Computers & Security 96 (2020) 101874

Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

TEEnder: SGX enclave migration using HSMs

João Guerreiro[a,b,c], Rui Moura[c], João Nuno Silva[a,b,*]

[a] *Instituto Superior Técnico, Universidade de Lisboa, Portugal*
[b] *INESC-ID Instituto de Engenharia de Sistemas e Computadores, Investigação e Desenvolvimento em Lisboa, Portugal*
[c] *MULTICERT - Serviços de Certificação Electrónica S.A., Portugal*

ARTICLE INFO

Article history:
Received 9 October 2019
Revised 21 February 2020
Accepted 11 May 2020
Available online 26 May 2020

Keywords:
Trusted execution environment
Intel SGX
Hardware security module
Enclave migration
Cloud computing

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 achieves faster migration for large enclaves or during multiple concurrent migrations. Since the this solution does not depend on the security of remote attestation, and uses the keys stored in the HSM it provides a higher degree of security than current enclave migration solutions.

1. Introduction

Trusted Execution Environments (TEEs) (Sabt et al., 2015) 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.

Work such as ReplicaTee (Soriente et al., 2019) is starting to address the management of enclave replicas scattered around a datacenter by providing trusted mechanism for the deployment and control of replicas of a certain SGX enclaves. It allows the easy initialization of such enclaves, offering guarantees about the number of replicas, and provides a simple data storage layer mostly for immutable state sharing among enclaves.

Nonetheless, one of the problems presented by the use of TEEs in cloud datacenters still is 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, and has a potential impact on datacenter service uptime, load balancing and power consumption.

Development of a secure migration protocol for enclave data and memory was first proposed by Park et al. (2016) through the addition of new hardware instructions and a migration enclave. This proposal was later implemented by Gu et al. (2017), in eMotion (Park et al., 2019), and in the OpenSGX (Jain et al., 2016) SGX simulator. Later work by Alder et al. (2018) proposes the implementation of enclave persistent data migration through software means only.

These solutions still face two different limitations and challenges: they do not explore nor evaluate the performance effects of large enclaves or multiple concurrent migrations, neither address some security vulnerabilities on remote attestation recently theorized (Lee et al., 2017; Schwarz et al., 2019; Swami, 2017).

Currently certain businesses, such as financial, banking or telecommunications, are required to implement security regulations that need the use of Hardware Security Modules (HSMs): dedicated appliances for private key protection and cryptography operations with auditing support and strict access policies. The use of HSM has become mainstream not only in these businesses, but also in cloud service providers that are starting to make HSM ser-

* Corresponding author.
E-mail address: joao.n.silva@inesc-id.pt (J.N. Silva).

https://doi.org/10.1016/j.cose.2020.101874
0167-4048/© 2020 Elsevier Ltd. All rights reserved.

vices available for their clients. In both scenarios, HSMs are starting to co-exist with virtualization mechanisms, namely the migration of VMs between hosts.

The work here presented has the objective of using HSMs to increase security and efficiency of the migration of SGX enclaves between separate machines in a datacenter environment. This work also evaluates the gains from integrating these two privacy protecting technologies.

In summary, this paper makes the following contributions:

- Identification of the challenges to integrating applications with HSMs and enclaves.
- Optimization of SGX-VM migration in a cloud environment using HSMs.
- Reduction of the effect of current vulnerabilities in SGX remote attestation in the context of enclave migration.
- 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 Sections 4 and 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)

SGX (Anati et al., 2013; Costan and Devadas, 2016; Intel, 2019; McKeen et al., 2013) 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 (Yitbarek et al., 2017) 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.

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) (Gueron, 2016) with a specific enclave key to ensure page confidentiality and integrity.

To develop an application using SGX, Intel provides its own Software Development Kit (SDK) which includes a smaller standard library and SGX specific functions for creating, entering and exiting enclaves. The developer writes an enclave definition file where the interface is defined and the SGX compiler adds the required low level instructions which link both components.

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

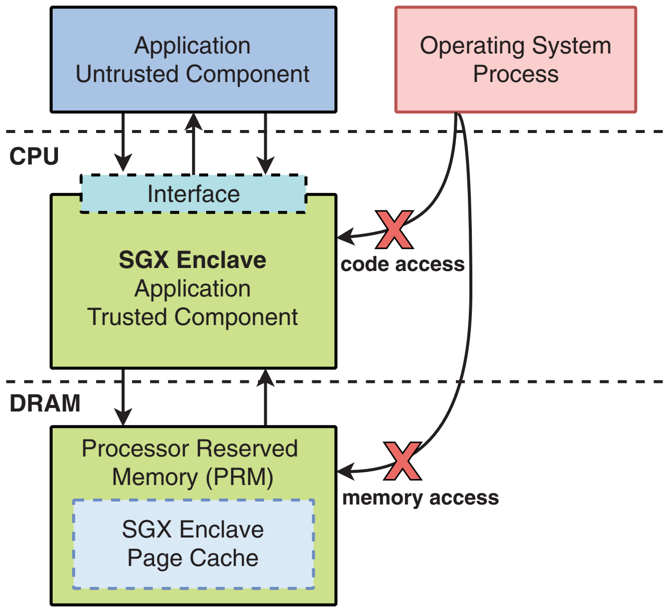


Fig. 1. Software partitioning in SGX.

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.

For data persistence an enclave can seal its data using an SDK sealing function that uses the CPU hardware key in combination with the enclave's hash to encrypt data and store it in the filesystem which can later be unsealed and used to set enclave state. In order to avoid being reset with an old copy of data, designated as a rollback attack, SGX uses monotonic counters. These counters are created on non-volatile memory (NVM) for each enclave performing sealing operations and are incremented every time data from an enclave is sealed. The monotonic counter is also embedded in the sealed data. To prevent an attacker from replaying data, SGX reads the monotonic counter value from the NVM and compares it with the data's monotonic counter when sealing thereby confirming if the data is the latest version or not.

SGX attestation is a mechanism that allows the code running on an enclave to verify some characteristics of another enclave: identity and version of the code running on the other enclave, and whether the other enclave is running on a legitimate SGX platform (and not on a virtualized environment). Depending on the location of the both enclaves (in the same or on different CPU), two attestation mechanisms are available: Local attestation and Remote attestation.

2.1.1. Local attestation

This type of attestation (Intel, 2019; McKeen et al., 2013) allows a local enclave to prove to another local enclave it is running on real SGX hardware and provide its code signature (*MRENCLAVE*).

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.

It's envisioned as a way for two applications to establish trust between each other but is only possible between enclaves running on the same CPU as it relies on the enclaves deriving the same value using a hardware dependent key.

Along with this confirmation, the source-enclave can send 512 bits of additional data (report-data) to claim knowledge of a determined secret.

2.1.2. Remote attestation

Remote attestation (Anati et al., 2013; Johnson et al., 2016) is used for attestation between one enclave and another enclave running on a different CPU, allowing two applications running on separate hosts to establish trust. This process requires a third entity called Quoting Enclave, provided by Intel, that needs one instance running on the host of the local enclave and another running in the host of the remote 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. Later iterations of SGX have introduced Intel SGX Data Center Attestation Primitives (SGX DCAP) (Zimmerman, 2019) which allow service providers to perform this function of trusted third-party by caching the required certificates and TCB info for each Intel SGX enabled platform in the data center.

2.1.3. Vulnerabilities

There have been concerns about possible man in the middle attacks (Lee et al., 2017; Swami, 2017) to the remote attestation mechanism and vulnerability to micro-architecture flaws such as Zombieload (Schwarz et al., 2019).

Swami (2017) theorizes on why SGX's Group Signing scheme has weak privacy guarantees since the provisioning ID (PPID) is linked to each individual CPU. This, along with Remote attestation's validation being reliant on Intel services during runtime, places Intel not only as a key trust element during manufacture but as well during each Remote attestation procedure.

Lee et al. (2017) focuses on exploiting Enclave Enters (*ENCLU*) and Exits (*EEXIT*) to find buffer overflow vulnerabilities by looking for *pop* operations in the page faults of enclaves Using this information the authors create a malware enclave that becomes a Man-in-the-middle for the target enclave. This new enclave effectively hijacks SGX remote attestation and is able to leaking the encryption key used for data sealing. The attacker can then use this encryption key and exit point to unseal an enclave's information.

Zombieload (Schwarz et al., 2019) 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. The author demonstrate that by compromising some SGX primitives, the attacker can use them to undermine Intel's architectural quoting enclaves and thereby breaking remote attestation.

2.2. 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. (2016) 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. (2017) provides a first implementation on OpenSGX (Jain et al., 2016), an open platform for SGX research, followed by Park et al. (2019) 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. (2018) builds on the communication mechanisms of Park et al. (2016) 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.

Alder et al. (2018) solves the issues with enclave migration by using a SGX-only approach 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.

In order for a developer to implement the system, a library is provided to be included in the application enclave to handle packing, unpacking and communication establishment with the migration enclave. A migration enclave must also be deployed in the source and destination hosts.

However, these solutions (Alder et al., 2018; Gu et al., 2017; Park et al., 2019) 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.

ReplicaTee also offers a data storage layer (BSL), that can be used for the sharing of data between replicas of the same enclave. BSL uses a Bizantine Fault Tolerant algorithm that, for small data, offers good throughput. It the BSL is used for complete enclave migration (with data in the order of tens or hundreds of MB), the delays will become higher due to all the necessary message exchanges and due to all the cryptographic operations inside the enclave needed to implement access control and privacy in BSL.

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

2.3. SGX in cloud environments

More service providers have started offering SGX services for their clients as the technology matures. Big providers such as IBM (IBM, 2019) and Azure (Microsoft, 2019a) already offer dedicated IaaS with SGX integrated while other providers such as Google Cloud investigate vendor agnostic TEEs such as Asylo (Google, 2019) for their offerings.

2.4. 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

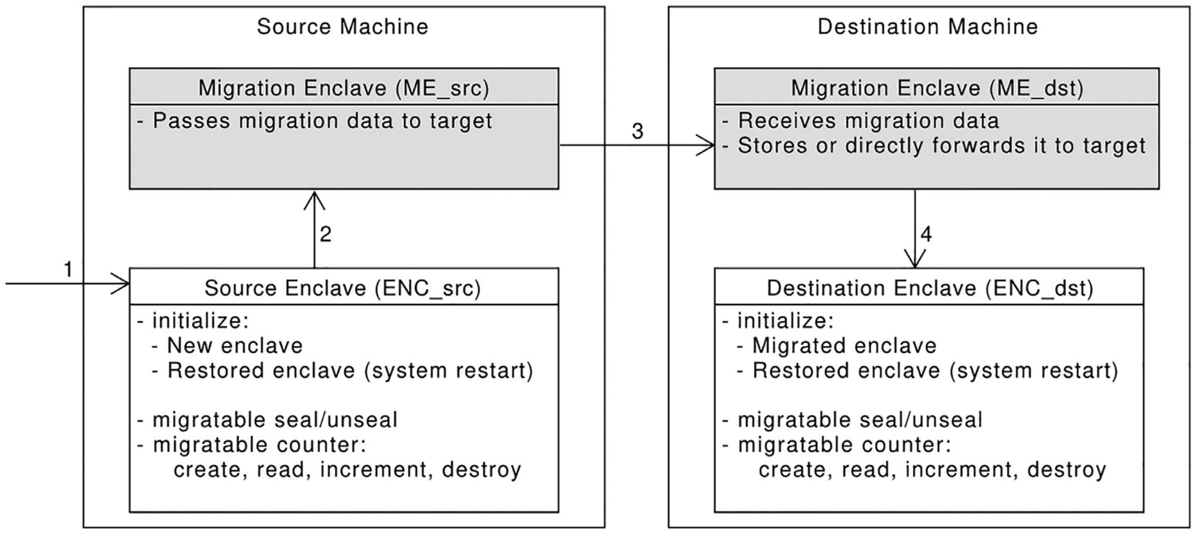

Source Machine
Migration Enclave (ME_src)
- Passes migration data to target
1
2
3
4
Source Enclave (ENC_src)
- initialize:
- New enclave
- Restored enclave (system restart)
- migratable seal/unseal
- migratable counter:
create, read, increment, destroy
Destination Machine
Migration Enclave (ME_dst)
- Receives migration data
- Stores or directly forwards it to target
Destination Enclave (ENC_dst)
- initialize:
- Migrated enclave
- Restored enclave (system restart)
- migratable seal/unseal
- migratable counter:
create, read, increment, destroy

Fig. 2. Alder et al. (2018) solution architecture. © [2018] IEEE. Reprinted, with permission, from Alder et al., 2018.

(National Institute of Standards and Technology, 2019) and Common Criteria (ISO Central Secretary, 2014).

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.

They are largely used in the payment card industry, the banking industry and Certificate Authorities (CAs) as a requirement of the industries' standards but have found their way into the cloud environment, industrial IOT and blockchain applications in newer hardware revisions as privacy and data treatment concerns spread to other industries. These appliances also allow logs of accesses, key usages and other operations to be kept enabling auditing which furthers their usefulness in regulated industries.

In the following sections we focus on HSM bootstrapping, authentication, message transport and deployment using Safenet HSMs.

2.4.1. *HSM Operation*

HSM bootstrap 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.

HSMs typically support two authentication methods: Password Authentication and PED (Trusted Path) authentication. In password authentication, a PIN is used to access HSM functions while in PED (Trusted Path) authentication, after this PIN is presented, access has to be allowed by the Security Officer in the HSM administration tools with a hardware specific key.

2.4.2. *Network trust links (NTL)*

NTL (AWS, 2019c) is a communication protocol specific to Safenet HSMs. 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.

NTL is envisioned to be used to deploy HSM services in cloud environments, or in situations where message integrity is paramount.

2.4.3. *HSMs in cloud environments*

A variety of cloud service providers now offer HSM services, mostly in two main classes: Dedicated HSMs where the clients retain complete administrative and cryptographic control of the HSM, meaning the Security Officer is appointed by the client and PIN distribution is handled by the client, and HSM-as-a-Service where the service provider handles provisioning, hardware firmware maintenance and HSM partitioning allowing the client to administer a partition but not the hardware.

Service providers such as Azure and AWS offer both HSM-as-a-Service (AWS, 2019b; Microsoft, 2019c) and Dedicated HSM services (AWS, 2019a; Microsoft, 2019b).

3. TEEnder

In this section we present TEEnder, a framework that implements enclave data migration and solves the current limitations 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 datacenters that implement the Infrastructure as a Service (IaaS) model (services deployed in Virtual machines), need VM migration (duet to resource optimization or fault tolerance), whose applications use the SGX services and that already have as HSM infrastructure working.

In datacenters, some of the most important factors and quality of service measures are service performance, uptime and power consumption. Virtual machine live migration (Clark et al., 2005) 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 (Ahmad et al., 2015).

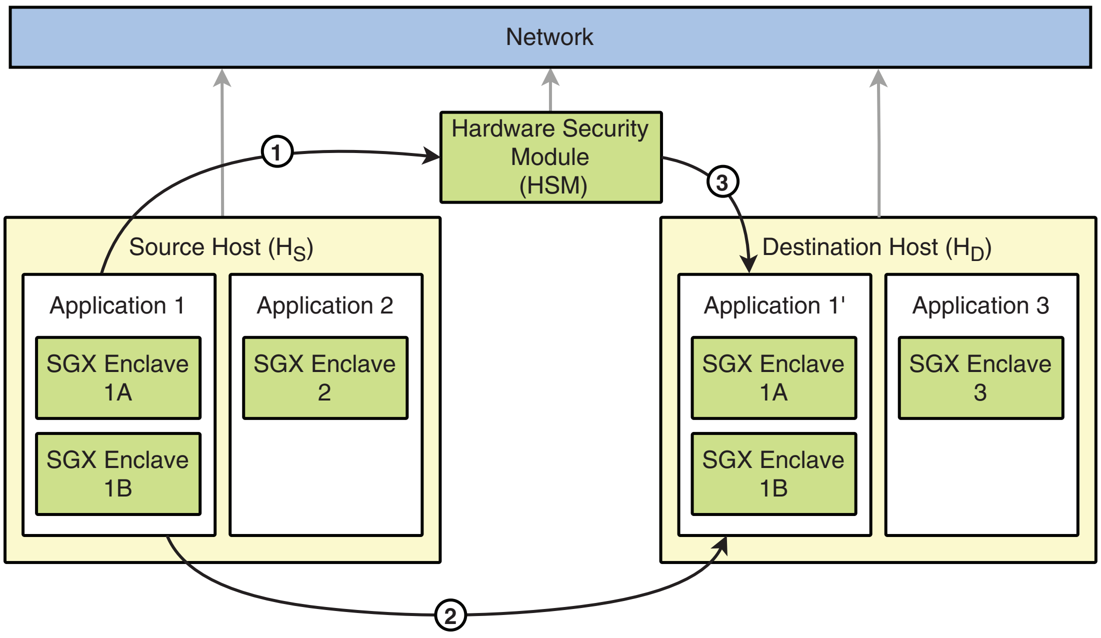


Fig. 3. Environment architecture.

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. Fig. 3 shows the envisioned environment for the deployment of the proposed solution.

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 (McKeen et al., 2013) 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.

The privileges this attacker can accumulate are enumerated by increasing degree of threat as:

1. Network access
2. Access to a Virtual machine on the same host as the application.
3. Access to the hypervisor.
4. Root access to the application OS.
5. Physical access to the machine.

SGX does not provide availability guarantees (Jang et al., 2017), as expected of a threat model where the attacker has physical access, therefore we focus mostly on data integrity and confidentiality.

For the purposes of our solution 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. Objectives

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_S1 The migration system must not add new attack vectors to the host machine.
R_S2 The system provide end to end protection of the migrated data.
R_S3 Sealed migration data must only be unsealed by an authorized machine.
R_S4 Data must not be migrated to rouge machines outside the datacenter controlled by the attacker.
R_S5 An audit log must be kept for the system to keep track of sealing and unsealing of data.
R_S6 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_P1 The system must handle large enclaves without serious loss of performance.
R_P2 The system must do concurrent sealing of data without serious loss of performance.

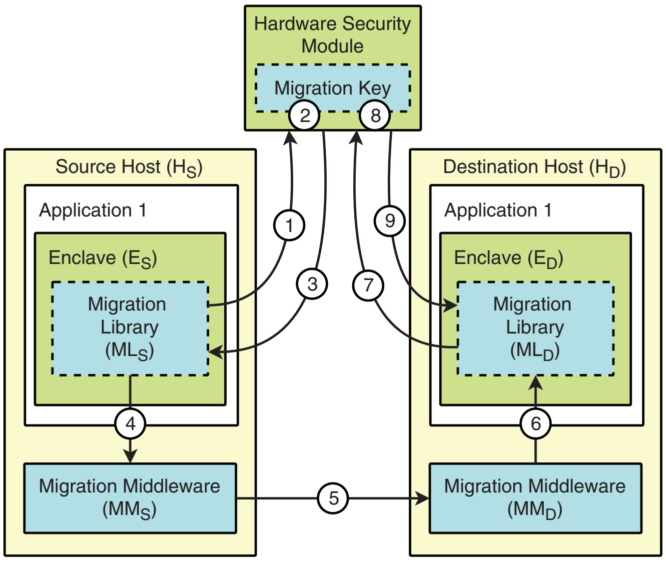


Fig. 4. Design architecture.

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. (2018) and Park et al. (2019) 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.

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

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 Fig. 5, modifications are only done inside of the applications enclave. The enclave must have the HSM pin, HSM certificate and HSM address 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:

One Time key Based on the same principle as Alder et al. (2018)'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.

In case the migration data is leaked over the network, an attacker does not have access to the key necessary to decrypt it and only a vector bound to that single migration process.

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. Initial HSM deployment is done as presented in the first section of Section 2.4.1 with the Security Officer setting up the HSM with a partition for use in the migration procedure and generating 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 (Intel, 2018) or through another method using remote attestation.

3.7. Implementation

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

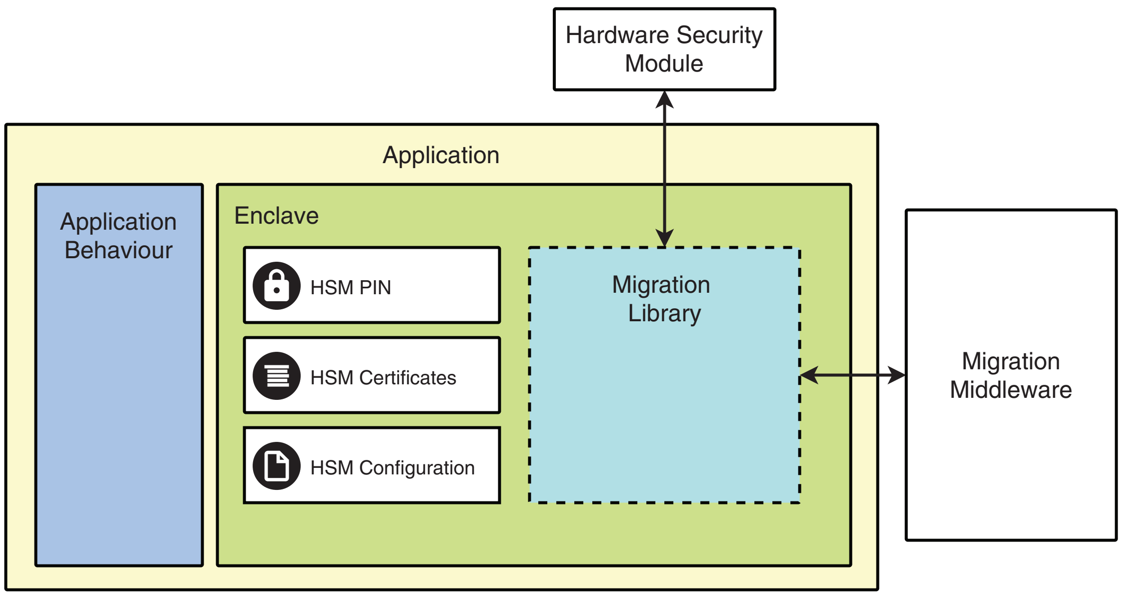
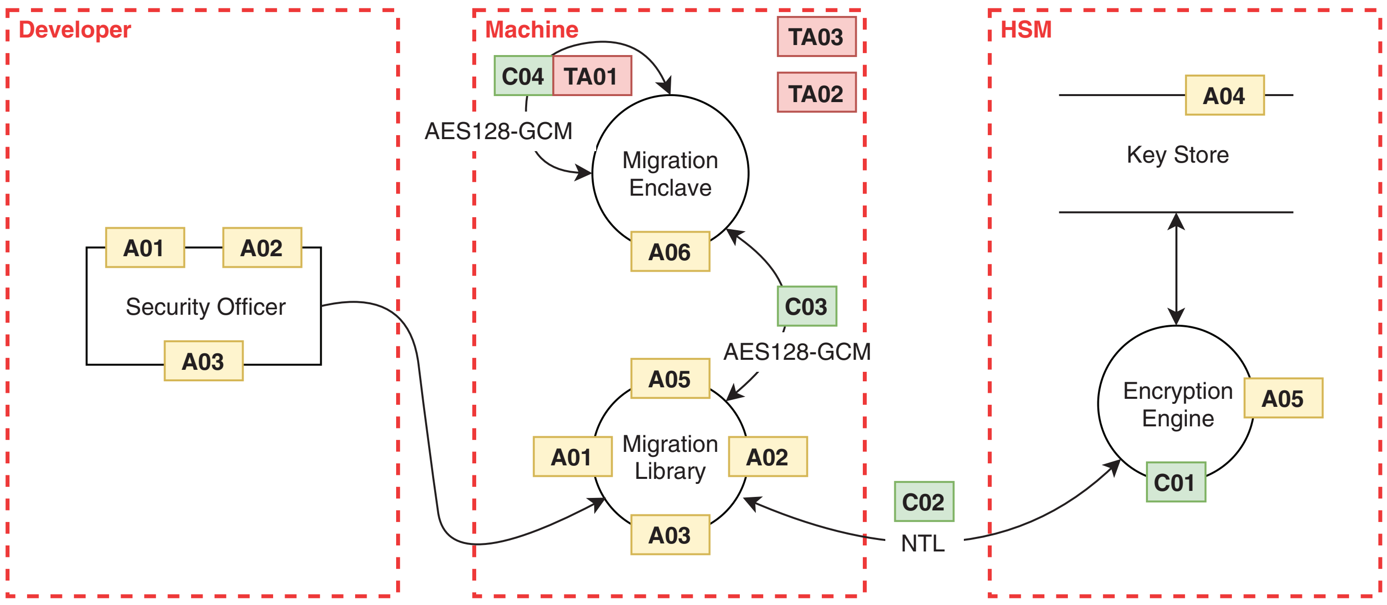


Fig. 5. Application architecture.



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.

The implementation should be interfaceable with any appliance that implements the *PKCS*#11 interface specification (OASIS, 2015) for communication as HSMs do not deviate from *PKCS*#11. We only consider the usage of Network HSMs since PCI HSMs trust the PCI lanes to be secure, an assumption excluded by the threat model (Section 3.2), and we exclude token HSMs as they lack the required performance to make the system feasible.

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 (C02,2.4.2), using pre-provisioned certificates. Authentication for the HSM functions is done through an application specific PIN (C01) explained in Section 2.4.1.

3.7.1. Migration process

Finally as a combination of both Figs. 4 and 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.

Unfilled arrows ($\rightarrow$) represent procedures and filled arrows (►) represent data movement.

The application's migrateable data is sent from ML_S to the HSM through a NTL channel (2.4.2) 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 or a label matching key as presented in Section 3.5. 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.

3.7.2. Sealed data structure

Fig. 8 shows the differences between the original SGX sealed data structure and the TEEnder structure with size presented in bytes. The variables x and y are calculated by the application in the beginning of the migration process. TEEnder reuses the *sgx_sealed_data*$_t$ defined by SGX for better compatibility but uses the variable size payload to add extra information it needs, this includes the IV and data tag. The default SGX structure already has memory allocated for the data tag but as the HSM takes in a buffer with the encrypted data and the tag immediately following it, the tag is kept redundantly in the payload.

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. Proof of concept

The implemented application layout is presented in Fig. 9 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.

The migration library, to be included with the application, has 1393 LOC (lines of code) versus the 808 of the library of Alder et al. (2018), representing a difference of 585 LOC, an increase of 72.4%. The only external dependency of the migration library is the *PKCS*#11 library required to communicate with the HSM.

With TEEnder the TCB will also include the HSM infrastructure, making it much larger than the current migration infrastructure TCB. Nonetheless for businesses that already depend on HSMs,it is already considered a TCB in their applications. From our point of view, to the applications that already use HSMs but now require the migration of enclaves the use of HSM will come with no increase of the TCB size.

The implementation mostly modifies the migration library's data sealing path but keeps a similar structure for compatibility reasons. Fig. 10 shows the difference between the SGX sealing path and the prototype's. These sealing functions can be called independently to cipher other data using the HSM.

A *ecall_hsm_get_required_size* calculates the size needed for the *sgx_sealed_data_t* structure (8). In the prototype this function also allocates a random buffer of characters of a given size to serve as test data.
B *ecall_hsm_seal* is called to allocate the structure.
C *ecall_seal_migrateable_data* opens a *PKCS*#11 session and populates the structure with the result of the calls to the HSM.

We consider the elapsed time in **B** and **C** as the time required for sealing data. The process for unsealing follows the same architecture.

4.2. 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 (Zhao et al., 2016; Zheng et al., 2017).

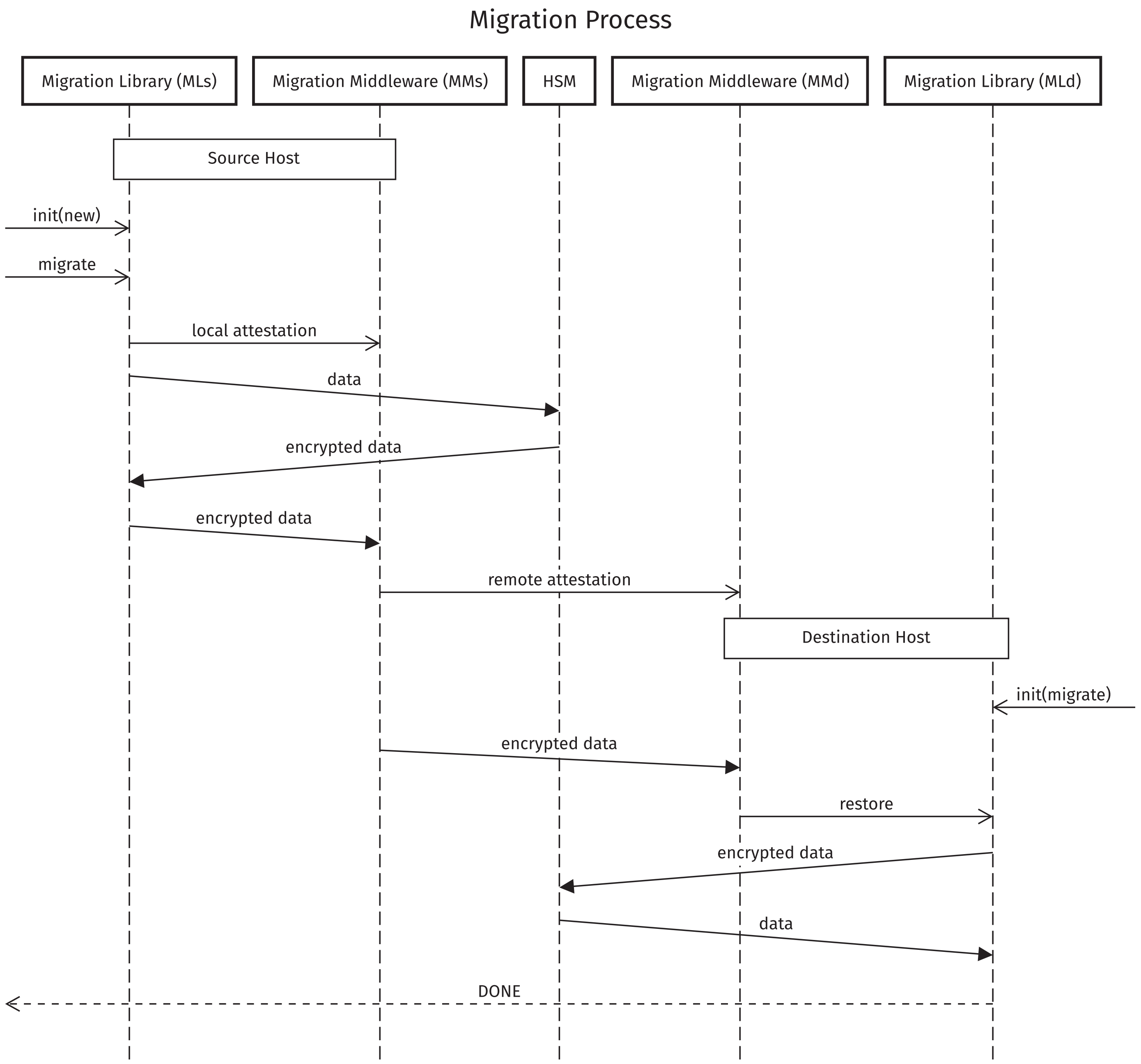


Fig. 7. Protocol diagram.

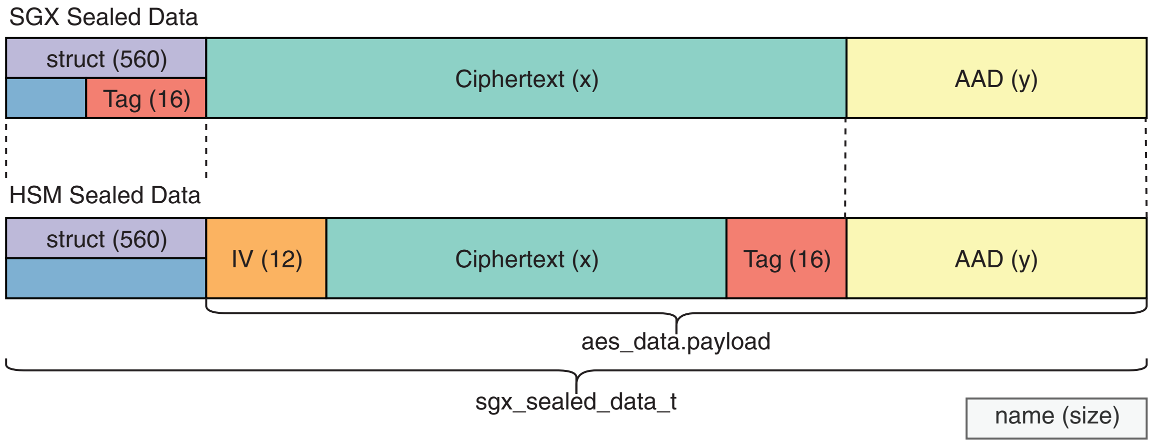


Fig. 8. Sealed data structure comparison between SGX and TEEnder.

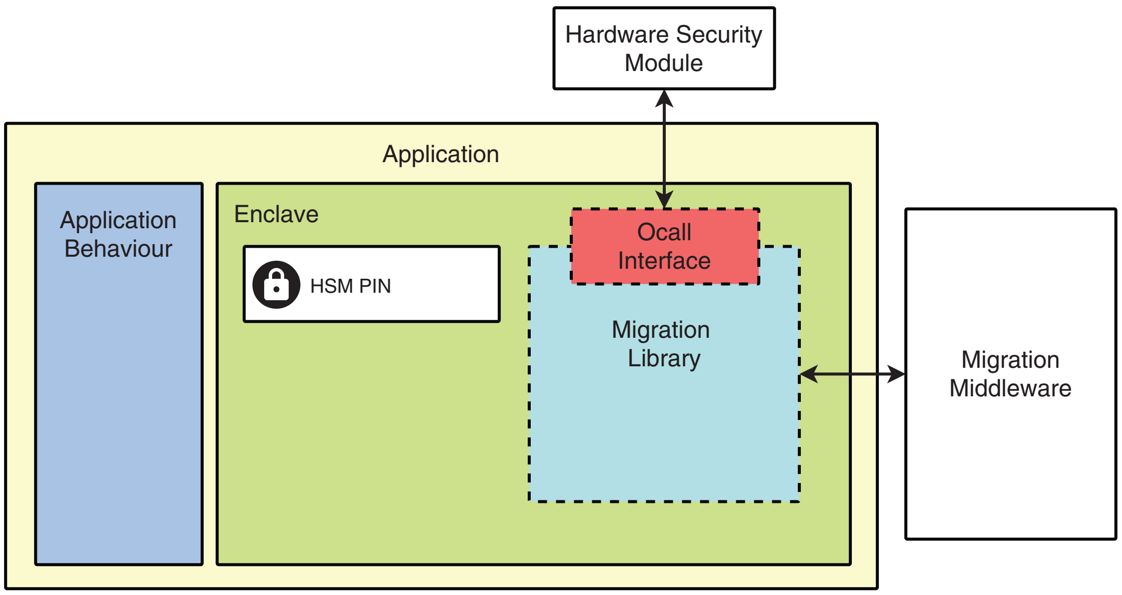


Fig. 9. Implemented application layout.

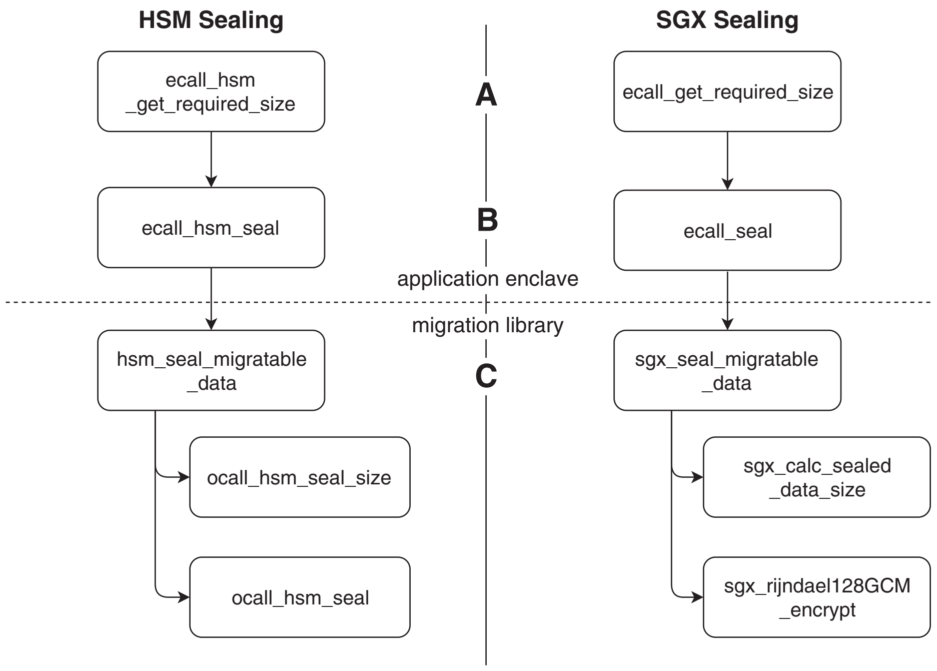


Fig. 10. SGX and TEEnder migrateable data sealing path.

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 research average enclave data size to validate our choice of data size for benchmarking in Section 4.2.2

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.

4.2.1. *Evaluation environment*

In this section we introduce the evaluation environment used for evaluating TEEnder performance.

Fig. 11 shows the networking setup of TEEnder used for testing where a mock migration is conducted with one host serving as both host and destination. The value measured is the elapsed time in steps **B** and **C** presented in Fig. 10 which include the round trip to the HSM for data encryption. On the SGX solution, the same steps in their SGX equivalent implementation are measured but the round trip time (RTT) is 0 in this case.

Table 1 presents the various environments used for the performance evaluation following the setup on Fig. 11.

4.2.2. *Average enclave size*

In order to better understand the size requirements for enclave migration using an HSM or SGX, there is a need to predict the size of the migrateable data. The total enclave size is determined by the sum of the stack size, heap size and application code size of the enclave. The most relevant of these is heap size as it does not have a fixed size and is generally the biggest. The max heap size

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

Table 2
Enclave sizes collected from various enclave projects implementations.

	CFHider (Wang et al., 2019)	Zero Trace (Sasy et al., 2018)	Slalom (Tramèr and Boneh, 2018)	SCBR (Pires et al., 2017)	Tensor Scone (Kunkel et al., 2019)	Safe Keeper (Krawiecka et al., 2018)	Lightbox (Duan et al., 2017)	Clemmys (Trach et al., 2019)	STANlite (Sartakov et al., 2018)	TaLoS (Pierre-Louis Aublin et al., 2017)
Max Heap Size (MB)	2100	570	520	11–56	330	110	214	128–4000	100–500	73

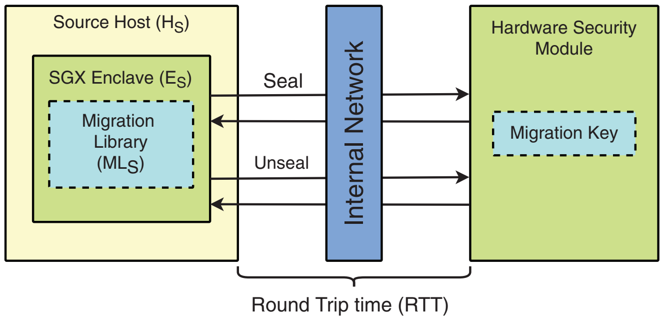


Fig. 11. Proof of concept evaluation environment setup.

can be defined in Intel SGX through the enclave configuration and as such is a good measure of the expected max size of the enclave.

The collected data in Table 2 shows that the max heap size of an enclave has a very wide range and specially for machine learning applications tends to grow beyond the Enclave Page Cache size of 128MB (Intel, 2019) meaning the solution has to be able to cope with larger data sizes.

Based on this data we conclude that testing data size between 1MB and 1GB should account for most development scenarios.

4.2.3. *Increasing data size*

In Fig. 12, 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.

In Fig. 12, 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.

4.2.4. *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.

Fig. 13 presents the results of the tests for $n = 1, 4, 7, 10$ concurrent enclave migrations, with the sealing and unsealing done inside the processor or on a HSM.

At $n = 4$ processes in Fig. 13, *sgx_p* becomes slower than *hsm_m* at $s = 41MB$ while *sgx_s* becomes slower than *hsm_m* at a data size of $s = 30MB$.

When only one process is sealing or unsealing data, the SGX times are lower that those from using HSM, this is due to the fact that to perform encryption of the HSM data needs to be transferred through a network link. This data transfer overhead voids any gain from the higher encryption speed of the HSM.

With several concurrent encryption/decryption process, the results are different. For small data, the gains from the high HSM speed are not enough to overcome the network data transfer overheads, but after a certain data size (around 20/30MB) the use of

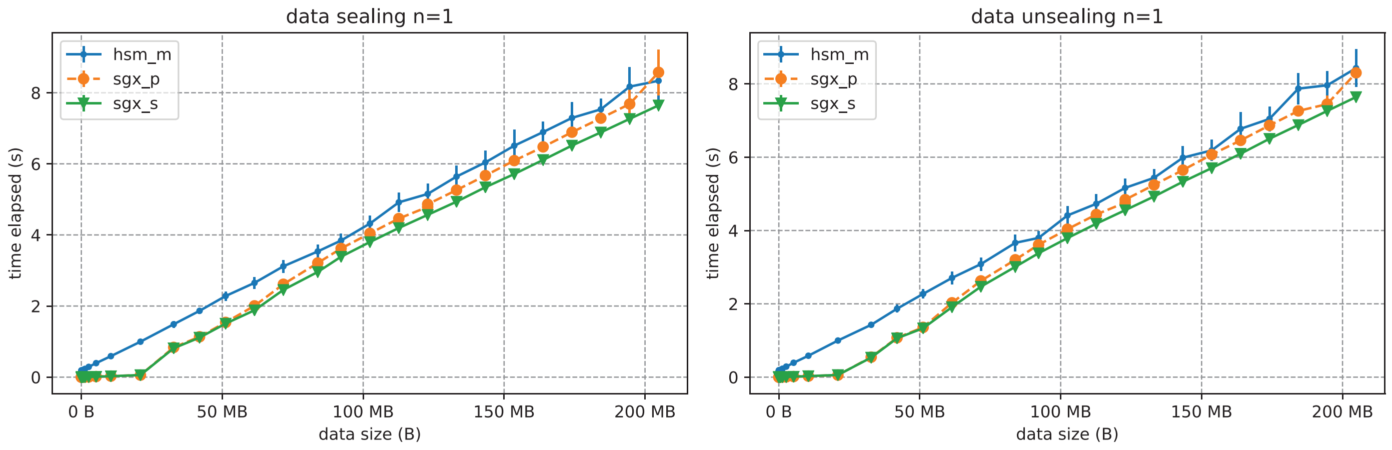


Fig. 12. Effect of increasing data size on SGX and HSM data sealing.

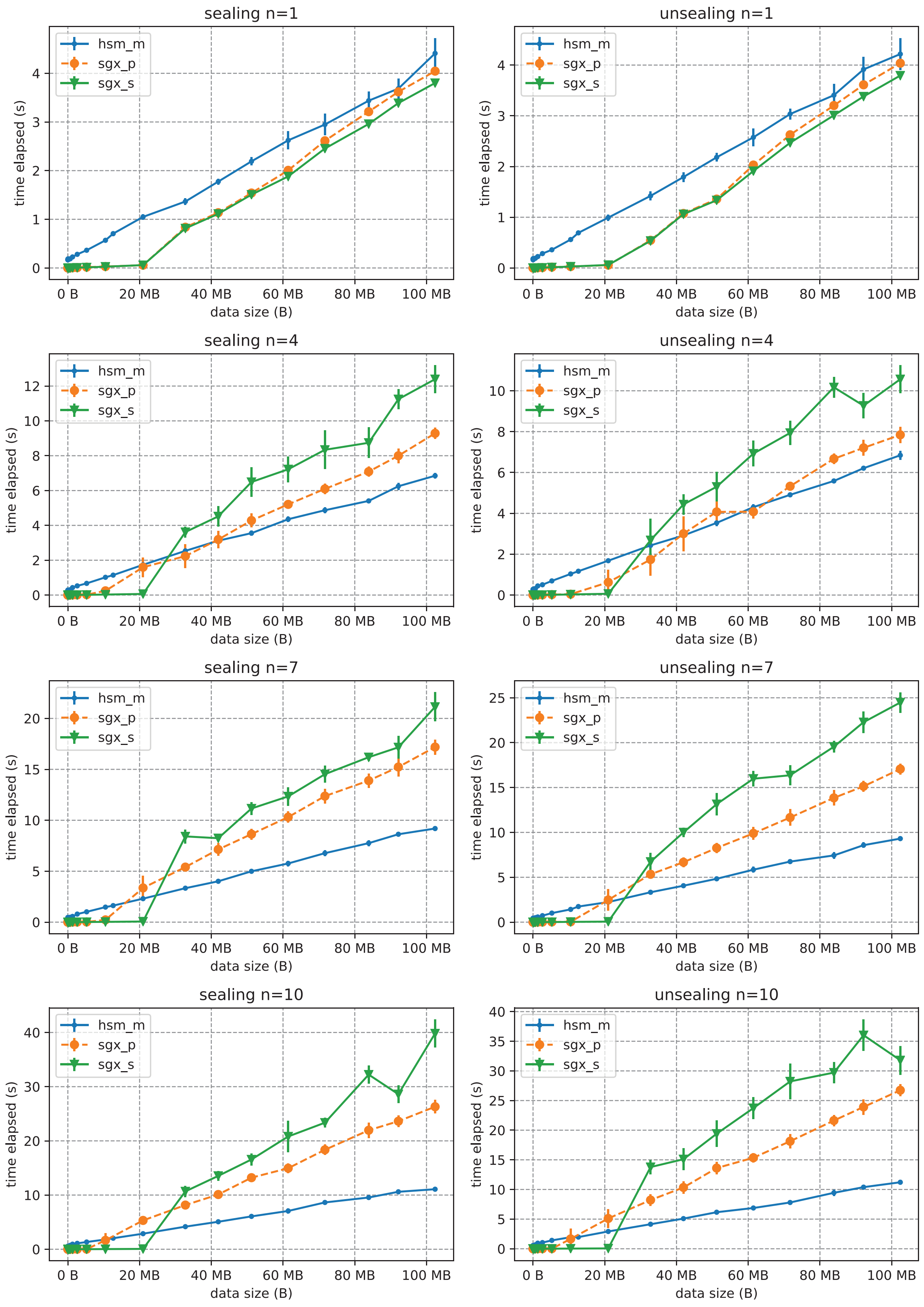


Fig. 13. Plot of data sealing time in relation to increasing number of processes.

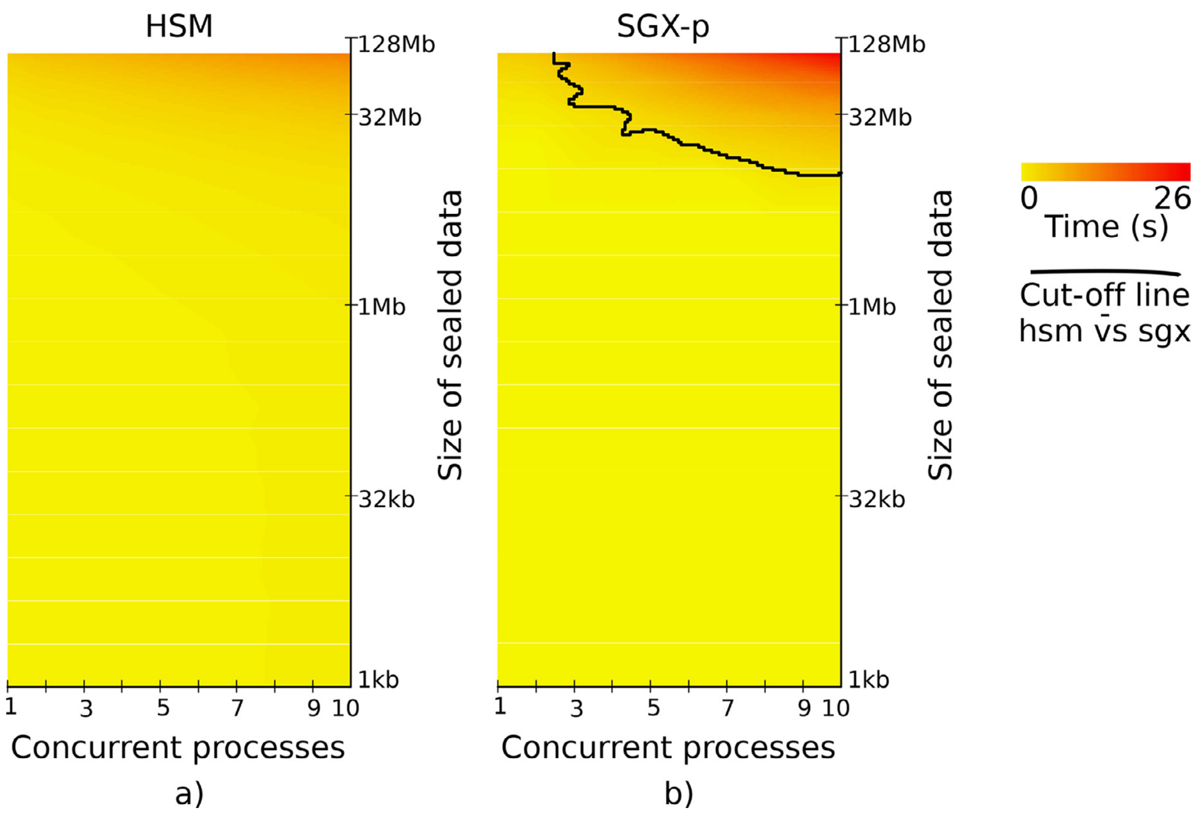


Fig. 14. Surface map of sgx and hsm data sealing times.

HSM is advantageous over the encryption/decryption locally inside the enclave.

This switch in execution times is due to the fact that for multiple concurrent processes the local encryption/decryption becomes slower at an higher rate that the same operations done on the HSM. As expected the HSM times for each test become higher (for about 100MB these time are 4, 6, 10 and 11 seconds), while for the sgx the same operation increases at a higher rate (4, 10, 20 and 30 seconds).

For low data sizes (a usage of 10-20MB per process), we theorize that SGX is within it's ddr cache (unencrypted but inaccessible by other processes) limit of 128MB and encryption/decription runs at maxumum speed. With a larger workload the system starts swapping to userspace the encrypted RAM, adding latency for encryption and decryption of memory pages. This affects SGX performance exponentially as more of the system reserved memory (unencrypted) is used to store page indexes.

This possibility is also consistent with the observation of the encrypt/decrypt performance on the Xeon E3-1230 v5 using the AES-NI instructions (Xu, 2010). For reference, in this machine and using Openssl it is possible to attain aes-128-gcm encryption speeds of about 5GB/s (as opposed to 280MB/s without AES-NI), leading to encryption of 10 or 100 Mb in the order of the milliseconds.

The graphics shows that the times to process data smaller than 10/20MB is close to zero and is compatible with the high performance of the Intel AES-NI instruction sets. When the size of data increases and we start to have concurrent encryption, the effects described earlier (such as cache and page management) start affect the performance.

To better illustrate the effect of concurrent processes and data size, Fig. 14 was produced from the linear interpolation of the average elapsed time until conclusion of data sealing.

For small data sizes and a low number of concurrent sealing processes both alternatives show approximate results. With the increase of the size of the data and number of concurrent processes, the SGX approach degrades at a higher pace.

The black line presented in Fig. 14 *b*) represents the points where the sealing/unsealing of data done using HSM or SGX is the same. Inside of the area defined by this curve (top right corner) it is better to use HSM enabled TEEnder (*HSM* in Fig. 14 *a*)) than an SGX-only solution. This intersection point is at about 100MB when $n = 2$, 30MB when $n = 4$ and closer to 20MB when $n = 10$.

This behavior seems to be consistent with the analysis of SGX performance degradation related to the cache use. For small data sizes we can point out network latency as the cause of the HSM's lower performance, while with multiple processes executing concurrent encryption/decryption inside enclaves, the contention on local resources and cache behavior makes the HSM solution better even for small data.

4.2.5. HSM Latency

Fig. 15 presents the relation between latency and data sealing time using TEEnder. The results were taken while executing 10 concurrent sealing/unsealing and with 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 of random data, was used for each value of latency to construct the plot.

The horizontal lines presented in Fig. 15 represent the time to seal/unseal the data (10MB or 100MB depending on the graph) locally, while the vertical lines represent the average network latency between two nodes 300Km and 500Km apart

We can then predict that a 100MB enclave will take around 150s to seal in a HSM at a distance of 300km while a 10MB En-

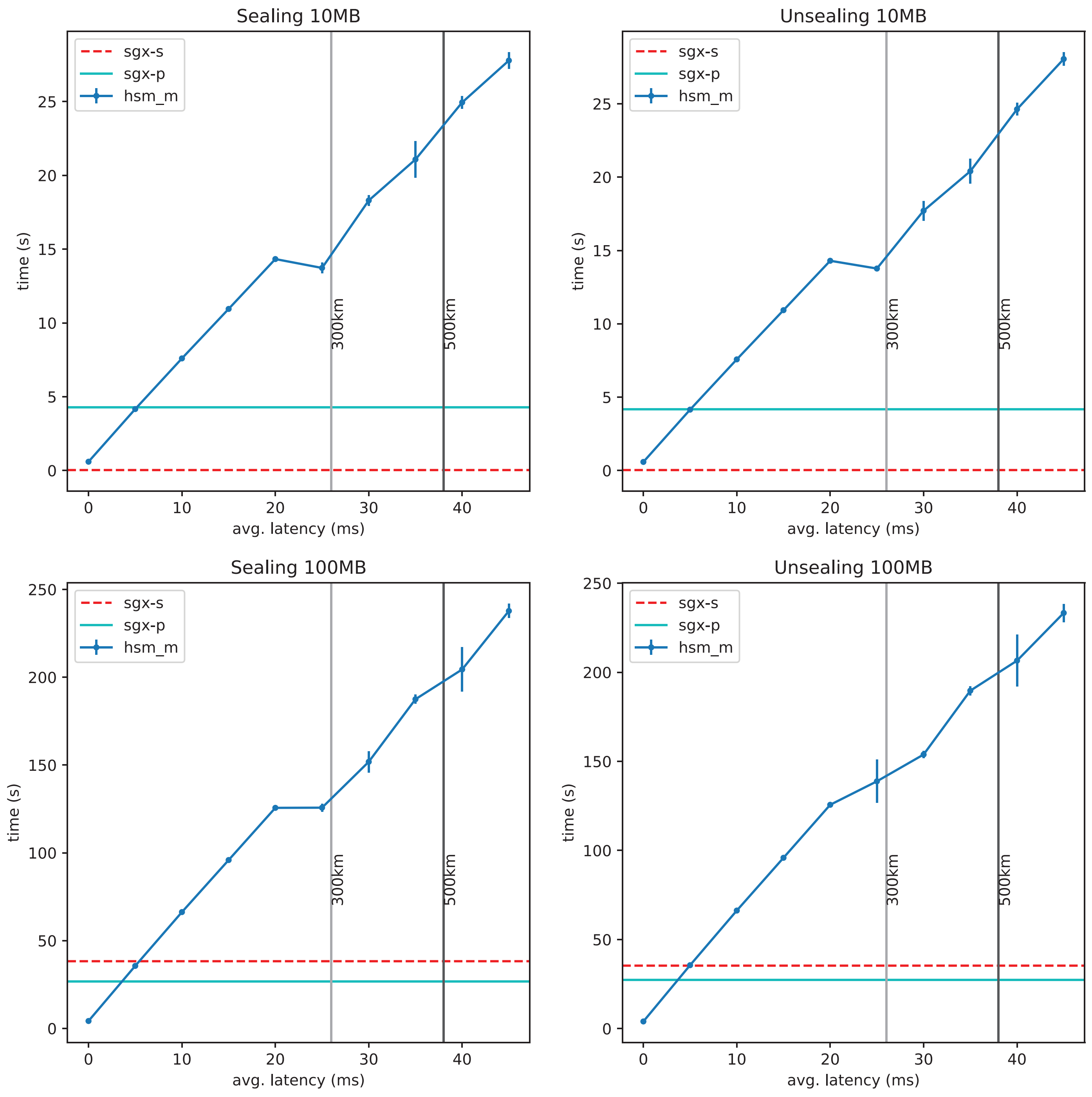


Fig. 15. HSM latency effects.

clave will only take around 15s. When compared with the local sealing the decrease in performance is about 5 times. This penalty comes from the data transmission latency.

From these observations, we conclude that an increase in latency between the host and the HSM will significantly move the cut-off line presented in Fig. 14 up in the graph. We therefore consider latency a strong variable in determining the performance of TEEnder. Nonetheless the possibility to use remote HSM with a still acceptable delays opens possibilities for other uses, such as migration to remote data-centers.

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 [R_S2] as plain-text data is only available to the HSM and the enclave but is encrypted in transport.

For [R_S3] and [R_S4], 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 [R_S5] 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, [R_S6] 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.

TEEnder reduces the effects of the previously described attacks (Schwarz et al., 2019; Swami, 2017). If the remote attestation is compromised, it is true that the link between the migration middleware instances becomes unsafe and the encrypted transferred data can be leaked but the attacker will not have access to the original information. In this case the sealing key used to encrypt the data is not transferred between enclaves, since the one used is inside the HSM.

The fact that encryption keys are only stored in HSM makes TEEnder's attack surface on the migration middleware severely reduced compared to previous work, since the migration middleware never manipulates the decrypted data nor transfers the encryption key.

TEEnder is not vulnerable to attacks on remote attestation during the migration process, but in a similar way is vulnerable during the initial deployment. From our point of view, since the migration is a more frequent process, and the system initialization (HW instalation, OS instalation, HSM configuration and SGX initialization) is to be done on a more controlled environment, the initialization does not add much exposure.

It should be possible to apply denial-of-service attacks on the HSM enabled migration system by restricting packets between the enclave and the HSM, but that is not in scope of the work as SGX does not provide availability guarantees and this same type of attack can be carried out in other primitives of SGX, such as local attestation or even resource availability.

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

In the first place, 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.

In the second place, the SGX enclave environment relies on statically compiled binaries with no external dependencies in order to provide complete isolation. The existence of external dependencies involves the usage of *OCALLS* (Intel, 2019) which exit the enclave in order to use the system resources and libraries. The normal process for running HSM software is to dynamically link the PKCS#11 library and dynamically load its symbols as shown in the Amazon Web Services CloudHSM samples breaking SGX's security guarantees.

Therefore we evaluate requirements [R_S1] positively on a design aspect but lacking in the implementation phase. We evaluate this requirements as accomplished but not production ready.

For these issues two possible alternatives were identified: use of containerization or patching of vendor changes.

5.1.1. Containerization

SCONE (Arnautov et al., 2016) allows a developer to do minimal alterations to his program and run them securely in SGX enclaves by using its own dynamic link loader to replace the standard C library with its own guarded library. This process would replace the insecure bindings in the PKCS#11 library and allow dynamic symbol loading in a safe space. Furthermore SCONE's container filesystem (SCONE fileshield) allows the PKCS#11 library to transparently load encrypted certificates and HSM configurations without any changes by the vendor.

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 (Pierre-Louis Aublin et al., 2017) 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 [R_P1] and [R_P2] achieved as an analysis of Section 4.2.3 and Section 4.2.4 respectively, but we also recognize the potential for improving TEEnder's sealing times by reducing the number of HSM sessions from 2 to 1. *ocall_hsm_seal_size* and *ocall_hsm_seal* represented in Fig. 10 both establish separate HSM sessions with their own key negotiation based on the provisioned certificates. This was done for implementation simplicity but has a measurable effect on performance.

Latency was controlled for the execution of tests but as Section 4.2.5 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.

5.3. Feasibility evaluation

In terms of feasibility, we acknowledge the increased importance of the HSM Security Officer and their workload in this scheme for ensuring a secure bootstrap, managing migration permissions and preparing client and server certificates.

As for developers, we don't identify increased obligations or usability issues comparing with previous work. Initiating the migration process and Migration Library requires a single call and HSM encryption functions share most of the variable inputs as their SGX sealing counterparts.

We therefore argue that in settings where HSM infrastructure is already set up, such as cloudHSM sites, the usage of our migration system is a low effort task to implement and for Security Officers does not differ from treating any other HSM enabled application.

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 significant. 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.

We also believe the higher degree of control and security offered by TEEnder justifies its bigger TCB footprint. This large TCB is due to the need to integrate the SGX infrastructure with the HSM one, but, due to the security guarantees offered by the HSM, its libraries, management procedures, and providers, we believe that the vulnerabilities will be reduced.

The co-existence of SGX, Virtualization and HSM is a trend that is now gaining popularity: as the cloud providers are starting to offer both SGX and HSM as a service, but also as other industries that have been using HSM to implement security requirements and are now adopting virtualization and the use of SGX. To the second users, the adoption of the proposed solution comes naturally since the use of HSM is already considered secure and our solution solves a vulnerability existent in the SGX infrastructure. To the cloud providers, the adoption of HSM allows a faster migration of enclaves (and reduction of service downtime) in high load and contention infrastructures.

This work also 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.

Funding

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UIDB/50021/2020.

Declaration of Competing interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

João Guerreiro: Conceptualization, Methodology, Software, Investigation, Writing - original draft. **Rui Moura:** Conceptualization, Methodology, Validation, Investigation, Writing - review & editing, Supervision. **João Nuno Silva:** Methodology, Validation, Writing - review & editing.

Acknowledgements

We would like to thank the availability and support of Multicert for this project in mentoring and providing necessary SGX hardware, as well as Raul Suarez of Thales Group for providing deeper insights into the working of the Luna SA HSMs and access to state of the art appliances that made this work possible.

References

Ahmad, R.W., Gani, A., Hamid, S.H.A., Shiraz, M., Yousafzai, A., Xia, F., 2015. A survey on virtual machine migration and server consolidation frameworks for cloud data centers. J. Netw. Comput. Appl. 52, 11–25. doi:10.1016/j.jnca.2015.02.002.

Alder, F., Kurnikov, A., Paverd, A., Asokan, N., 2018. Migrating SGX Enclaves with Persistent State. In: 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). IEEE, Luxembourg City, Luxembourg, pp. 195–206. doi:10.1109/DSN.2018.00031.

Anati, I., Gueron, S., Johnson, S., Scarlata, V., 2013. Innovative technology for CPU based attestation and sealing. In: HASP - International Workshop on Hardware and Architectural Support for Security and Privacy, pp. 1–7.

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., Fetzer, C., 2016. SCONE: secure linux containers with intel SGX. In: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16). USENIX Association, pp. 689–703.

AWS, A., 2019a. AWS CloudHSM. https://aws.amazon.com/cloudhsm/.

AWS, A., 2019b. Key Management Service - Amazon Web Services (AWS). https://aws.amazon.com/kms/.

AWS, A., 2019c. 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.

Clark, C., Fraser, K., Hand, S., Hansen, J.G., Jul, E., Limpach, C., Pratt, I., Warfield, A., 2005. Live migration of virtual machines. In: Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2. USENIX Association, pp. 273–286.

Costan, V., Devadas, S., 2016. Intel SGX Explainedhttps://eprint.iacr.org/2016/086.

Duan, H., Wang, C., Yuan, X., Zhou, Y., Wang, Q., Ren, K., 2017. LightBox: full-stack protected stateful middlebox at lightning speed. arXiv:1706.06261 [cs].

Gjerdrum, A.T., Pettersen, R., Johansen, H.D., 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. SCITEPRESS - Science and Technology Publications, Porto, Portugal, pp. 696–703. doi:10.5220/0006373706960703.

Google, 2019. Introducing Asylo: An open-source framework for confidential computing. https://cloud.google.com/blog/products/gcp/introducing-asylo-an-open-source-framework-for-confidential-computing/. (Accessed: 2019-08-05).

Gu, J., Hua, Z., Xia, Y., Chen, H., Zang, B., Guan, H., 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). IEEE, Denver, CO, USA, pp. 225–236. doi:10.1109/DSN.2017.37.

Gueron, S., 2016. Memory encryption for general-purpose processors. IEEE Secu. Priv. 14 (6), 54–62. doi:10.1109/MSP.2016.124.

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).

Intel, 2018. Intel(R) SGX Protected Code Loader for Linux User Guide.

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

ISO Central Secretary, 2014. Information Technology – Security Techniques – Evaluation Criteria for IT Security – Part 1: Introduction and General Model. Standard. International Organization for Standardization, Geneva, CH.

Jain, P., Desai, S., Kim, S., Shih, M.-W., Lee, J., Choi, C., Shin, Y., Kim, T., Byunghoon Kang, B., Han, D., 2016. OpenSGX: an open platform for SGX research. In: Proceedings 2016 Network and Distributed System Security Symposium. Internet Society, San Diego, CA doi:10.14722/ndss.2016.23011.

Jang, Y., Lee, J., Lee, S., 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. ACM Press, Shanghai, China, pp. 1–6. doi:10.1145/3152701.3152709.

Johnson, S., Scarlata, V., Rozas, C., Brickell, E., Mckeen, F., 2016. Intel® Software Guard Extensions: EPID Provisioning and Attestation Services, p. 10.

Krawiecka, K., Kurnikov, A., Paverd, A., Mannan, M., Asokan, N., 2018. SafeKeeper: protecting web passwords using trusted execution environments. In: Proceedings of the 2018 World Wide Web Conference on World Wide Web - WWW '18, pp. 349–358. doi:10.1145/3178876.3186101.

Kunkel, R., Quoc, D.L., Gregor, F., Arnautov, S., Bhatotia, P., Fetzer, C., 2019. TensorSCONE: a secure tensorflow framework using intel SGX. arXiv:1902.04413 [cs].

Lee, J., Jang, J., Jang, Y., Kwak, N., Choi, Y., Choi, C., Kim, T., Peinado, M., Kang, B.B., 2017. Hacking in darkness: Return-oriented programming against secure enclaves. In: 26th USENIX Security Symposium (USENIX Security 17). USENIX Association, Vancouver, BC, pp. 523–539.

McKeen, F., Alexandrovich, I., Berenzon, A., Rozas, C.V., Shafi, H., Shanbhogue, V., 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. ACM Press, Tel-Aviv, Israel, p. 1. doi:10.1145/2487726.2488368.

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

Microsoft, 2019b. Dedicated HSM - Hardware Security Module | Microsoft Azure. https://azure.microsoft.com/en-in/services/azure-dedicated-hsm/.

Microsoft, 2019c. Key Vault | Microsoft Azure. https://azure.microsoft.com/en-us/services/key-vault/.

National Institute of Standards and Technology, 2019. Security Requirements for Cryptographic Modules. Technical Report. National Institute of Standards and Technology, Gaithersburg, MD doi:10.6028/NIST.FIPS.140-3.

OASIS, 2015. PKCS #11 Cryptographic token interface base specification version 2.40.

Park, J., Park, S., Kang, B.B., Kim, K., 2019. Emotion: an SGX extension for migrating enclaves. Comput. Secur. 80, 173–185. doi:10.1016/j.cose.2018.09.008.

Park, J., Park, S., Oh, J., Won, J.-J., 2016. Toward Live Migration of SGX-Enabled Virtual Machines. In: 2016 IEEE World Congress on Services (SERVICES). IEEE, San Francisco, CA, USA, pp. 111–112. doi:10.1109/SERVICES.2016.23.

Pierre-Louis Aublin, Kelbert, F., Dan O 'Keeffe, Muthukumaran, D., Priebe, C., Lind, J., Krahn, R., Fetzer, C., Eyers, D., Pietzuch, P., 2017. TaLoS: Secure and Transparent TLS Termination inside SGX Enclaves. Imperial College London doi:10.13140/rg.2.2.13308.95368.

Pires, R., Gavril, D., Felber, P., Onica, E., Pasin, M., 2017. A lightweight MapReduce framework for secure processing with SGX. In: 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), pp. 1100–1107. doi:10.1109/CCGRID.2017.129.

Sabt, M., Achemlal, M., Bouabdallah, A., 2015. Trusted execution environment: what it is, and what it is not. In: 2015 IEEE Trustcom/BigDataSE/ISPA. IEEE, Helsinki, Finland, pp. 57–64. doi:10.1109/Trustcom.2015.357.

Sartakov, V., Weichbrodt, N., Krieter, S., Leich, T., Kapitza, R., 2018. STANlite – a database engine for secure data processing at rack-scale level. In: 2018 IEEE International Conference on Cloud Engineering (IC2E). IEEE, Orlando, FL, pp. 23–33. doi:10.1109/IC2E.2018.00024.

Sasy, S., Gorbunov, S., Fletcher, C.W., 2018. ZeroTrace : oblivious memory primitives from intel SGX. In: Proceedings 2018 Network and Distributed System Security Symposium. Internet Society, San Diego, CA, p. 15. doi:10.14722/ndss.2018.23239.

Schwarz, M., Lipp, M., Moghimi, D., Van Bulck, J., Stecklina, J., Prescher, T., Gruss, D., 2019. Zombieload: cross-privilege-boundary data sampling arXiv:1905.05726.

Soriente, C., Karame, G., Li, W., Fedorov, S., 2019. Replicatee: Enabling seamless replication of sgx enclaves in the cloud. In: 2019 IEEE European Symposium on Security and Privacy (EuroS&P). IEEE, pp. 158–171.

Swami, Y., 2017. Intel SGX Remote Attestation is Not sufficient. BlackHat.

Trach, B., Oleksenko, O., Gregor, F., Bhatotia, P., Fetzer, C., 2019. Clemmys: towards secure remote execution in FaaS. In: Proceedings of the 12th ACM International Conference on Systems and Storage - SYSTOR '19. ACM Press, Haifa, Israel, pp. 44–54. doi:10.1145/3319647.3325835.

Tramèr, F., Boneh, D., 2018. Slalom: fast, verifiable and private execution of neural networks in trusted hardware. arXiv:1806.03287 [cs, stat].

Wang, Y., Shen, Y., Su, C., Cheng, K., Yang, Y., Faree, A., Liu, Y., 2019. CFHider: control flow obfuscation with intel SGX. In: IEEE INFOCOM 2019 - IEEE Conference on Computer Communications, pp. 541–549. doi:10.1109/INFOCOM.2019.8737444.

Xu, L., 2010. Securing the Enterprise with Intel aes-ni. Intel Corporation.

Yitbarek, S.F., Aga, M.T., Das, R., 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). IEEE, Austin, TX, USA, pp. 313–324. doi:10.1109/HPCA.2017.10.

Zhao, C., Saifuding, D., Tian, H., Zhang, Y., Xing, C., 2016. On the performance of intel SGX. In: 2016 13th Web Information Systems and Applications Conference (WISA). IEEE, Wuhan, China, pp. 184–187. doi:10.1109/WISA.2016.45.

Zheng, W., Dave, A., Beekman, J.G., Popa, R.A., Gonzalez, J.E., Stoica, I., 2017. Opaque: an oblivious and encrypted distributed analytics platform. In: 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17). USENIX Association, pp. 283–298.

Zimmerman, D., 2019. Intel SGX's Open Source Approach to 3rd Party Attestation.

João Guerreiro concluded in 2019 the integrated M.S. degree in Electrical and Computer Engineering at Instituto Superior Tècnico, Portugal. He was a junior researcher at Multicert and INESC-ID and is now working at GitGuardian as Backend Developer / Data Engineer. His research interest includes system security, application of Intel SGX and systems programming.

Rui Moura completed his B.E. degree in Informatics Engineering at Universidade do Minho, Portugal. He currently works a Systems Architect at Multicert, one of the Portuguese Certification Authorities and the only private organization in Portugal with a Root CA. His research interest includes system and network security.

João Nuno Silva is an assistant professor at the Electrical Engineering Department at Instituto Superior Tècnico, Lisbon University, Portugal. He has a PhD in Computer Science and has been doing research in Distributed Systems, with a main focus on the mobile computing.