TRUSTED CODE EXECUTION ON UNTRUSTED PLATFORMS USING INTEL SGX

Guevara Noubir & Amirali Sanatinia
Northeastern University, USA

Email {noubir, amirali}@ccs.neu.edu

ABSTRACT

Today, isolated trusted computation and code execution is of paramount importance to protect sensitive information and workflows from other malicious privileged or unprivileged software. *Intel Software Guard Extensions* (*SGX*) is a set of security architecture extensions first introduced in the *Skylake* microarchitecture that enables a Trusted Execution Environment (TEE). It provides an 'inverse sandbox', for sensitive programs, and guarantees the integrity and confidentiality of secure computations, even from the most privileged malicious software (e.g. OS, hypervisor).

SGX-capable CPUs only became available in production systems in Q3 2015, and they are not yet fully supported and adopted in systems. Besides the capability in the CPU, the BIOS also needs to provide support for the enclaves, and not many vendors have released the required updates for the system support. This has led to many wrong assumptions being made about the capabilities, features, and ultimately dangers of secure enclaves. By having access to resources and publications such as white papers, patents and the actual *SGX*-capable hardware and software development environment, we are in a privileged position to be able to investigate and demystify *SGX*.

In this paper, we first review the previous trusted execution technologies, such as *ARM Trust Zone* and *Intel TXT*, to better understand and appreciate the new innovations of *SGX*. Then, we look at the details of *SGX* technology, cryptographic primitives and the underlying concepts that power it, namely the sealing, attestation, and the Memory Encryption Engine (MEE). We also consider use cases such as trusted and secure code execution on an untrusted cloud platform, and digital rights management (DRM). This is followed by an overview of the software development environment and the available libraries.

1. INTRODUCTION

Today, cloud platforms are becoming more widely used, both by end-users and enterprises. However, the notion of trusting a third party with your secrets is not very desirable for many entities. The status quo not only forces users to put their faith in the honesty and trustworthiness of the cloud providers but also forces them to trust in the lack of malware and comprise of the cloud platforms. *Intel SGX* is a new technology that guarantees the confidentiality of users' data on a remote node, against other unprivileged or even privileged software such as the operating system and hypervisor. Without adequate support from the hardware to provide a secure execution environment, previous work relied on trusted hypervisors to protect applications against malicious OSs [1–3]. An alternative approach that mobilized the research community is to compute over encrypted data [4], for example using Fully Homomorphic Encryption (FHE) schemes [5] that can perform general operations on encrypted data. However, current FHE techniques are still several orders of magnitude slower than necessary for practical applications.

Earlier attempts such as *Intel TXT*, formerly known as *LaGrande Technology*, did not succeed in becoming widely adopted and deployed. *Intel TXT* is a platform-level enhancement and set of extensions to attest the authenticity of the hardware and operating system by enabling the measurement and verification of the environment [6]. Currently, *ARM TrustZone* is one of the most successful and widely deployed TEEs both for clients and enterprises.

Previous works have looked at *Intel SGX* and discussed its potentials and shortcomings [7–9]. However, they were based on the information available prior to the official release of the *SGX* hardware (processors and supporting motherboards) and its specifications. In this work, by having access to resources and publications such as white papers, patents and the actual *SGX*-capable hardware and software development environment, we are in a privileged position to be able to report on our experience with *SGX*. We first look at *ARM TrustZone*, the other competing TEE technology that is widely used. Then we overview the *SGX* internals and the underlying concepts that power it, followed by a discussion of its use cases. Finally, we review the software development model and libraries available in *SGX*.

2. ARM TRUSTZONE

ARM TrustZone is a set of security enhancement extensions to the *ARM* architecture that appears in *ARMv6* and later versions. It introduces two security modes, which divide the CPU into two isolated worlds, the secure mode and the normal mode. A third mode, called the monitor mode, is in charge of the switch between the secure and normal worlds. The Secure Monitor Call (SMC) instruction is invoked to switch between the two worlds. In *TrustZone*, the two worlds have their own separate address spaces and different privileges. The memory is partitioned into two sections, one of which is reserved exclusively for the secure mode. Furthermore, individual peripherals can be assigned to different worlds. Both worlds can run any software, ranging from unprivileged user-level applications, to the OS.

To guarantee the integrity of the secure world's components and software, upon powering the device, it boots into the secure world, and after executing the secure boot and verifying the signature of the boot image, it can attest that the software has not been modified.

To determine the state of the CPU, an extra bit is added to the Secure Configuration Register (SCR), called the non-secure (NS) bit, which indicates the security context of the CPU. When the NS bit is zero, the CPU is in the secure world mode, and when the NS bit is set to 1, the CPU is in the normal mode.

Previous work has looked at the use and application of *TrustZone* for a wide range of domains. For example, to regulate devices in restricted spaces [10], for cache-assisted secure execution [11],

and for enabling a *Samsung* mobile security solution called *KNOX*.

As noted in *ARM TrustZone*, the TCB is much larger than *SGX*. The larger size of TCB can lead to errors and ultimately vulnerabilities. Furthermore, a trusted system stack, including OS, firmware, and libraries, needs to be implemented and trusted by all the users.

3. INTEL SOFTWARE GUARD EXTENSIONS (SGX)

Intel SGX allows the creation of secure enclaves that can keep and be trusted with a secret. In the context of *SGX*, enclaves are isolated execution units, with encrypted code and data. At the beginning, enclaves have no secret, since they can be disassembled and viewed like any other normal program. After their launch, the enclaves need to be provisioned, to retrieve the secret data. The following is an overview of *SGX* [12] (Figure 1 provides a diagram of the procedure and lifecycle of an *SGX* enclave):

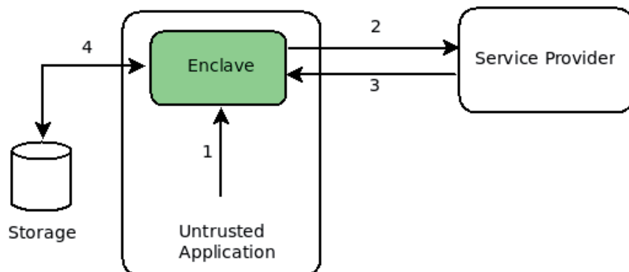

Figure 1: Diagram of the SGX enclave lifecycle.

- **Step 1 (Launch)**: the untrusted application loads the enclave code and instantiates it. During this process a log is created called the enclave's measurement. This measurement is used in the verification by the remote party (service provider).
- **Step 2 (Attestation)**: the enclave contacts the service provider for provisioning and retrieving the secrets. As mentioned earlier, the enclave does not contain any secret information before provisioning. The enclave presents its attested measurement to the service provider, which identifies the hardware environment and the enclave.
- **Step 3 (Provisioning)**: after verifying the attestation provided by the enclave in step 2, the service provider establishes a secure communication channel with the enclave. The service provider uses the key exchange information in the attestation. After establishing the secure channel, the service provider sends the secure data to the enclave.
- **Step 4 (Sealing/Unsealing)**: to allow an enclave to access the secret material in a secure and confidential way, the data can be sealed (encrypted) and stored on persistent storage. Later, based on the policies defined by the service provider, the data can be decrypted into an enclave without going through the remote attestation and provisioning again.

3.1 New instructions and data structures

The two main challenges to enable the functionalities of *SGX* are memory access semantics and protection of the address mappings [13]. To address this, new instructions, data structures, and a new mode of execution have been introduced.

The 18 new instructions can be categorized into five groups: five instructions to build and destroy enclaves, four instructions to enter and exit enclaves, five instructions to move enclave pages to and from memory, two instructions to debug enclaves, and two instructions for the security operation of enclaves, including key generation and the measurement of the enclaves.

Six new data structures have been introduced to hold the enclave's data and metadata:

- The Enclave Page Cache (EPC) is a protected memory region used to hold the protected code and data, in 4k pages. The EPC is encrypted through the Memory Encryption Engine (MME), and is managed by the OS/VMM.
- The Enclave Page Cache Map (EPCM) contains the metadata of the enclave pages, and is used by the CPU to keep track of the content of EPC pages. The EPCM is controlled by the CPU and is not directly accessible by the software or devices.
- The SGX Enclave Control Store (SECS) and Thread Control Structure (TCS) hold the metadata for each enclave, and each thread, respectively.
- The Version Array (VA) of evicted pages.
- The SIGSTRUCT record, which is responsible for the signature and sealing identity of the enclave.

The new mode (enclave mode) is activated when a process moves into an enclave. In this mode, extra memory access checks are performed to ensure the confidentiality and protection of the enclave's memory from other processes.

3.2 Types of enclaves

Enclaves are the secure computation units that run in ring level 3 (user level). They have no privileged access, yet they are protected against the higher level, privileged programs, including the OS, VMM and hypervisor. Since enclaves run in ring 3, and do not have direct access to peripheral and I/O devices, they cannot harm systems [7]. The enclaves are designed to work on multi-core platforms, since multiple enclaves can run at the same time. Furthermore, the enclave and the untrusted application can run in simultaneous threads. *SGX* provides isolation between enclaves, and mitigates against replay attack, by checking for the freshness and integrity of the pages, through the MEE. To ensure the security of enclaves, access control mechanisms make sure that the enclave data is protected from other software while it is in the register and the cache inside the CPU. Not even exits from enclaves or exception handling leak information about them.

The secrets will be provisioned into the enclaves after the remote attestation is complete. Special 'architectural enclaves' are involved in this process to generate a measurement report and attestation of the enclave. After this set-up, the service provider can provision their secret into the enclave. To avoid

going through the remote attestation each time and provisioning the enclave, *SGX* provides a sealing mechanism. Sealing binds the data and key to the enclave and the CPU. In the future the enclave will be able to access the protected content without going through the provisioning and remote attestation process. The attestation process uses the Enhanced Privacy ID (EPID) and group signature algorithm to preserve the privacy of the individuals, since each private key belongs to a much larger set of private keys that correspond to a public key, therefore it will not be possible to identify or track an individual in the set.

To create an enclave, first the ECREATE instruction creates and initializes the SECS structure. EADD adds the pages to the enclave; after the page is added, EEXTEND measures the content, and EINIT finalizes the creation of the enclave.

3.2.1 Architectural enclaves

There are two types of enclaves, the 'architectural enclaves', which are provisioned and belong to *Intel*, and the normal/user enclaves, which are created by the user or service provider. The architectural enclaves facilitate the attestation, provisioning and licensing capabilities. Only these enclaves have access to the keys that are inside the CPU.

3.2.1.1 Provisioning enclave

The provisioning enclave uses EGETKEY to access the provisioning key that is provided by *Intel* to the CPU. It is used to authenticate the CPU to the *Intel* provisioning service [6]. The *Intel* provisioning service generates an attestation key and returns it to the provisioning enclave which is encrypted with the provisioning seal key for storage on the platform.

3.2.1.2 Quoting enclave

The quoting enclave creates the EPID key that is used to sign the platform attestations. Only this enclave has access to the EPID key inside the CPU fuse (by calling the EGETKEY instruction). This key also indicates the trustworthiness of the platform. The EPID key is bound to the device's firmware version. The quoting enclave and underlying keys facilitate the remote attestation procedure.

3.2.1.3 Licensing enclave

This is used to produce the code in the deployment mode, otherwise the program is compiled and run in debug mode, which means it does not utilize the full power of the *SGX* capability and the protection it provides. All public keys need to be registered with *Intel* (at least in the current generation of *SGX* – in future generations, this process might be moved to other domains, for example an enterprise could be in charge of its own licensing server). As of now, *Intel* claims, this is a security measure, and it is not intended to marginalize service providers who have not paid enough licence fee to develop and deploy *SGX*-capable software. This is one of the less discussed aspects of *SGX* that has raised some concerns [14].

3.2.2 Normal/user enclaves

The other type of enclaves are the ones that are created by the user or service providers. These enclaves do not have access to the keys inside the fuses. They rely on the architectural enclaves for attestation. Note that the enclave code should not contain any secret, since they can be disassembled just like any other binary. The secret should be provisioned in the enclaves after instantiation and provisioning.

3.3 Attestation

Attestation is used to ensure that the software and enclaves are instantiated on a genuine *Intel SGX* platform [12] [15]. There are two modes of attestation: local attestation and remote attestation. In the former, one enclave wishes to prove and authenticate to another on the same platform that it is also running on the same platform. In the latter, an enclave wishes to prove to a remote third party the authenticity of itself and the platform on which it is being instantiated.

For attestation and sealing, *SGX* has access two measurement registers. MRENCLAVE holds the identity of the code and data. It is a SHA-256 digest of the enclave creation log. It includes code, data, stack, heap, and the position of the pages and the security flags. MRSIGNER, which acts as the identity of the signer authority, is a structure which contains a signed enclave certificate (SIGSTRUCT) and the expected value for the MRENCLAVE. If the checks in the hardware pass, then the public key of the signer is stored in MRSIGNER.

3.3.1 Local attestation

Local attestation is used when a developer wants two enclaves to operate together on the same platform. The two enclaves can authenticate each other, and ensure that they are running on the same platform.

When an enclave invokes the EREPORT instruction, it creates a signed structure called REPORT that contains the identity of the enclave, attributes, and additional information that the developer has specified to be passed on to the target enclave, as well as the Message Authentication Code (MAC). The target enclave would verifies the MAC of the report, to ensure that the enclave that created the report runs on the same platform. The MAC is created using AES128-CMAC, and the key is a shared symmetric key retrieved by calling the EREPORT instruction on the source enclave and EGETKEY on the target enclave.

The REPORT structure also has a 256-bit field for user data, which can be used, for example, to authenticate randomly generated Diffie-Hellman keys that two enclaves now share and can use for further secure communication and data sharing.

3.3.2 Remote attestation

The secrets are provisioned into an enclave after a remote service provider has verified the enclave based on the remote attestation that the quoting enclave generates. Therefore, the remote attestation is of paramount importance. The remote attestation is mostly used at the beginning for the provisioning, and after that the data can be sealed to the platform and stored on the persistent storage. The remote attestation also allows the establishment of a secure communication channel between the service provider and the enclave, by negotiating authenticated Diffie-Hellman keys. This is analogous to the key negotiation procedure in SSL/TLS.

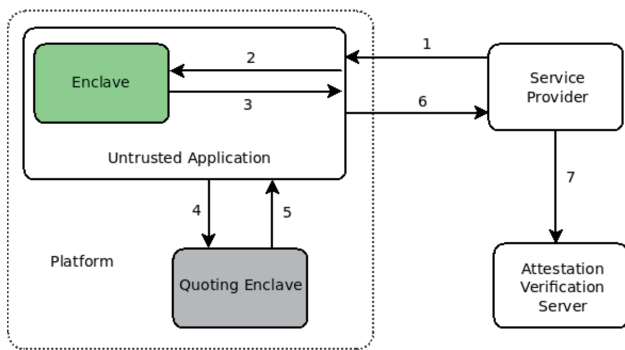

Figure 2: Flow diagram of remote attestation.

Figure 2 depicts the attestation procedure. First, the service provider (challenger) asks the application to provide an attestation (step 1). Then the application asks its enclave to create an attestation (step 2), and the enclave returns the local attestation to the application (step 3). To transform the local attestation to a remote attestation, the application sends it to the quoting enclave (step 4). The quoting enclave replaces the MAC of the REPORT with a signature created with the CPU's private key, using the EPID group signature. The newly created structure is called QUOTE. The QUOTE is returned to the application (step 5). The application returns the remote attestation (QUOTE) to the challenger (step 6). To verify the remote attestation, the service provider contacts the attestation verifier server (step 7).

3.4 Sealing/unsealing

After an enclave is provisioned with a secret, the confidentiality of the secret is guaranteed within the CPU and TCB boundary. However, after the enclave exits, or after a power outage, when the enclave is destroyed, the secret that resided within the protected memory is also removed. To allow access to the secrets in the future, *SGX* provides the sealing functionality, where the data can be encrypted, using the sealing keys provided by EGETKEY, and stored on persistent storage. There are two sealing policies supported by *SGX*: sealing to the enclave and sealing to the author/sealing identity [12].

When sealing to the enclave identity, EGETKEY keys are based on the enclave measurement (MRENCLAVE). Therefore, it provides an isolation for data access between different versions of the same enclave. Additionally, any changes to an enclave that result in a different measurement make the data unusable, since the key also changes. This makes the migration of the data between software upgrades harder.

When choosing to seal the data to the sealing identity, EGETKEY returns keys based on the value of MRSIGNER, and the enclave's version. This approach allows easy migration of data between different versions of an enclave. Furthermore, it allows the transparent sharing of the sealed data between different enclaves created by the same developer/service provider (sealing identity). The sealing authority still has the option to limit the data sharing between enclaves of same security version number (SVN), by specifying this attribute in EGETKEY.

3.5 Memory Encryption Engine (MEE)

In *SGX*, only the CPU and its internals are in the TCB, and the memory is not. Therefore, to protect the contents of EPC while in the RAM, it needs confidentiality, integrity and freshness. Merely encrypting the data is not enough, it also needs to be integrity checked and mitigated against replay attacks [16]. However, in many physical attacks, such as the cold boot attack, merely encrypting the memory contents is sufficient.

The Memory Encryption Engine (MEE) is an extension of the memory controller which provides the aforementioned functionalities. The requests for memory access to the protected memory pass through MEE, which encrypts/decrypts the data before writing/reading it to/from RAM, and verifies the integrity and freshness of the data. The limit for protected memory is 128MB, but only 96MB is usable for the enclaves, because the rest of the space is used to store the integrity tree and the MACs. For encryption, MEE uses the AES block cipher in counter mode (CTR), for increased speed and parallelization. The MACs and the integrity tree tags are based on the Carter-Wegman MAC.

Note that MEE is not an oblivious RAM, and it is not protected against side channel attacks and traffic analysis. MEE has the following three properties [16]:

1. The keys are generated randomly at boot time and never leave the TCB boundary.
2. The authentication and encryption keys are different.
3. The MME enforces the drop-and-lock policy, meaning that if the MAC verification of a page fails, MEE issues a fault and it drops the transaction immediately. This causes the system to stop and require a reboot. At the reboot time, new keys will be generated. Therefore, the adversary has one trial per key.

The MEE introduces an overhead to the operations. According to the measurement in [16], the performance overhead ranges from 2.2% to 14%, with an average of 5.5%. The statistics are based on adapting SPECINT2006 v01 and the Graphene library OS.

3.6 Enhanced Privacy ID (EPID)

Due to privacy concerns with asymmetric signing schemes, *Intel* created EPID, which is an extension of the Direct Anonymous Attestation (DDA). The DDA scheme is a cryptosystem that provides anonymous signatures, specifically designed for the Trusted Platform Module (TPM). EPID improves DDA by adding revocation capabilities, and improved efficiency.

In EPID, one group key corresponds to many private keys. Each one of the private keys can generate a signature that can be verified by the group public key. The issuer does not need to know the members' private keys. Moreover, the signatures are anonymous, meaning the verifier cannot determine who created the signature. Furthermore, an important difference between EPID and other group signatures is that the EPID signatures are untraceable, meaning that not even the issuer can determine the group member who created the signature [17].

EPID is used by the quoting enclave to sign the enclave's remote attestations. In the context of *Intel SGX*, the group refers

to the set of CPUs of the same type. For example, CPUs from the core i3, i5, or i7 families. Therefore, based on this grouping, the size of a fully populated group would be a few million platforms [18].

There are two signature modes, with different linkability capabilities. For each signature a base is chosen; if the base of two signatures is different, then the two signatures are unlinkable. However, if the bases for two signatures are the same, then it is possible to determine if the signatures are generated by the same key. Note that it is still not possible to identify the specific key that generated the signature, but only whether the same key has generated the signatures. The two modes discussed are called Random Base Mode, and Name Base Mode.

From the security point of view, the Name Base Mode signatures are preferred. Imagine a scenario where an EPID key is compromised, and a malware writer is able to trick users into using this enclave. If the Random Base Mode is used instead of the Name Base Mode, the EPID owner will not be able to detect that all the signatures belong to the same key, or even notify users about the key. Therefore, the Name Base Mode is preferred [18].

One of the enhancements to EPID is the revocation capability. There are four supported revocation mechanisms/modes. Private key revocation (if *Intel* receives a private EPID key), verifier local revocation (if a key is noticed to be compromised, the key can be revoked locally, which is possible when the Name Base Mode is used), signature-based revocation (when evidence is provided to a revocation authority that a key is compromised, the corresponding certificate will be added to the Certificate Revocation List (CRL), which is available in both the Random Base, and Named Base modes), and finally, group-based revocation (when a group is no longer valid, e.g. if the group master key is compromised).

4. USE CASES

In this section we look at a few use cases for the *SGX* technology. *SGX* is an evolution of trusted code execution and trusted platform. Compared to previous technologies such as *ARM TrustZone*, the TCB is much smaller in *SGX*, and the only source of trust is *Intel* and the CPU boundaries. Such configuration makes it a very attractive and promising technology for digital rights management (DRM), where the content provider and distributors can be assured of the protection of their content from theft. Another venue where *SGX* is attracting attention is in trusted code execution on untrusted cloud platforms, since the users do not need to trust the cloud service provider, the OS or the VMM. Furthermore, they have the capability to attest their enclaves remotely. Note that, at the time of writing this paper, the first generation of *SGX* is not available on server-end CPUs and is targeted towards client computers.

4.1 Digital rights management (DRM)

Digital rights management (DRM) refers to techniques and mechanisms used to restrict access to digital content and material, mostly sought after by content distributors for profit and revenue. There are many DRM technologies available, proposed and deployed by different companies and alliances to address different issues and mitigate against evolving and ever more complex DRM circumvention tools and techniques. For example, *Google* products use *Wivedine*; *Netflix* and *Microsoft* products rely on *Microsoft*'s *Play Ready*; and *Apple* uses the in-house *FairPlay*. DRM technologies are mostly based on a few functionalities, namely key management, rights management, and a secure playback mechanism for audio and video [19]. To address the incompatibility issue of different DRM technologies, in 2011, *Intel* introduced *UltraViolet* in the '*Sandy Bridge*' family of CPUs. *UltraViolet* is not a DRM, but a cloud-based system that contains several DRMs to unify different schemes [19].

Given the capability of the enclaves to guarantee the secrecy of their data and availability of remote attestation, content providers and distributors can use *SGX* to deploy a DRM technology. Furthermore, to secure the transmission of the content on the bus to the GPU, they can use *Intel*'s Protected Audio Video Path (PAVP) and High-bandwidth Digital Content Protection (HDCP). These technologies protect the audio and video flow in the graphic processor unit (GPU) by sending the GPU the encrypted data and having the GPU decrypt the data. Even though DRM technologies can rely on *TrustZone* as well, its two shortcomings are persistent non-volatile storage for device keys and installed licences, and secure audio and video path [19]. *Intel SGX* can make the whole process easier because of the remote attestation, secure execution and sealing.

4.2 Trusted execution on untrusted cloud platforms

As mentioned earlier, as of now, server-end *SGX*-capable CPUs are not yet available. However, previous studies have looked at the utilization of *SGX* functionalities and services for trusted verifiable code execution on untrusted cloud providers, as discussed in the following.

VC3 [20] allows the execution of Hadoop Map-Reduce jobs on an untrusted platform, while keeping the data and code secret. *VC3* excludes the OS, hypervisor and Hadoop framework from the TCB, and works on the unmodified Hadoop platform. *VC3* relies on *SGX* functionalities and services, such as memory isolation, to achieve this. To deploy tasks, users implement their map-reduce code in C++, encrypt them, bind them to the code that implements the *VC3* protocol, and upload their encrypted code to the cloud. After the code is loaded, the map and reduce functions will be decrypted, and the distributed computations will run. To ensure the integrity of the computations, *VC3* uses a job execution protocol where nodes produce a summary of their computations and aggregate them. Later, the user can verify that the cloud provider did not interfere with the computations, by reviewing the aggregate summaries.

Haven [21] introduces the concept of shielded execution, a reverse sandboxing mechanism to protect the confidentiality and integrity of the application from a malicious OS, or hypervisor. It ensures the secrecy and confidentiality of the application's code and data. Furthermore, if the application executes, it will produce verifiable correct results. This means that the users can be assured that the software executed correctly. Haven allows the shielded execution of unmodified software on the *Windows* platform. It relies on *SGX* for isolation and protection of the

software from the privileged system software, Iago attacks [22], and other unprivileged software and processes.

5. SGX SOFTWARE DEVELOPMENT AND LIBRARIES

As of the time of writing this paper, the *SGX* capability is only available for the *Microsoft Windows* platform. However, according to *Intel*, a *Linux* Software Development Kit (SDK), will be available in June 2016 [23]. Currently, only *Visual Studio* Integrated Development Environment (IDE) has support for *SGX* programming. Furthermore, there is an *SGX* simulator available for the *Windows* platform, which allows the simulation of *SGX* programs on non-*SGX* CPUs. Note that the simulator is neither performant, nor does it provide the actual *SGX* secrecy guarantees, since it works at the software level.

Even though the whole program can run inside an enclave, this is not the recommended approach, since: 1) the enclaves' memory size is very limited, 2) enclaves do not have direct access to the peripherals, I/O devices and some of the system calls, and 3) increasing the size of TCB can lead to a higher error rate and an increase in vulnerabilities. The recommended *SGX* programming model is to redesign ad split applications into two different sections. One section for secure and information-sensitive functionalities that run inside an enclave, and another section for general operations. The *SGX* does not support dynamic library loading for enclaves. Programs need to be statically linked, and the libraries also should not have external dynamic dependency. Everything should be compiled as a single static binary blob. The calls from the untrusted application to inside an enclave are called ECalls, and the calls from inside an enclave to the untrusted application are called OCalls. These interfaces enable the interaction between the enclave and the application.

The *Intel SGX* SDK provides a set of trusted static libraries that can be used inside an enclave. These libraries provide sets of functionalities, such as standard C library (sgx_tstdc.lib), standard C++ libraries and STL (sgx_tstdcxx.lib), cryptographic functions (sgx_tcrypto.lib), and trusted key exchange (sgx_tkey_exchange.lib) [24].

CONCLUSION

SGX is a new functionality introduced by *Intel*, in its sixth-generation CPUs (code-named *Skylake*), which allows the launch and execution of secure enclaves. In this paper, we have presented an overview of the *SGX* internals, its use cases, the programming model, and the available libraries. *SGX* can be used for a range of sensitive applications, from digital rights management to trusted code execution on untrusted platforms. As of the time of writing this paper, *SGX* is limited to the *Windows* operating system. Furthermore, at this moment the only IDE available for *SGX* programming is *Visual Studio 2012*. Even though *SGX* does not provide any security measure against side channel attack, power analysis attack, and low-level hardware attacks, it would be interesting to evaluate the difficulty and accuracy of such attacks. Another issue that may limit the adoption and deployment of the *SGX* platform is the current licensing mechanism. However, unlike many other previous TEE attempts, *SGX* has the potential of gaining widespread adoption because of its small TCB and affordable low cost.

REFERENCES

[1] Chen, X.; Garfinkel, T.; Lewis, E. C.; Subrahmanyam, P.; Waldspurger, C. A.; Boneh, D.; Dwoskin, J.; Ports, D. R. Overshadow: A Virtualization-Based Approach to Retrofitting Protection in Commodity Operating Systems. International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2008.

[2] Hofmann, O. S.; Kim, S.; Dunn, A. M.; Lee, M. Z.; Witchel, E. InkTag: Secure Applications on an Untrusted Operating System. International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2013.

[3] Zhang, F.; Chen, J.; Chen, H.; Zang, B. CloudVisor: Retrofitting Protection of Virtual Machines in Multi-tenant Cloud with Nested Virtualization. Symposium on Operating Systems Principles (SOSP), 2011.

[4] Sahai, A. Computing on Encrypted Data. International Conference on Information Systems Security, 2008.

[5] Gentry, C. A fully homomorphic encryption scheme. 2009.

[6] Intel. Intel Trusted Execution Technology: White Paper.

[7] Davenport, S.; Ford, R. SGX: the good, the bad and the downright ugly. 2014. https://www.virusbulletin.com/virusbulletin/2014/01/sgx-good-bad-and-downright-ugly.

[8] Rutkowska, J. Thoughts on Intel's upcoming Software Guard Extensions (Part 1). 2013. http://theinvisiblethings.blogspot.com/2013/08/thoughts-on-intels-upcoming-software.html.

[9] Rutkowska, J. Thoughts on Intel's upcoming Software Guard Extensions (Part 2). 2013. http://theinvisiblethings.blogspot.com/2013/09/thoughts-on-intels-upcoming-software.html.

[10] Brasser, F.; Kim, D.; Liebchen, C.; Ganapathy, V.; Iftode, L.; Sadeghi, A.-R. Regulating ARM TrustZone Devices in Restricted Spaces. ACM International Conference on Mobile Systems, Applications, and Services (MobiSys), 2016.

[11] Zhang, N.; Sun, K.; Lou, W.; Hou, Y. T. CaSE: Cache-Assisted Secure Execution on ARM Processors. 37th IEEE Symposium on Security and Privacy (Oakland), 2016.

[12] Anati, I.; Gueron, S.; Johnso, S. P.; Scarlata, V. R. Innovative Technology for CPU Based Attestation and Sealing. International Workshop on Hardware and Architectural Support for Security and Privacy (HASP), 2013.

[13] McKeen, F.; Alexandrovich, I.; Berenzon, A.; Rozas, C.; Shafi, H.; ShanbhogueV.; Savagaonkar, U.

Innovative Instructions and Software Model for Isolated Execution. 2nd International Workshop on Hardware and Architectural Support for Security and Privacy (HASP), 2013.

[14] Costa, V.; Devadas, S. Intel SGX Explained. Cryptology ePrint Archive: Report 2016/086, 2016.

[15] Intel. Intel Software Guard Extensions: Intel Attestaion Service API. 2016.

[16] Gueron, S. A Memory Encryption Engine Suitable for General Purpose Processors. Cryptology ePrint Archive, Report 2016/204, 2016.

[17] Brickell, E.; Li, J. Enhanced Privacy ID from Bilinear Pairing for Hardware Authentication and Attestation. IEEE Second International Conference on Social Computing (SocialCom), 2010.

[18] Johnson, S.; Scarlata, V.; Rozas, C.; Brickell, E.; Mckeen, F. Intel Software Guard Extensions: EPID Provisioning and Attestation Services. Intel, 2016.

[19] Ruan, X. Platform Embedded Security Technology Revealed, Apress, 2014, p. 272.

[20] Schuster, F.; Costa, M.; Fournet, C.; Gkantsidis, C.; Peinado, M.; Mainar-Ruiz, G.; Russinovich, M. VC3: Trustworthy Data Analytics in the Cloud using SGX. Symposium on Security and Privacy, 2015.

[21] Baumann, A.; Peinado, M.; Hunt, G. Shielding Applications from an Untrusted Cloud with Haven. USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2014.

[22] Checkoway, S.; Shacham, H. Iago attacks: why the system call API is a bad untrusted RPC interface. Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2013.

[23] Z. Dan. Intel Software Guard Extensions SDK for Linux Availability Update. 11 4 2016. https://software.intel.com/en-us/blogs/2016/04/11/intel-software-guard-extensions-sdk-for-linux-availability-update.

[24] Intel. Intel Software Guard Extensions Evaluation SDK for Windows OS. 2016.