TEEnder: SGX Enclave Migration using HSMs

João Guerreiro
Instituto Superior Técnico

Abstract—Intel Software Guard Extensions (SGX) is a new method of enhancing application security by creating safe areas of memory (enclaves) where data and code are protected from inspection and tampering.

This technology is being applied to cloud computing as well, however, software deployed with SGX enclaves is complex to migrate between machines using traditional methods as SGX uses unique hardware keys for data sealing.

This paper proposes a novel method of migrating SGX enclaves between different machines using Hardware Security Modules (HSMs) to encrypt and decrypt data using HSM generated keys. The use of HSMs reduces the amount of time needed to migrate a machine with large enclaves and provides a higher degree of security controlled by the developer.

The result is a proof of concept implementation that does not depend on the security of remote attestation during the migration phase and achieves faster migration with large data sizes or concurrent enclave migrations.

Index Terms—HSM, Intel SGX, Enclave Migration

1 Introduction

Trusted Execution Environments (TEEs)[26] are hardware solutions offering code and data protection with respect to confidentiality and integrity. Intel Software Guard Extensions (SGX) allows the creation of TEEs called enclaves which protect application data and secrets inside the processor and encrypts them outside in system memory, protecting those secrets from various attacks such as memory dumping, rogue kernel injection and data tampering.

Cloud service providers become increasingly vital to software deployment in terms of scale and efficiency as more software moves to the cloud. This, however, has raised security concerns of sensitive data being processed on untrusted and off-premises machines. Service providers, like Azure, have begun to provide virtual machines with SGX support in order to address this concern of sensitive data offload as well as due to increased privacy regulations.

One of the problems presented by the use of TEEs in cloud datacenters is, however, the complexity of migrating enclaves to other machines as their data is encrypted with a key derived form an unique CPU hardware key. This makes virtual machine (VM) migration, where data is simply copied and execution context is restored, not enough to migrate enclaves. This has a potential impact on datacenter service uptime, load balancing and power consumption.

The only work that covers the development of a secure migration protocol for enclave data is by Alder et al. [2] but it doesn't explore the performance of large enclave migration, concurrent migrations and disaster recovery.

Increased privacy regulations have also imposed the use of Hardware Security Modules (HSMs), dedicated appliances for private key protection and cryptographic operations with auditing support and strict access policies. Several cloud service providers have, as well, made available HSM services for use by their clients.

The work here presented has the objective of using HSMs to facilitate a secure and efficient migration of SGX enclaves between separate machines in a datacenter environment. We also studying the integration of these two privacy protecting technologies.

In summary, this paper makes the following contributions:

- Identification of the challenges to integrating applications with HSMs and enclaves.
- Evaluation of SGX-VM migration using HSMs in a cloud environment.
- Implementation of a proof-of-concept based on previous work and compare the performance of our solution with a baseline implementation, while also evaluating the security and usability of our approach.

In this paper, we start by presenting the basic concepts of SGX and HSMs and analyzing the related work in Section 2. We propose a new managed migration solution in Section 3. Following that, we present and analyze the results in Section 4 and Section 5. Finally, we draw conclusions and pointing out possible future work in Section 6.

2 Related Work

In this section we present the basis of SGX enclaves and all the current work done on SGX enclave migration, following with an explanation of Hardware Security Module technology and deployment.

2.1 Intel Software Guard Extensions

SGX[18] is an extension to the x86–64 instruction set designed to protect critical application's runtime and data from a potentially malicious system stack, from the operating system to hardware. SGX creates a hardware encrypted memory region called an enclave within the protected application, so that neither compromised operating systems, nor hardware attacks such as a cold-boot attack[30] can retrieve the application secrets.

These enclaves work as private regions of memory that are protected even from processes running at higher privilege levels. An application can contain various enclaves which are dynamically created and destroyed on demand. This results in an application divided in two parts, a trusted component and an untrusted component as show in fig. 1.

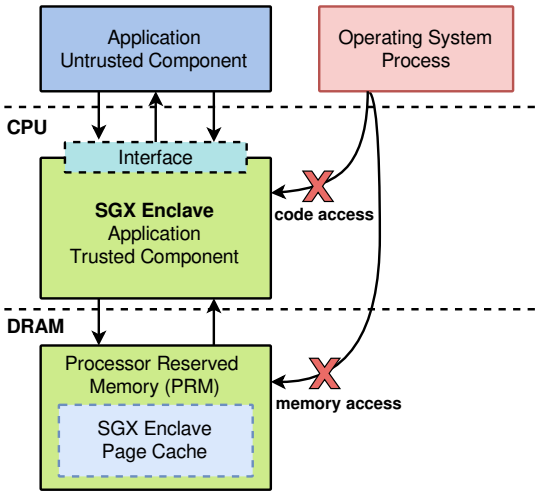


Fig. 1. Software partitioning in SGX.

Enclave pages are kept in the Processor Reserved Memory (PRM) which is protected from Operating System access. Enclave pages are encrypted using a Memory Encryption Engine (MEE)[10] with a specific enclave key to ensure page confidentiality and integrity.

The untrusted component can then call functions from the trusted component through a strict interface defined in a manifest file and receive their output but is unable to inspect the function data.

Running these applications requires the operating system to have the required drivers to interface with SGX hardware and Intel's architectural enclave service that creates the enclaves required for SGX's attestation mechanisms.

2.1.1 Local Attestation

This type of attestation[13, 18] allows a local enclave to prove to another local enclave it is running on real SGX hardware and provide its code signature (*MRENCLAVE*). Along with this confirmation, the source-enclave can send 512 bits of additional data (report-data) to claim knowledge of a determined secret.

The *MRENCLAVE* is the result of the cryptographic log generated during Enclave construction, it produces an hash (SHA256) based on the content of the enclave (code, data, stack and heap), the location of each page within the enclave and other enclave specific information such as security flags used.

2.1.2 Remote Attestation

Remote attestation[3, 17] is used for attestation between a local enclave and an enclave on another CPU when trust between two applications running on separate hosts needs to be established. It requires a third enclave called Quoting Enclave, provided by Intel, running on the host of the local enclave and on the host of the target enclave.

The quoting enclave verifies and transforms the REPORT (locally verifiable) into a QUOTE (remotely verifiable) by signing it with EPID, a group signature scheme verifiable by Intel's Remote Attestation Service. This third party who has knowledge of each of the host's provisioning keys, the keys used for remote attestation, can confirm the validity of the hardware and code signature produced between enclaves.

2.1.3 Vulnerabilities

There have been concerns about possible man in the middle attacks[28] to the remote attestation mechanism and vulnerability to micro-architecture flaws[6, 27, 29].

Zombieload[27] is able to read arbitrary kernel memory by exploiting the fill buffer of the CPU by faulting load instructions allowing an attacker to leak secrets from the application. In specific the researchers demonstrate the compromising of some SGX primitives that an attacker could use to undermine Intel's architectural quoting enclaves and thereby breaking remote attestation.

2.2 Hardware Security Modules (HSMs)

HSMs are physical appliances with secure cryptographic storage, hardware cryptographic engines and a restricted API for operation. They are used as the root of trust for many services, providing safe storage for cryptographic keys and digital identity while being compliant with various international norms, such as FIPS[21] and Common Criteria[14].

HSMs provide various desirable features to security minded organizations: (i) Symmetric and Asymmetric key storage, (ii) Hardware encryption, decryption, signing and verification of data without exposing private keys, (iii) Hardware random number generation and (iv) Large entropy availability.

The HSM bootstrapping procedure starts with a designated Security Officer initializing the HSM, creating an HSM partition and managing the distribution and care of PINs and access keys, among the users of HSM partitions.

Once this process is done and the client (application using an HSM) is in possession of a PIN, a trusted link must be established between the HSM and the client to ensure a encrypted connection. This is done by exchanging public keys before the HSM is used by the client. The client's public and private key and server's public key must be kept in the client's host as well as a configuration of the server's location. Access to HSM functionalities by an application can then be done via password authentication.

For communication Safenet HSMs use NTL[5]. It protects HSM/client communications by utilizing an encrypted tunnel and by using endpoint and message authentication and verification. This type of channel is comparatively similar in operation to a SSL/TLS channel.

2.3 SGX Enclave Migration

Work on SGX enclave migration can be categorized in two varieties: enclave page cache migration and enclave data migration, the mechanisms underlying each being applicable to the other with some tuning.

Work by Park et al. [24] provides the theoretical basis for the field, exploring the challenges inherent to state

migration between enclaves and arguing for the need of a host-wide migration middleware (migration enclave) with remote attestation for a safe transfer process.

On the subject of enclave page cache migration, Gu et al. [9] provides a first implementation on OpenSGX[15], an open platform for SGX research, followed by Park et al. [23] with another implementation on OpenSGX. Their work focuses on adding new instructions to SGX in order to enable the safe migration of enclave page cache, meaning they don't deal with the issue of CPU unique keys and sealed data.

On the subject of enclave data migration, Alder et al. [2] builds on the communication mechanisms of Park et al. [24] and provides support for migrating persistent data and monotonic tokens on hardware versions of SGX. An overview of the solution architecture is presented in fig. 2.

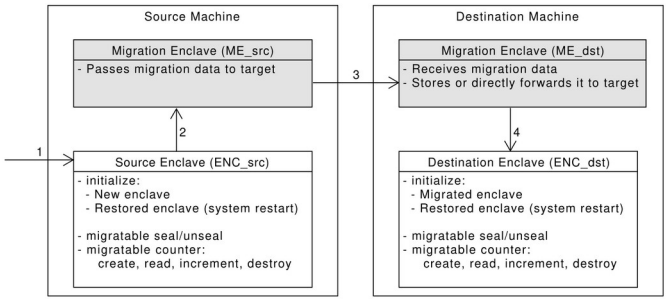


Fig. 2. Alder et al. [2] solution architecture. Source:[2]

Alder et al. [2] solves the issues with enclave migration by using a SGX-only with the following steps:

1) The source enclave (ENC_{src}) receives a command to migrate data and its destination. Data is then sealed using a randomly generated key.
2) The key and data are sent to the source machine migration enclave (ME_{src}).
3) A channel is established using remote attestation with the destination machine's migration enclave (ME_{dst}). The data and key are transferred using this channel.
4) ME_{dst} provisions a newly created enclave (ENC_{dst}) with the data and key. ENC_{dst} unseals the data with the provided key and sets its state.

However, these solutions ([2, 9, 23]) are reliant on the security of remote attestation during migration as their communication channel established with remote attestation carries both the sealed data and sealing key. If key negotiation in these solutions is broken, through cryptographic flaw or a man-in-the-middle attack, enclave data is leaked.

By compromising the hosts' migration enclave, access can be obtained to the data as the migration enclave has access to the sealed data and the sealing key and relies on the remote attestation's guarantees to guarantee non-tampering.

Although the solutions mentioned evaluate solution performance, they do not evaluate real world scenarios such as large data sealing, concurrent migrations or disaster recovery.

2.4 Current cloud environment

In the context of the current cloud environment, a variety of cloud service providers, such as IBM[11] and Azure[19, 20], now provide SGX and HSM services in the form of IaaS.

These SGX services suffer the issue of having no VM migration mechanism and no integration with HSM services.

3 TEEnder

In this section we present TEEnder, a solution to the current limitations of enclave data migration with respect to performance and security. We start by analyzing the deployment environment and threat model of the solution. Next, objectives are laid out for its architecture and implementation which are thus presented after.

3.1 Environment

TEEnder is targeted at deployment in datacenters held by service providers in an Infrastructure as a Service (IaaS) model who provide HSM services or data-sensitive industry datacenters which hold their own HSMs with strict access control.

In datacenters, some of the most important factors and quality of service measures are service performance, uptime and power consumption. Virtual machine live migration[7] was developed to provide uninterrupted service during maintenance, for moving computation away from failure-prone hosts and for saving power by effectively distributing load across the data center[1].

Our solution should be applicable in a scenario where a virtual machine containing services with enclaves needs to be migrated to another physical host for the reasons mentioned above. Figure 3 shows the envisioned environment for the deployment of the proposed solution.

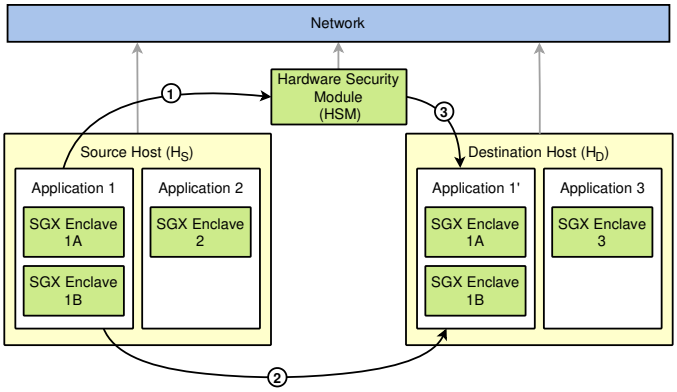


Fig. 3. Environment architecture

The process would be split into three phases (represented from 1 to 3 on fig. 3):

1) Transfer data from H_S to the HSM and encrypt it on the HSM.
2) Return encrypted data to H_S and send it to H_D.
3) Decrypt the data on the HSM and set enclave state on H_D.

3.2 Threat Model

Our trusted computing base (TCB) assumptions are based on the combination of SGX's[18] and the HSM's TCB.

On the SGX side, we consider the CPU and the code running in an enclave as its only TCB on the host, meaning

the OS and the hypervisor the software is running on is considered as untrusted. On the HSM side, we consider the appliance and its code as safe. On this conjugation we must exclude PCI HSMs as they are in the host machine but outside of SGX's TCB leaving us only with network HSMs as safe appliances.

We use an attacker model based on motivation, skill level and available privileges to identify the threats to our solution. Our attackers motivation is to leak data from inside an enclave and his skill level allows him to do attacks such as: Network sniffing and spoofing, data tampering, rogue kernel injection and hypervisor tampering.

SGX does not provide availability guarantees[16], as expected of a threat model where the attacker has physical access, therefore we focus mostly on data integrity and confidentiality

We assume the attacker is unable to subvert the security guarantees of SGX or the HSM. The attacker is not assumed to be in control of the system during initial setup of the HSM infrastructure or initial provisioning of the enclaves. Only acquiring the varied privileges during program execution. We further assume the communication between SGX and HSM is done solely through TLS with safe system calls and data resulting from that communication is only loaded inside of enclave memory space.

3.3 Requirements

TEEnder presents a series of requirements based on the technologies it uses and the data sensitive industries it fits in order to achieve a secure migration of data between enclaves.

As security requirements we identify:

R$_S$1. The migration system must not add new attack vectors to the host machine.

R$_S$2. The system provide end to end protection of the migrated data.

R$_S$3. Sealed migration data must only be unsealed by an authorized machine.

R$_S$4. Data must not be migrated to rouge machines outside the datacenter controlled by the attacker.

R$_S$5. An audit log must be kept for the system to keep track of sealing and unsealing of data.

R$_S$6. Resist compromise of remote attestation during migration. Data must not unsealable by attackers.

In a real world application a machine is likely to run N enclaves of various different sizes, as opposed to running just one enclave of fixed size, and during a full machine migration all of these will be migrated at the same time.

Therefore, as performance requirements we identify:

R$_P$1. The system must handle large enclaves without serious loss of performance.

R$_P$2. The system must do concurrent sealing of data without serious loss of performance.

3.4 Architecture

TEEnder can be broken into three components: HSM, migration library and migration middleware.

The HSM, is responsible for encrypting and decrypting the migration package on behalf of the source and target machine respectively.

The migration library, is responsible for packing the enclave's data into a migration package and sending it to the migration middleware for dispatch.

The migration middleware, is based on the solution devised by Alder et al. [2] and Park et al. [23] which is implemented with enclaves and similar in design to Intel's architectural enclaves, such as the Quoting Enclave (QE) for remote attestation and Launch Enclave (LE) for enclave creation. The middleware can be deployed in the same virtual machine as the application or in a separate virtual machine. Its function is to create a channel between the two machines' migration middlewares for transporting the sealed data and provisioning the newly created enclave on the target machine with the sealed data.

Figure 4 provides an overview of the infrastructure setup and the communication between each component of the system.

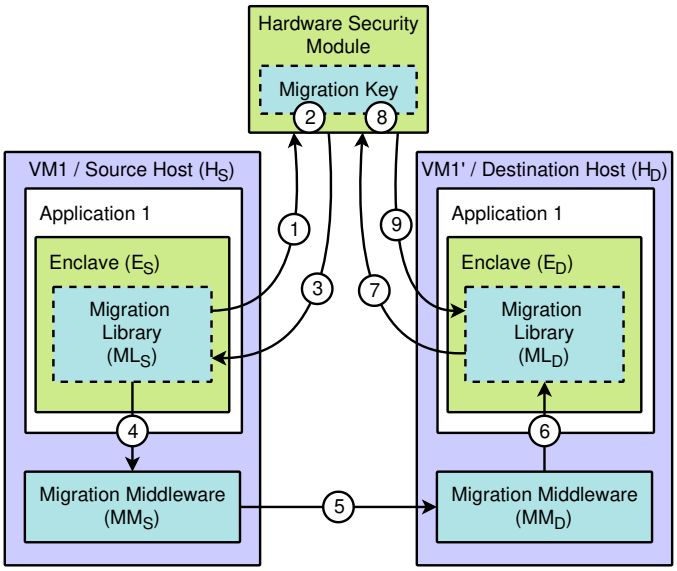


Fig. 4. Design Architecture

The process of migration should be as follows after a migration command is given to the Application in the Source Host:

1) The ML_S sends the data of E_S to the HSM.
2) Data is encrypted inside the HSM using a key generated and stored in the HSM.
3) The encrypted data is returned to the ML_S.
4) The encrypted data is sent from the ML_S to the MM_S.
5) The encrypted data is sent to the MM_D.
6) The MM_D transfers the data to an inactive enclave E_D.
7) The ML_D inside the E_D sends the encrypted data to the HSM.
8) Data is decrypted inside the HSM using the key stored in the HSM.
9) The decrypted data is sent to the ML_D. The decrypted data can now be used to reset enclave state and confirmation can be sent to the MM_S to allow deleting E_S.

3.4.1 Application Architecture

An example application layout is presented in Figure 5, modifications are only done inside of the applications enclave.

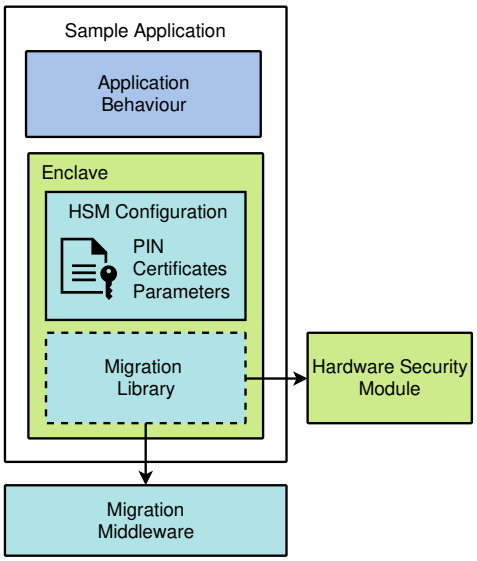


Fig. 5. Application architecture

The enclave must have the HSM pin, HSM certificate and HSM parameters in order to communicate with the HSM. These secrets can be initially provisioned using remote attestation, local attestation or a protected code loader. The PIN allows the application to interface with the HSM while the HSM certificate allows verification that the HSM at the saved address is the intended HSM.

3.5 Key Management

In line with HSM standards, encryption keys should never leave the HSM therefore encryption operations must be done by the HSM. This access to the HSM is controlled with a PIN that is generated by the HSM Security Officer and provisioned with the application.

After authenticating with the HSM the application then proceeds to request an existing key handle using a label or a key characteristic as filter.

Enclave migration can therefore be encrypted in two ways:

3.5.1 One Time key

Based on the same principle as Alder et al. [2]'s solution, an enclave on machine A generates a key on the HSM with the label matching the *MRENCLAVE* of the enclave. AES-CBC key derivation is used on the generated key in the HSM with an initialization vector generated randomly by the HSM to bind the data to a single migration process. Finally, the derived key is used for encrypting the migration data.

The initialization vector is then sent along with the migration data to the new machine's enclave where it is used to derive the key on the HSM. Once the decrypting of the migration data is done and the target enclave is restored, the key in the HSM can be deleted preventing the migration data from being reused by another machine or the source machine.

3.5.2 Migrateable data backup key

For data backup purposes, a key is generated on the HSM with the label matching the *MRENCLAVE* of the enclave and the backup date. The data encrypted using this key will be valid as long as the key exists in the HSM. For extra protection against rollback attacks the key can be created as an encryption only key, requiring a security officer to manually change its parameters to allow for decryption as well.

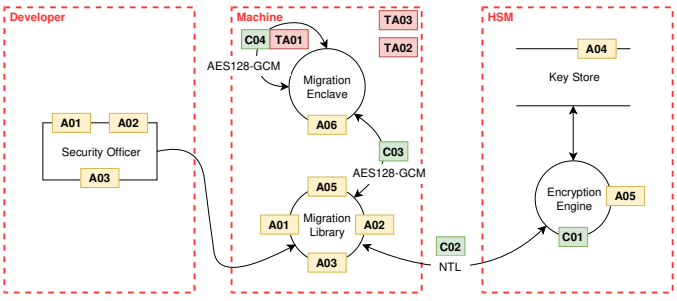
3.6 Bootstrap procedure

The bootstrap procedure for the solution can be divided into three parts: Initial HSM deployment, migration middleware deployment and data provisioning.

1) The Security Officer sets up the HSM with a partition for use in the migration procedure and generates the necessary access PINs.
2) For every new virtual machine added to the datacenter, a migration middleware must be deployed on it, this can be done automatically.
3) the application must be provisioned with the necessary PINs and certificates. This provisioning can be done through Intel's Protected Code loader[12] or through another method using remote attestation.

3.7 Implementation

In this section we explain how we put the design of TEEnder into practice. Figure 6 provides an overview of the assets held by each component of the system and the communication mechanisms between each of them.



Assets	
ID	Description
A01	HSM PIN
A02	HSM Public Certificate
A03	HSM Client Configuration
A04	Migration Key
A05	Unsealed Data
A06	Sealed Data

Security Controls	
ID	Description
C01	Password Authentication
C02	NTL Certificate Authentication
C03	Local Attestation
C04	Remote Attestation

Threat Actors	
ID	Description
TA01	Trusted Third Party
TA02	Cloud Service Provider
TA03	System Administrator

Fig. 6. Data Flow Diagram

In fig. 6 identify potential threat actors that must be taken into account when developing the implementation such as: i) A malicious system administrator (TA03) modifying host software. ii) Malicious or insecure system images deployed by the Cloud Service Provider (TA02). iii) The compromising of the remote attestation trusted third party (TA03).

To ensure encrypted end-to-end communication all transport channels are encrypted. For communication between enclaves in the same machine Local attestation (C03,2.1.1) is used to establish an encrypted channel. For

communication between enclaves in different machines (Source migration middleware to Destination migration middleware) Remote attestation (C04,2.1.2) is used to establish an encrypted channel. Communication between the Migration Library and the HSM is done through NTL (C02,2.2), using pre-provisioned certificates. Authentication for the HSM functions is done through an application specific PIN (C01) explained in section 2.2.

Finally as a combination of both fig. 4 and fig. 6 we arrive at out final implementation diagram: fig. 7 which shows the migration process as the interactions between the HSM and two hosts: Source (H_S) and Destination (H_D). For the migration process to begin, H_S and H_D must have their migration middlewares (MM_S and MM_D) running and be able to resolve the HSM's address. Both hosts' enclaves must be have their migration libraries (ML_S and ML_D) pre-provisioned with HSM access PINs. In case of a VM migration, the H_D's enclave would be a blank slate of the H_S's enclave as VM migration only recreates the enclave but doesn't reset the enclave's state.

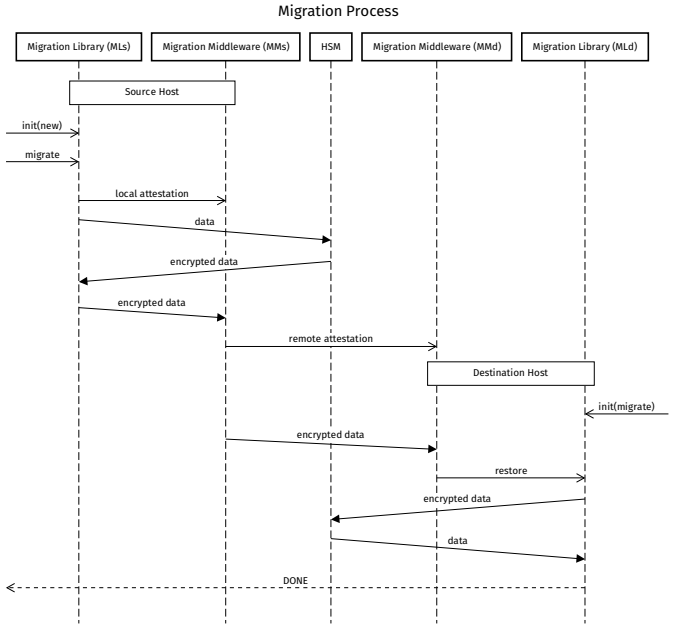


Fig. 7. Protocol Diagram

Unfilled arrows (→) represent procedures and filled arrows (I) represent data movement.

The application's migrateable data is sent from ML_S to the HSM through a NTL channel (2.2) and encrypted using the HSM.

The key used for encrypting the data can either be a derived one time key in the HSM as presented in section 3.5.1 or a label matching key as presented in section 3.5.2. These keys are kept safe from unwanted access by use of two mechanisms: an HSM partition PIN and a certificate based authentication. The key is selected by label so it is never moved to the application, as the HSM only needs a key handle to proceed with decryption.

The encrypted data is then sent without the sealing key attached to the migration middleware (MM_D) through the MM_S on H_S by a remote attestation established TLS channel.

The MM_D sends the encrypted data to the Migration Library ML_D using local attestation to establish a TLS channel.

The ML_D send the data to the HSM through NTL where it is decrypted following the same key scheme used for its encryption.

Once the data is decrypted by the HSM it is sent back to the ML_D in the H_D enclave. It is up to the enclave to implement its own state setting mechanism. Confirmation of successful migration is sent back to ML_S, which can then delete its data and finish running.

The implemented application layout is presented in fig. 8 and the major differences when compared with fig. 5 are the use of the host system's certificates for HSM communication and using ocalls, enclave exits and reentries, to communicate with the HSM using a *PKCS#11* library provided by HSM vendor Safenet.

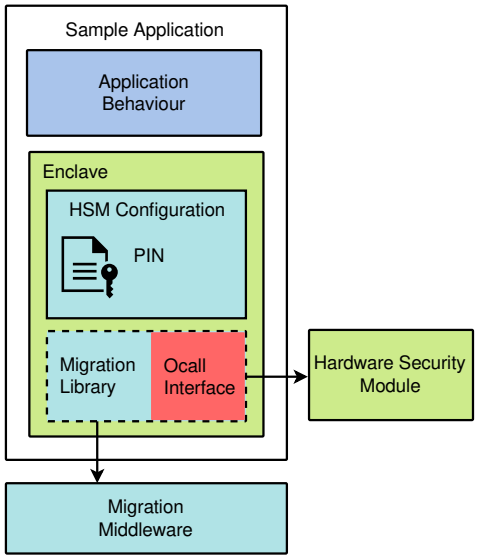


Fig. 8. Implemented application layout

The implementation mostly modifies the migration library's data sealing path but keeps a similar structure for compatibility reasons.

The implementation should be interfaceable with any appliance that implements the *PKCS#11* interface specification[22] for communication as HSMs do not deviate from *PKCS#11* .

4 Results

In this section we present the results from our proof of concept implementation of TEEnder. We start by presenting the proof of concept implementation and follow with performance evaluations on increasing data size, multiple enclaves and increasing latency.

4.1 Performance evaluation

In evaluating the performance of the solution, we compare our proof of concept with a SGX only solution and vary the variables we consider most affecting of enclave migration: data size and concurrent migrations.

Data size is a decisive variable in SGX1 performance as Enclave Page Cache (EPC) is limited to 128MB and swapped when needed. As well, possibly due to this swapping as data size increases, there are larger overhead times[31, 32]. The other possible performance affecting variable is the number of enclaves being migrated as they compete for the same hardware when sealing their data as shown in Gjerdrum et al., as the creation of multiple enclaves increased latency.

We also test network latency as a variable that could have an impact on HSM total sealing time to evaluate the use of on-premises HSMs in use with cloud machines.

Table 1 presents the various environments used for the performance evaluation.

4.1.1 Increasing data size

One of the first objectives in the performance evaluation is to verify using HSMs doesn't affect single enclave migration performance to the point of infeasibility.

32 tests were run of data sealing with data size ranging between $0B$ and $200MB$ on TEEnder with HSM support enabled and disabled. Each of these tests ran 20 times in order to get an average of data sealing time. In Figure 9, we find that for smaller data sizes SGX migration sealing performance is higher than the HSM enabled system, as the size of the data to seal increases, the HSM sealing performance degrades less than the SGX system.

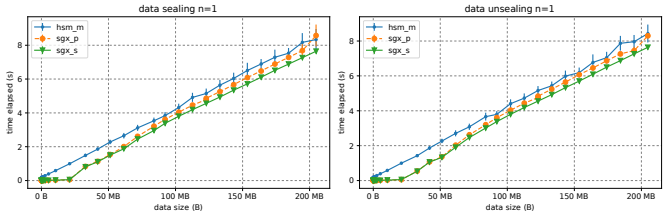


Fig. 9. Effect of increasing data size on SGX and HSM data sealing

In fig. 9, no concrete difference can be discerned between sgx_p and sgx_s meaning the SGX hardware platform the code was run on had no influence on the results.

HSM sealing time follows a linear regression tightly matched by SGX's linear regression past 100MB of enclave data size.

4.1.2 Multiple enclaves

In a real world application, a machine is likely to run N enclaves as opposed to running just one enclave and during a full machine migration all of these will be migrated at the same time.

Tests were conducted with an increasing number of processes ($1 < n < 10$) and with an increasing data size $1024B < s < 100MB$ in order to determine the feasibility of the HSM enabled system on simultaneous migration.

A sample of these tests is presented in Figure 10 for $n = 4, 10$ showing a decrease on the data size required for SGX data sealing time to exceed HSM data sealing time.

Figure 11 was made from the linear interpolation of the average elapsed time until conclusion of data sealing. 20 simulations for each number of concurrent processes ($1 <$

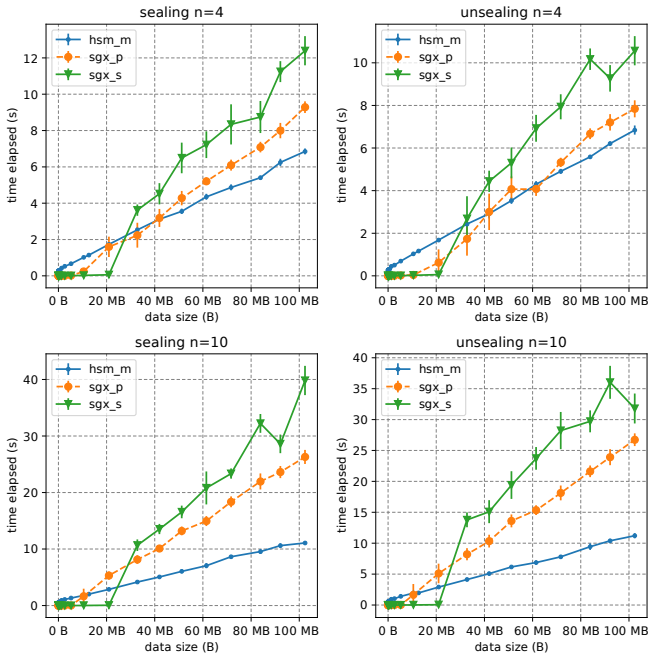


Fig. 10. Plot of data sealing time in relation to increasing number of processes

$n < 10$) and for 32 points between $1024B < s < 100MB$ were conducted in order to determine the feasibility of the HSM enabled system on simultaneous migration. Data size is presented in logarithmic scale.

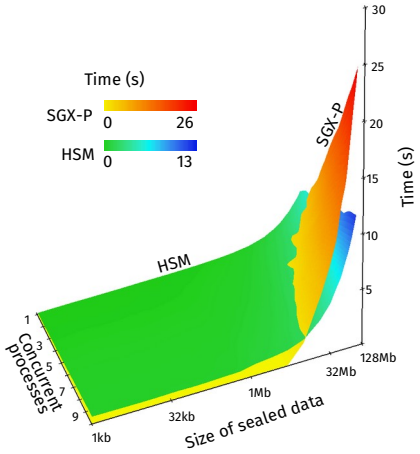


Fig. 11. Plot of sgx and hsm data sealing times.

Figure 11 shows that as datasize and number of processes increase SGX performance degrades faster than TEEnder's resulting in an intersection where beyond those data points TEEnder performance is superior.

4.1.3 HSM Latency

Figure 12 presents the relation between latency and data sealing time using TEEnder. The results are taken from a base latency of 0.5ms. Further latency is added using the linux kernel's traffic control *netem*. A sample of 10 runs of sealing, with 1024 bytes of MAC data and 10MB or 100MB

TABLE 1
Evaluation environment table

Label	Type	SGX Hardware	Threads	HSM Hardware	RTT (ms)	Additional notes
sgx-p	SGX	CPU: Intel Core i5–7200U RAM: 16GB	4@3.1GHz	N/A	0	Bare metal
sgx-s	SGX	CPU: Intel Xeon E3–1230 v5 RAM: 16GB	4@1.6GHz	N/A	0	Bare metal
hsm-m	SGX+HSM	CPU: Intel Xeon E5603 RAM: 2GB	2@1.6GHz	Luna S790	0.5	Virtual Machine. SGX in Simulation mode

of random data, were used for each value of latency to construct the plot.

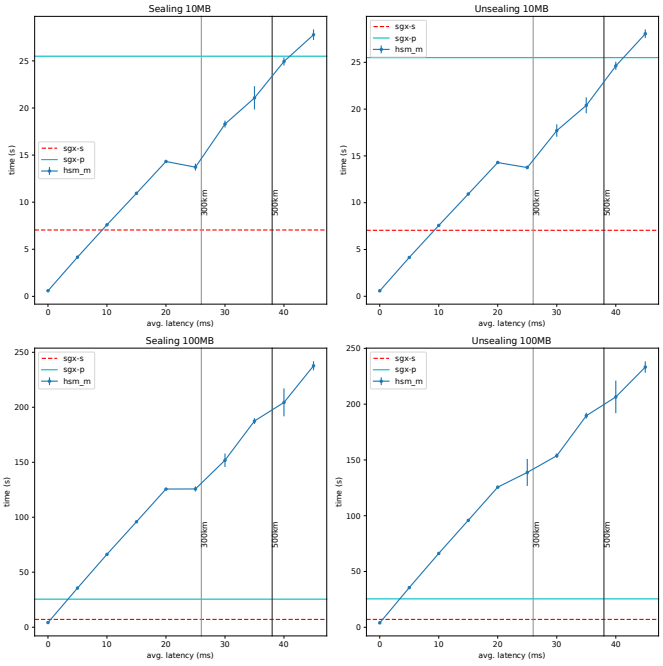


Fig. 12. HSM latency effects

The horizontal lines presented in fig. 12 are the time values of SGX only data sealing for 10MB, 100MB of data and different number of concurrent migrations (n).

The vertical lines are representations of distance by latency, as measured from our main test location to two remote locations of known distance.

We can then predict a 100MB Enclave will take around 150s to seal when the HSM is at a distance of 300km while a 10MB Enclave will take around 15s showing a higher penalization in larger enclaves due to latency.

It is possible to conclude there is a linear relation between latency and data sealing time on HSMs.

From this data, we conclude increases in latency will significantly move the intersection observed in Figure 11. We therefore consider latency a strong variable in determining the performance of TEEnder.

5 Discussion

We evaluate TEEnder based on the objectives set on section 3.3 as a measure for its success and as a basis for discussion.

5.1 Security Evaluation

Our proposed migration scheme fills the requirement set in 2 as plaintext data is only available to the HSM and the enclave but is encrypted in transport.

For 3 and 4, we confirm through the mechanisms described in section 3.5 the resolution of the first requirement. The second requirement is accomplished through the innate use of HSMs as their server-client public key scheme and identity verification forces a new server to always be verified and added by the authorized Security Officer.

We consider 5 accomplished due to the nature of HSM usage and its ability to log each operation and key usage under an audit user, allowing data to stay confidential but controlling for intrusions and unauthorized accesses.

Finally, 6 can be categorized as successful during the migration phase as remote attestation is only used to verify a legitimate SGX platform but not used for identity verification and data running through it is encrypted using AES128-GCM without the key attached.

In case remote attestation is compromised after initial deployment causing the link between the migration middleware instances to be considered unsafe, the data recovered will not include the sealing key used to encrypt it.

Two issues were identified with the usage of HSMs in an SGX environment: Trust establishment and dynamic linking.

Trust establishment and TLS tunnel establishment with the HSM is done through public key exchange. The server's public key and client's public and private key are stored in the filesystem decrypted. According to our threat model 3.2 this is a potential vector for attack.

The SGX enclave environment relies on statically compiled binaries with no external dependencies in order to provide complete isolation. The existence of external dependencies, as needed by the *PKCS#11* library used for HSM communication, involves the usage of *OCALLS* [13] which exit the enclave in order to use the system resources and libraries.

Therefore we evaluate the requirement 1 positively on a design aspect but lacking in the implementation phase.

For these issues two possible alternatives were identified:

5.1.1 Containerization

SCONE[4] is a secure container mechanism for Docker, allowing the creation of containers that run inside Processor Reserved Memory (PRM) using SGX.

An example application architecture is presented in Figure 13 using SCONE.

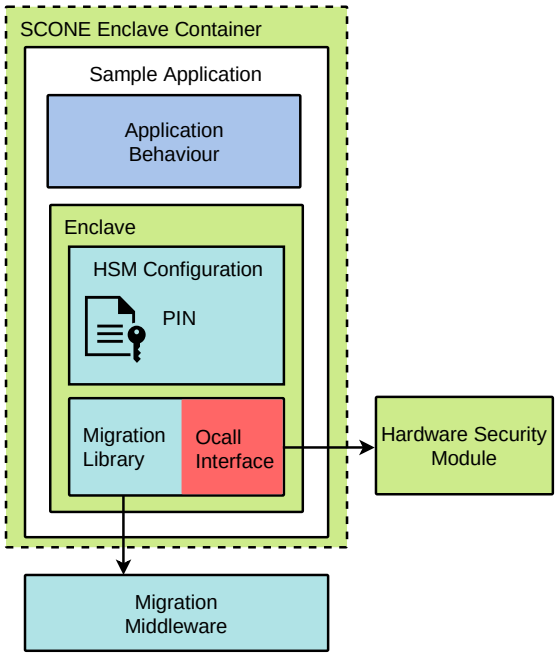


Fig. 13. TEEnder container architecture

SCONE provides a libc implementation meaning no changes should be needed from the implemented application layout (fig. 8).

Deploying the existing implementation over a SCONE base image would protect the vulnerable certificates since SCONE encrypts the filesystem and the unpartitioned library included since it would be in secure enclave memory with SCONE handling all of the syscall interfacing.

5.1.2 Vendor Changes

On the vendor side, changes can be made to allow the easier integration of HSM and SGX technologies. Allowing the loading of certificates and configurations from the application memory instead of the filesystem would solve the first issue identified as enclave memory is encrypted and trusted. Finally, the porting of the library to enclave space, namely enclaving the TLS termination by way of the TaLoS[25] framework, and establishing communication by local attestation with user applications, would fix the second issue presented while keeping the vendor's distribution model unchanged.

5.2 Performance Requirements

In terms of goals, we consider 1 and 2 achieved as an analysis of section 4.1.1 and section 4.1.2 respectively.

Latency was controlled for the execution of tests but as section 4.1.3 shows: results may differ on higher latency environments as well depending on the HSM model and CPU threads available in the case of multiple enclaves. In a migration between two different datacenters, synchronizing HSMs between datacenters would allow data to be sealed in the origin datacenter and unsealed by an HSM in the target datacenter, minimizing latency penalty to local level.

Despite the variables that need to be analysed for individual context, TEEnder can surpass SGX performance in enclave data sealing and unsealing during large enclave migration and concurrent migration.

6 Conclusion

Trusted Execution Environments (TEEs) such as Intel SGX provide strong security guarantees in a cloud computing setting and data sensitive industries, but Virtual Machine (VM) migration remains a staple of cloud datacenter operation as it allows providers to increase uptime and reduce costs. SGX Enclaves must be easily migrateable between machines in a cloud computing environment to have the lowest impact on VM migration time while retaining their security guarantees.

TEEnder implements an enclave migration system based on HSM generated keys and HSM cryptographic offloading. TEEnder retains the security guarantees of Intel SGX and guarantees secure enclave migration between hosts. As added benefits, the system is not vulnerable to attacks on remote attestation during the migration procedure and allows the solution to be audited.

In our performance evaluation, TEEnder out-performs an SGX-only system for enclaves larger than 100MB, in all other cases the impact on performance is not significative. This threshold enclave size of 100MB tends to decrease as more enclaves are migrated in parallel from the source host.

Taking into consideration the average enclave size in a cloud environment, this translates into an overall lower migration time and lower downtime during virtual machine migration.

This work opens the possibility of implementing an hybrid system which uses SGX encryption when the machine is idle for small data sizes with HSM keys and switches to using HSM encryption with HSM keys when data sizes get larger which would allow an even higher performance of the system while retaining the security advantages obtained. This work also opens new possibilities for the disaster recovery of virtual machines and their enclaves by allowing their restart on a different datacenter with low recovery time.

References

[1] Ahmad, R. W., Gani, A., Hamid, S. H. A., Shiraz, M., Yousafzai, A., and Xia, F. (2015). A survey on virtual machine migration and server consolidation frameworks for cloud data centers. *Journal of Network and Computer Applications*, 52:11–25.

[2] Alder, F., Kurnikov, A., Paverd, A., and Asokan, N. (2018). Migrating SGX Enclaves with Persistent State. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 195–206, Luxembourg City, Luxembourg. IEEE.

[3] Anati, I., Gueron, S., Johnson, S., and Scarlata, V. (2013). Innovative Technology for CPU Based Attestation and Sealing. *HASP - Int. Workshop Hardw. Archit. Support Secur. Priv.*, pages 1–7.

[4] Arnautov, S., Trach, B., Gregor, F., Knauth, T., Martin, A., Priebe, C., Lind, J., Muthukumaran, D., OKeeffe, D., Stillwell, M. L., Goltzsche, D., Eyers, D., Kapitza, R., Pietzuch, P., and Fetzer, C. (2016). SCONE: Secure Linux

Containers with Intel SGX. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 689–703. USENIX Association.

[5] AWS, A. (2019). Network Trust Links. https://cloudhsm-safenet-docs.s3.amazonaws.com/007-011136-002 lunasa 5-1 webhelp rev-a/Content/concepts/network_trust _links.htm.

[6] Chen, G., Chen, S., Xiao, Y., Zhang, Y., Lin, Z., and Lai, T. H. (2018). SgxPectre Attacks: Stealing Intel Secrets from SGX Enclaves via Speculative Execution. *ArXiv180209085 Cs*.

[7] Clark, C., Fraser, K., Hand, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I., and Warfield, A. (2005). Live Migration of Virtual Machines. In *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, pages 273–286. USENIX Association.

[8] Gjerdrum, A. T., Pettersen, R., Johansen, H. D., and Johansen, D. (2017). Performance of Trusted Computing in Cloud Infrastructures with Intel SGX:. In *Proceedings of the 7th International Conference on Cloud Computing and Services Science*, pages 696–703, Porto, Portugal. SCITEPRESS - Science and Technology Publications.

[9] Gu, J., Hua, Z., Xia, Y., Chen, H., Zang, B., Guan, H., and Li, J. (2017). Secure Live Migration of SGX Enclaves on Untrusted Cloud. In *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 225–236, Denver, CO, USA. IEEE.

[10] Gueron, S. (2016). Memory Encryption for General-Purpose Processors. *IEEE Secur. Priv.*, 14(6):54–62.

[11] IBM (2019). Data-in-use protection on IBM Cloud using Intel SGX. https://www.ibm.com/cloud/blog/data-use-protection-ibm-cloud-using-intel-sgx. (Accessed: 2019-08-05).

[12] Intel (2018). Intel(R) SGX Protected Code Loader for Linux User Guide.

[13] Intel (2019). *Intel R 64 and IA-32 Architectures Software Developer's Manual, Volume 3D: System Programming Guide, Part 4*. (Accessed: 2019-08-05).

[14] ISO Central Secretary (2014). Information technology – Security techniques – Evaluation criteria for IT security – Part 1: Introduction and general model. Standard ISO/IEC 15408-1:2009, International Organization for Standardization, Geneva, CH.

[15] Jain, P., Desai, S., Kim, S., Shih, M.-W., Lee, J., Choi, C., Shin, Y., Kim, T., Byunghoon Kang, B., and Han, D. (2016). OpenSGX: An Open Platform for SGX Research. In *Proceedings 2016 Network and Distributed System Security Symposium*, San Diego, CA. Internet Society.

[16] Jang, Y., Lee, J., Lee, S., and Kim, T. (2017). SGX-Bomb: Locking Down the Processor via Rowhammer Attack. In *Proceedings of the 2nd Workshop on System Software for Trusted Execution - SysTEX'17*, pages 1–6, Shanghai, China. ACM Press.

[17] Johnson, S., Scarlata, V., Rozas, C., Brickell, E., and Mckeen, F. (2016). Intel R Software Guard Extensions: EPID Provisioning and Attestation Services. page 10.

[18] McKeen, F., Alexandrovich, I., Berenzon, A., Rozas, C. V., Shafi, H., Shanbhogue, V., and Savagaonkar, U. R. (2013). Innovative instructions and software model for isolated execution. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy - HASP '13*, pages 1–1, Tel-Aviv, Israel. ACM Press.

[19] Microsoft (2019a). Azure Confidential Computing — Microsoft Azure. https://azure.microsoft.com/en-us/solutions/confidential-compute/. (Accessed: 2019-08-05).

[20] Microsoft (2019b). Dedicated HSM - Hardware Security Module — Microsoft Azure. https://azure.microsoft.com/en-in/services/azure-dedicated-hsm/.

[21] National Institute of Standards and Technology (2019). Security requirements for cryptographic modules. Technical Report NIST FIPS 140-3, National Institute of Standards and Technology, Gaithersburg, MD.

[22] OASIS (2015). *PKCS #11 Cryptographic Token Interface Base Specification Version 2.40*.

[23] Park, J., Park, S., Kang, B. B., and Kim, K. (2019). eMotion: An SGX extension for migrating enclaves. *Computers & Security*, 80:173–185.

[24] Park, J., Park, S., Oh, J., and Won, J.-J. (2016). Toward Live Migration of SGX-Enabled Virtual Machines. In *2016 IEEE World Congress on Services (SERVICES)*, pages 111–112, San Francisco, CA, USA. IEEE.

[25] Pierre-Louis Aublin, Kelbert, F., Dan O 'Keeffe, Muthukumaran, D., Priebe, C., Lind, J., Krahn, R., Fetzer, C., Eyers, D., and Pietzuch, P. (2017). TaLoS: Secure and Transparent TLS Termination inside SGX Enclaves. *Imp. Coll. Lond.*

[26] Sabt, M., Achemlal, M., and Bouabdallah, A. (2015). Trusted Execution Environment: What It is, and What It is Not. In *2015 IEEE Trustcom/BigDataSE/ISPA*, pages 57–64, Helsinki, Finland. IEEE.

[27] Schwarz, M., Lipp, M., Moghimi, D., Van Bulck, J., Stecklina, J., Prescher, T., and Gruss, D. (2019). ZombieLoad: Cross-Privilege-Boundary Data Sampling. *arXiv:1905.05726*.

[28] Swami, Y. (2017). Intel SGX Remote Attestation is not sufficient. *BlackHat*.

[29] Van Bulck, J., Minkin, M., Weisse, O., Genkin, D., Kasikci, B., Piessens, F., Silberstein, M., Wenisch, T. F., Yarom, Y., Strackx, R., and Leuven, K. (2018). Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. *Proc. 27th USENIX Secur. Symp.*

[30] Yitbarek, S. F., Aga, M. T., Das, R., and Austin, T. (2017). Cold Boot Attacks are Still Hot: Security Analysis of Memory Scramblers in Modern Processors. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 313–324, Austin, TX, USA. IEEE.

[31] Zhao, C., Saifuding, D., Tian, H., Zhang, Y., and Xing, C. (2016). On the Performance of Intel SGX. In *2016 13th Web Information Systems and Applications Conference (WISA)*, pages 184–187, Wuhan, China. IEEE.

[32] Zheng, W., Dave, A., Beekman, J. G., Popa, R. A., Gonzalez, J. E., and Stoica, I. (2017). Opaque: An Oblivious and Encrypted Distributed Analytics Platform. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 283–298. USENIX Association.