy is compromised. For the time-based policy, the sample project shows how to initialize and use the policy, and how to detect timer expiration [2] [4].

4 Architecture Deep Dive

As discussed in the architecture overview section (Section 2.2), the PSE and the CSME use a SIGMA protocol to pair with each other, then use their shared pairing secret to establish an ephemeral secure session. The SIGMA protocol requires the PSE to be provisioned an ECDSA certificate, and the CSME to be provisioned with an EPID certificate. In addition, the application enclave uses the SGX report based protocol to establish secure session with the PSE. This section provides an in-depth discussion of these architecture topics.

4.1 SGX and CSME Identities for Platform Services

Before pairing, the PSE and the CSME are required be provisioned with valid certificates for mutual authentication. Provisioning of the PSE certificate in turn requires that the Intel SGX platform is provisioned with an EPID attestation key.

The EPID algorithm is an asymmetric key signature scheme that allows a platform to attest to a remote server before receiving services from the latter, while preserving its privacy. Many platforms become members of an EPID group. Each member has a unique private key, while the EPID group has a group ID and a single public key. The remote server has the group ID and public key. An entity called the EPID authority is responsible for management of EPID groups, public and private key provisioning, and platform revocation. A platform uses its own private key to sign messages for the remote server. While

the remote server uses the group public key to verify signed messages, the EPID is designed so that it is not able to determine which specific private key is used to create the signature [5].

The provisioning of an EPID attestation key to an Intel SGX platform is performed by an enclave in the Intel SGX SDK called the Provisioning Enclave (PvE) working with an Intel provisioning server. The PvE demonstrates to the provisioning server that it runs on an authentic Intel SGX CPU and has an up to date TCB. Then it joins an EPID group selected by the server, and is provisioned its EPID private key. After EPID provisioning, the PvE shares its EPID credential (including the private key, the group ID, and the group public key) with another enclave called the Quoting Enclave (QE). An application enclave performs remote attestation to a remote application server using the QE's EPID signing capability. The QE uses the SGX report based protocol to verify that the application enclave runs on the same local SGX platform as the QE itself. Then the QE signs the security properties of the application enclave – contained in the application enclave's SGX report – with its EPID private key into a data structure called SGX quote. The remote application server works with an Intel attestation server to verify the validity of the SGX quote, then verifies the application enclave security information in the signed SGX report. More information about Intel SGX EPID provisioning and remote attestation can be found in white paper [9].

The CSME is provisioned with its own EPID private key during Intel's manufacturing process, as well as the rest of its EPID credential (including the EPID certificate with the group public key) through other means, before PSE – CSME pairing is invoked the first time [5].

4.2 PSE Certificate Provisioning and PSE – CSME Pairing

The PSE and the CSME use a SIGMA protocol to pair with each other. SIGMA is a key exchange protocol between two end points, with one called the SIGMA prover and the other the SIGMA verifier. This SIGMA protocol requires the prover to be in an EPID group and provisioned with an EPID private key and its EPID certificate, and requires the verifier to be provisioned with an ECDSA certificate. The prover and verifier verify each other's' certificates and signatures and use Elliptic Curve Diffie-Hellman (ECDH) algorithm to establish shared session keys between them for secure message exchange [5] [8].

In the SIGMA protocol for PSE – CSME pairing, the CSME functions as the SIGMA prover, and the PSE functions as the SIGMA verifier.

Before pairing with the CSME, the PSE needs to be provisioned with its ECDSA certificate by an Intel server. In the provisioning flow, the PSE performs SGX remote attestation with the Intel server to prove that it is a PSE instance with up to date TCB running on an authentic Intel SGX platform. Upon successful attestation, the Intel server signs the ECDSA public key from the PSE into a certificate. Figure 3 depicts a high-level view of the PSE certificate provisioning flow.

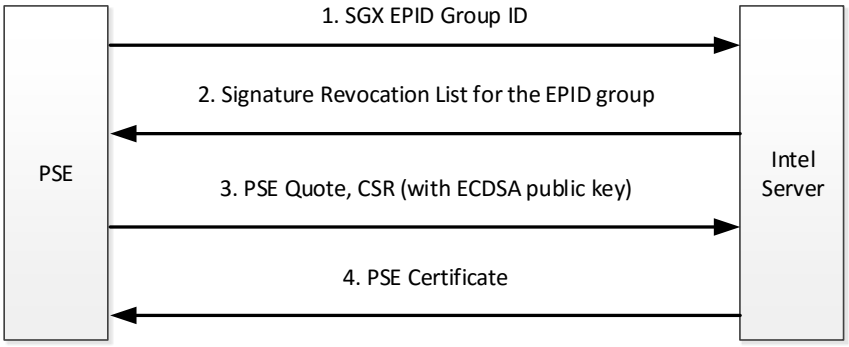

Figure 3 PSE Certificate Provisioning Flow

a. In steps 1 and 2, the PSE sends the SGX EPID group ID to the Intel server, and receives the signature revocation list for the SGX EPID group.
b. In step 3 and 4, the PSE generates an ECDSA private / public key pair, puts the public key in a data structure called Certificate Signing Request (CSR). It then works with the quoting enclave (QE, described in Section 4.1) to sign its SGX report that contains the PSE security properties and a hash of the CSR into an SGX quote. The QE also checks the revocation list the PSE received in step 2 to verify that its EPID private key is not revoked. By inspecting the received SGX quote and the signed SGX report and CSR, the Intel server verifies the PSE security properties and the CSR integrity, and signs the ECDSA public key in the CSR into a certificate.

Upon successful completion of the PSE certificate provisioning flow, the PSE has its ECDSA private key and a certificate for its ECDSA public key.

With the PSE provisioned its ECDSA certificate and the CSME having its EPID credential, they can pair with each other using the SIGMA protocol, to verify mutual trust and to establish shared pairing secrets. Figure 4 shows a high-level view of the SIGMA based PSE – CSME pairing flow.

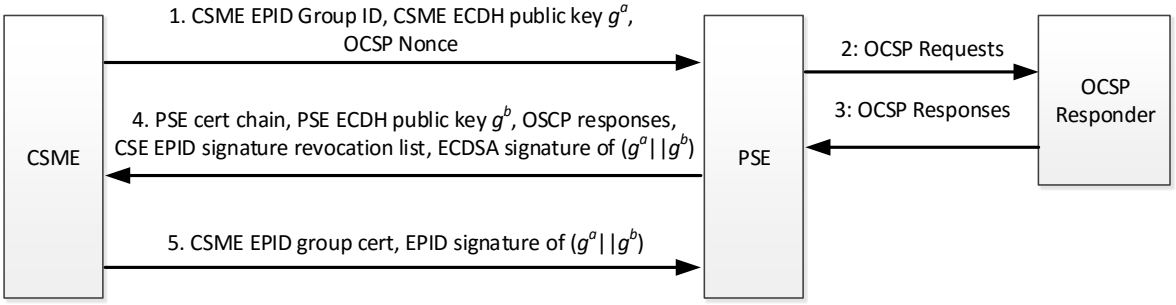

Figure 4 PSE – CSME SIGMA-based Pairing Flow

a. In step 1, the CSME generates an ECDH private key a and calculates the public key g^a. It also generates a nonce for use in OCSP requests (called OCSP nonce). The CSME sends its EPID group ID, its ECDH public key g^a, and the OCSP nonce to the PSE.

b. In steps 2 and 3, the PSE retrieves an OCSP response for every cert in its ECDSA cert chain, using the CSME OCSP nonce in its OCSP requests. It verifies the OCSP responses and triggers re-provisioning of its ECDSA cert if its existing cert is revoked.
c. In step 4, the PSE finds the signature revocation list for the CSME EPID group, generates its ECDSA key pair (b, g^b), derives shared secret g^{ab} from its private key b and the CSME public key g^a, and signs the two ECDH public keys ($g^a||g^b$) with its ECDSA private key. From the shared secret g^{ab}, the PSE uses a Keyed-Hash Message Authentication Code (HMAC) algorithm to derive shared keys (called SMK, SK, and MK). Finally the PSE sends its ECDSA cert chain, OCSP responses, its ECDH public key g^b, the CSME EPID signature revocation list, and the ECDSA signature of ($g^a||g^b$) to the CSME.
d. In step 5, the CSME verifies the PSE cert chain and the OCSP responses, verifies the ECDSA signature of ($g^a||g^b$) using the public key in the PSE cert, signs ($g^a||g^b$) using its EPID private key and the received EPID signature revocation list, and derives the same shared secret SMK, SK, and MK from shared secret g^{ab}. Finally the CSME sends its EPID group cert and the EPID signature of ($g^a||g^b$) to the PSE.
e. The PSE, upon receiving the message from the CSME in step 5, verifies the received EPID group cert and the EPID signature of ($g^a||g^b$).

Upon successful completion of the PSE – CSME pairing flow, the PSE and the CSME have verified each other's certificates, and are in possession of the same shared secrets SK and MK. They use the SK and MK to derive additional values as their identities (called PSE ID and CSME ID). In addition, the PSE generates a nonce, called Pairing-Nonce, to uniquely identify this pairing. The PSE also collects security information of the paired CSME (see Section 4.5 for details). The PSE and the CSME only need to perform pairing for one time. They securely save their pairing secrets in their persistent storage.

4.3 PSE – CSME Ephemeral Session Establishment

With the PSE – CSME pairing in place, upon startup, the PSE initiates a flow to establish an ephemeral secure session with the CSME using their shared pairing secrets, as shown in Figure 5.

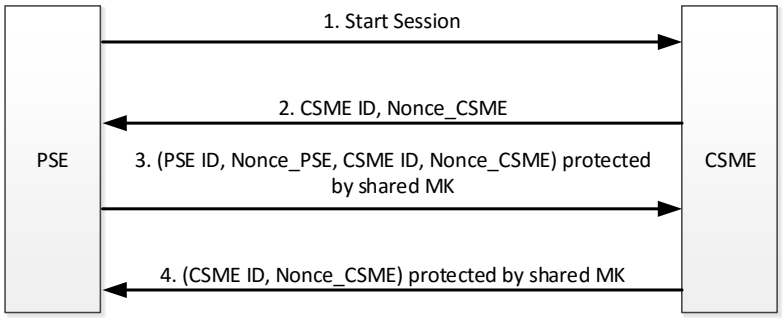

Figure 5 PSE and CSME Ephemeral Session Establishment Flow

a. The PSE and the CSME generate their nonce, Nonce_PSE and Nonce_CSME respectively.
b. In step 2 and 3, the PSE and the CSME exchange their IDs and nonce.

c. In steps 3 and 4, messages between the PSE and the CSME are integrity-protected by shared pairing key MK, so the PSE and the CSME can verify the integrity of their exchanged messages and m the IDs in the messages against their locally-saved pairing secrets

Upon successful completion of the PSE – CSME ephemeral session establishment flow, the PSE and the CSME use an HMAC algorithm with their shared pairing key SK and nonce Nonce_PSE and Nonce_CSME to derive session specific symmetric keys. The established session only lasts until the PSE stops and is thus called an ephemeral session.

4.4 Application Enclave – PSE Secure Session Establishment

Before an application enclave (AppEnclave) can use the trusted platform services, it establishes a secure session with the PSE using an SGX report based key exchange protocol. A high-level view of the secure session establishment flow is shown in Figure 6.

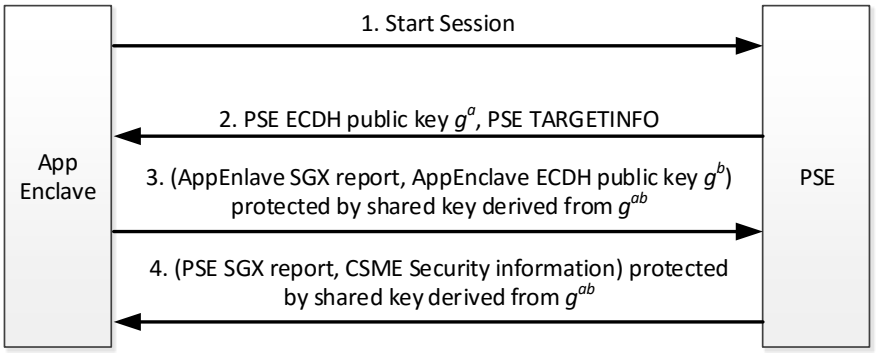

Figure 6 Application Enclave - PSE Secure Session Establishment Flow

a. The AppEnclave and the PSE generate their ECDH private keys *a* and *b*, and calculate their corresponding public keys g^a and g^b respectively.
b. In steps 2 and 3, they exchange their public keys and calculate their shared ECDH secret g^{ab} for integrity protection of their message exchange in steps 3 and 4.
c. In steps 3 and 4, the AppEnclave and the PSE send their SGX reports to each other for verification that both enclaves run on the same SGX platform. The PSE also passes the security information of the paired CSME to the AppEnclave. The AppEnclave retrieves the security information of the PSE from its SGX report.

Upon successful completion of the AppEnclave – PSE secure session establishment flow, the AppEnclave and the PSE use a Cipher-based Message Authentication Code (CMAC) algorithm with their shared ECDH secret g^{ab} to derive session keys for protection of follow-up messages between them. The AppEnclave also has the security information of the PSE and the paired CSME.

4.5 PSE and CSME Security Information for Remote Attestation

Before an application enclave (AppEnclave) can receive secrets such as private keys from a remote server, it needs to attest its security properties to the latter. If the enclave uses trusted platform services, the Intel SGX Platform Services subsystem becomes part of its TCB. As a result, the enclave

needs to provide the security information of the PSE and its paired CSME to the remote server for attestation.

The PSE security information is received in the PSE SGX report as part of the AppEnclave – PSE secure session establishment flow. The information includes PSE product ID, security version number (SVN), the PSE signing key information, and the PSE enclave measurement.

The CSME security information is received by the PSE as part of the PSE – CSME pairing flow, and provided by the PSE to the AppEnclave as part of the AppEnclave – PSE secure session establishment flow. The CSME security information includes the CSME's EPID group ID, the version numbers of the revocation lists for the CSME EPID private key and signature, and the identity information and security version number of the PSDA.

4.6 Trusted Time Service Architecture

As discussed in Section 3.2, for trusted time service, the PSE uses the CSME Protected Real-Time Clock (PRTC) based timer, and provides a timer source epoch to allow application enclaves to detect timer discontinuity. A high-level view of the architecture for the trusted time service is shown in Figure 7.

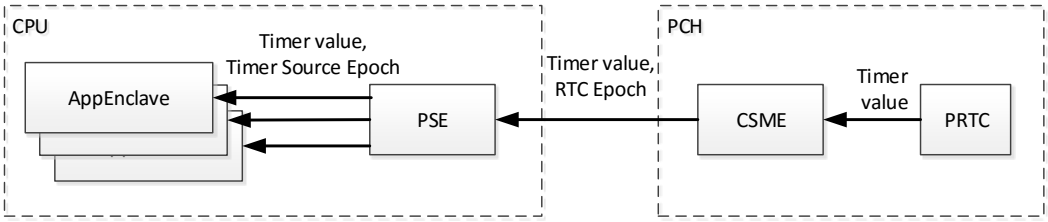

Figure 7 High-level Architecture for Trusted Time Service

In the CSME, each time the PRTC is reset (for example, due to battery replacement), it generates a new nonce for its RTC epoch. Each time the PSE reads the RTC timer value from the CSME, the CSME returns the current RTC epoch along with the timer value.

For each AppEnclave, the PSE derives per-enclave timer source epoch as a hash of the RTC epoch, the PSE – CSME Pairing-Nonce, and the singing key information of the AppEnclave. As a result, different enclaves have different timer epoch values. Within an enclave, the timer source epoch value changes if the PRTC is reset or the PSE – CSME pairing changes.

4.7 Monotonic Counter Service Architecture

4.7.1 Overall Architecture

As discussed in Section 3.3, the PSE maintains a secure database to manage all the monotonic counters, and uses the replay-protected storage (RPS) in the CSME for integrity and replay protection of the database. Since the monotonic counters are implemented in software by the PSE, they are also called Virtual Monotonic Counters (VMC). The database is also called the VMC database (VMC DB). Figure 8 shows a high-level architecture.

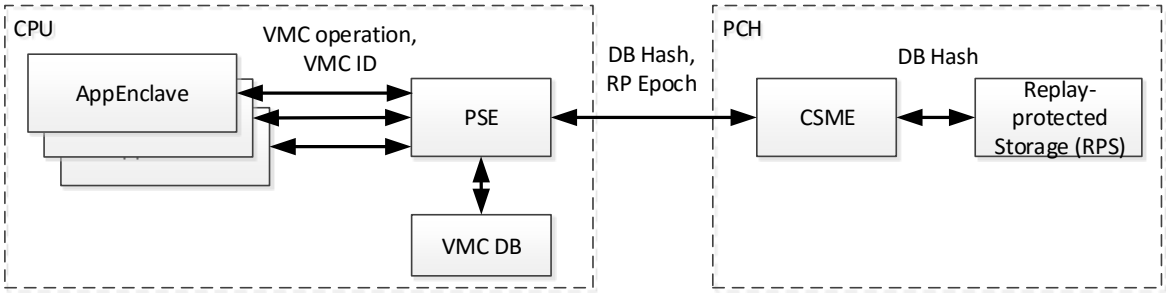

Figure 8 High-level Architecture for Monotonic Counter Service

Each VMC instance in the VMC DB is assigned a cryptographically unique VMC ID, and includes an access control policy based on the signing key and / or measurement of the requesting AppEnclave.

To ensure integrity- and replay-protection, the PSE securely calculates and tracks a hash (called DB hash) over all the database entries. Upon each operation requested by an AppEnclave that results in creation, deletion, or update of a VMC entry in the VMC DB, the PSE generates a new DB hash, and sends it to the CSME for storage.

The CSME provides a replay-protected storage (RPS) for the PSE to securely store its DB hash. It generates a random number as an epoch for replay-protection (RP epoch). Anytime when the CSME detects a condition that invalidates replay-protection of the stored DB hash, it generates a new RP epoch value.

The PSE uses the RP epoch and the PSE – CSME Paining-Nonce in its calculation of the VMC DB hash. This allows it to detect loss of reply-protection of the database when RP epoch or Pairing-Nonce changes, and to invalidate the existing VMC DB, resulting in deleting all existing VMC instances.

4.7.2 Monotonic Counter Database Protection

In the PSE, for integrity and replay protection of the VMC database, the VMC entries are organized into a binary tree structure, with all the VMC entries as leaf nodes, one or more layers of intermediate nodes, and a single root node. Each intermediate node holds the hash of its left and right children. The root node contains the hash of these fields: left and right children, Pairing-Nonce of the PSE – CSME pairing, and RP epoch from the CSME. Figure 9 shows a high-level view of the binary tree structure.

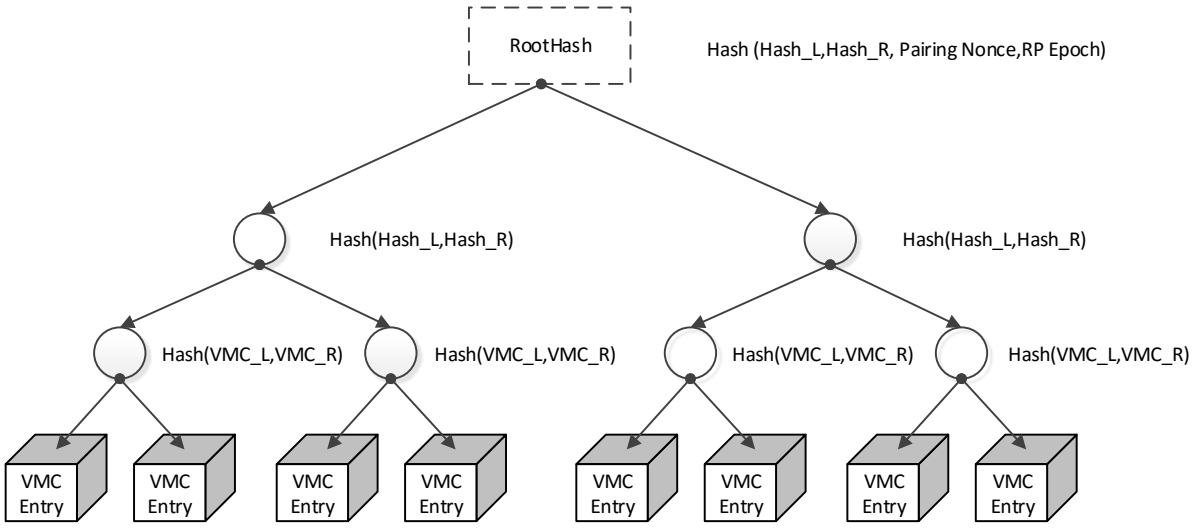

Figure 9 VMC Database Binary Tree Structure for Integrity and Replay Protection

To verify the integrity of a VMC entry, the PSE reads into its secure memory all the nodes along the branch from the VMC entry leaf node up to the root node, as well as the other immediate children of the intermediate and root nodes. The PSE calculates the hash of all the ancestors of the target VMC entry, along the path to and including that of the root node. It then retrieves from CSME the saved DB hash, and compares the retrieved and the calculated root node hash. Verification is successful if the two hash values are the same.

After integrity verification of a VMC entry, the PSE can update it. After an update operation, the PSE re-calculates the hash of all the ancestor nodes of the VMC entry, writes all updated VMC entry and nodes back to the VMC database, and sends the new root node hash value to the CSME for storage.

In the PSE implementation, the VMC database is initialized with a fixed number of unused VMC entries, so the binary tree structure of the VMC entries does not need to change. When a new VMC instance is allocated, the PSE finds an unused VMC entry in the database for it. When a VMC instance is removed, the corresponding VMC entry in the database is simply marked as unused.

5 Summary

The purpose of this paper is to help software developers in their programming with the Intel SGX Platform Services, and to help readers in their understanding the internal design principles. This paper presented an overview of the trusted platform services and the architecture and the software stack. It then explained the APIs provided by the Intel SGX SDK. For the readers that are interested in learning about the design principles, the paper provided an in-depth discussion of the flows for the components to establish their identities and to create secure sessions between them, as well as the design of the supported platform services. More information about the Intel SGX SDK and its source code can be found in [1] [2] [3] and [4].

6 References

[1] Intel SGX SDK Developer Zone, sources: https://software.intel.com/en-us/sgx-sdk and https://01.org/intel-software-guard-extensions

[2] Intel SGX SDK Developer Guide, source: https://01.org/intel-software-guard-extensions/documentation/intel-sgx-developer-guide

[3] Intel SGX SDK Developer Reference, source: https://01.org/intel-software-guard-extensions/documentation/intel-sgx-sdk-developer-reference

[4] Intel SGX Software for Linux* OS Source Code Repositories, source: https://github.com/01org/linux-sgx

[5] Xiaoyu Ruan, Platform Embedded Security Technology Revealed: Safeguarding the Future of Computing with Intel Embedded Security and Management Engine, ISBN-13: 978-1430265719

[6] A Primer on Intel Software Guard Extensions (Intel SGX), source: https://software.intel.com/sites/default/files/managed/3e/b9/SF15_ISGC003_81_SGX_DL_100_small.pdf

[7] Intel 64 and IA-32 Architectures Software Developers Reference Manual, November 2015, source: http://www.intel.com/content/www/us/en/processors/architectures-software-developermanuals.html

[8] Key Exchange with Anonymous Authentication using DAA-SIGMA Protocol, source: https://eprint.iacr.org/2010/454.pdf

[9] Simon Johnson et al, Intel Software Guard Extensions: EPID Provisioning and Attestation Services, source: https://software.intel.com/en-us/blogs/2016/03/09/intel-sgx-epid-provisioning-and-attestation-services

[10] Intel Software Guard Extensions (Intel SGX) Tutorial, June 2015, source: https://software.intel.com/sites/default/files/332680-002.pdf

[11] Intel Software Guard Extensions Tutorial Series, source: https://software.intel.com/en-us/articles/introducing-the-intel-software-guard-extensions-tutorial-series