Towards Management of Chains of Trust for Multi-Clouds with Intel SGX*

Houssem Kanzari and Marc Lacoste

Orange Labs

{houssem.kanzari, marc.lacoste}@orange.com

ABSTRACT

In multi-cloud infrastructures, despite the great diversity of current isolation technologies, a federating model to manage trust across layers or domains is still missing. Attempts to formalize trust establishment through horizontal and vertical Chains of Trust (CoTs) still lack a precise supporting technology. This paper is a first step towards reconciling the two standpoints towards a broader trust management framework. We consider the horizontal, single-layer case, focusing on Intel SGX as promising isolation technology. We propose a protocol for establishing trust between a chain of Intel SGX enclaves, both when they are located on the same and on remote platforms. Preliminary evaluation of an OpenSGX implementation shows our protocols present encouraging scalability results.

1. INTRODUCTION

In current multi-cloud infrastructures, trust management and isolation might be two sides of the same coin. Protection concerns come from the high level of vulnerability found both in multiple untrusted layers and heterogeneous network security domains.

For single clouds, trust management and isolation were considered so far largely orthogonally. This is due to the considered threat model, capturing where attacks were understood to come from. Thus, security objectives were either: (1) to protect the infrastructure from malicious VMs (or VMs from one another) assuming a trusted infrastructure operated by the provider (*isolation*); or (2) to protect customer VMs from an untrusted cloud infrastructure, where some layers or security domains could be compromised (*trust management*).

In a multi-cloud infrastructure, some clouds might be considered trusted (e.g., private clouds). Others might be less worthy of trust (e.g., public clouds). A comprehensive security framework might thus need to address both objectives simultaneously.

Solutions to reach the first security objective have been extensively explored, both from the system side [9] and the network side [7]. Though, it became rapidly clear that the trusted infrastructure assumption could not hold due to its complexity (e.g., administrative domain) and many vulnerabilities – insider attacks [4] being rather the real threat in terms of data security and privacy. Even assuming a trusted cloud provider, a malicious administrator has normally enough permissions to steal and modify sensitive customer information. Misconfigured device drivers and side-channel attacks are also major infrastructure isolation threats.

Different classes of solutions have thus also been proposed to meet the second security objective, addressing both trust management and isolation. For instance: virtualization architectures based on a trusted layer [3, 8], cryptographic schemes to compute over encrypted data [10], hardware-based trusted execution environments [5] or secure virtual enclaves [2]. Those isolation technologies each have their limitations in terms of performance, scalability, or compatibility with cloud platforms.

Beyond such diversity, two elements seem to be missing for a comprehensive trust management and isolation framework for multi-clouds. A first requirement is to establish and verify the integrity of the link between a virtual machine (VM) and hardware resources. The Trusted Computing Group introduced the *Chain of Trust (CoT)* abstraction: integrity of a component may be verified by following the CoT to a root of trust (RoT), usually a tamperproof hardware element such as a TPM. Abbadi et al. defined a model to describe CoTs in a multi-cloud infrastructure, both vertically (across layers) and horizontally (across domains) [1]. However, it remains unclear how to map this model to concrete cloud isolation technologies.

A second requirement is to guarantee secure execution of VMs with hardware protection even if some intermediate infrastructure layers are compromised. Intel's Software Guard Extensions (SGX) [2] through the *enclave* abstraction for a secure computation unit provides significant enhancements compared to previous isolation solutions (e.g., [5]): it guarantees VM security even if the hypervisor is completely compromised, reducing the TCB only to the CPU chip. This isolation technology could provide a starting point towards a comprehensive security framework handling both types of CoT, and supporting multiple isolation technologies. However, it remains unclear how enclaves can be chained together practically.

In this paper, we propose protocols for establishing trust between chains of Intel SGX enclaves. Our protocols formalize horizontal (single layer) CoT establishment for multi-cloud infrastructures according to the model of [1], both when enclaves are located on the same and on remote Intel SGX platforms. We implemented our attestation protocols on the OpenSGX [6] Intel SGX emulator. Preliminary evaluation results tend to show our protocols could scale to large CoTs, thus being applicable to realistic multi-cloud infrastructures.

This paper is organized as follows. Sections 2 and 3 provide some background on CoTs in a multi-cloud infrastructure and on Intel SGX. Sections 4 and 5 describe the protocols and their implementation. Finally, Section 6 presents some experimental results.

2. CHAINS OF TRUST IN A MULTI-CLOUD

2.1 Multi-Cloud Infrastructure Model

Abbadi et al. proposed a simple architectural model to manage trust in a distributed cloud [1] (see Figure 1).

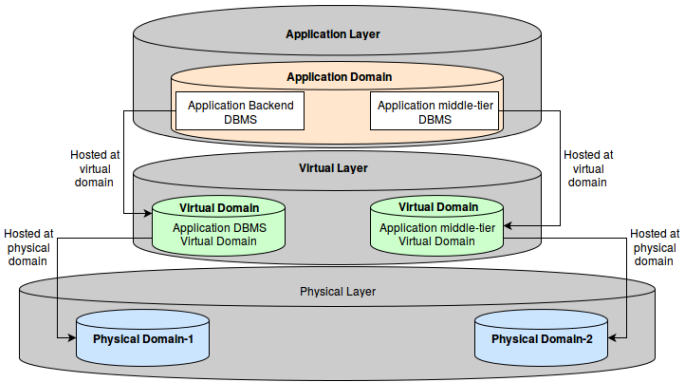

Figure 1: Multi-cloud infrastructure model

*This work is partially supported by the EU H2020 project SUPER-CLOUD (grant No. 643964) and Swiss Secretariat for Education, Research and Innovation (contract No. 15.0091).

Vertically, the infrastructure is modeled as several layers, software and hardware containing resources. The *physical layer* consists of computing (CPU, memory) and storage hardware resources. The *virtual layer* contains the VMs (virtual CPU and memory) and virtual storage. The *application layer* leverages virtualized resources to run applications. One or several *virtualization layers* manage allocation of host resources among VM instances.

Horizontally, the infrastructure is seen as a federation of *provider domains* that manage resources within a given perimeter, and according to a common policy. To simplify, we assume domains are layer-specific. *Domain federations* group together domains in each layer. In reality, the infrastructure is two-dimensional, with cross-cutting layers and domains.

2.2 Chains of Trust

Several techniques enable one party to establish trust in an unknown entity: direct interaction, trust negotiation, reputation, and trust recommendation and propagation. Most of these establish trust based on identity. In a cloud context, establishing trust is rather based on both identities and properties.

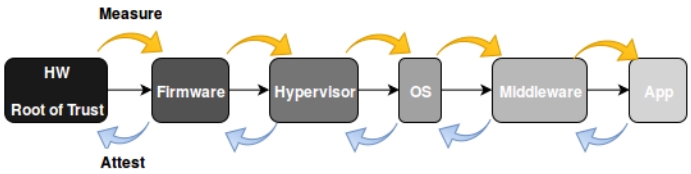

Figure 2: CoT concept

In a *Chain of Trust (CoT)*, the links between elements of the chain represent confidence between two entities: a *Trustor* and a *Trustee*. To establish this confidence, an assertion is needed in order to demonstrate that a piece of software has been properly instantiated on the platform. This process is know as *attestation*: it provides a Trustor with an authentic and fresh copy of the properties of a Trustee. Thus, the Trustor can make timely decision of the ability of the Trustee to operate in certain state.

A *Root of Trust* (RoT) is a component that must always behave in the expected manner: its misbehavior cannot be detected. The RoT set includes at least the minimal set of functions enabling a description of the platform characteristics that affect its trustworthiness.

A CoT provides an iterative means to extend the trust boundary from the RoT set to extend the collection of trustworthy functions. Typically, a CoT could be built as follows. The first element of the CoT (RoT) should be established from a trusted entity or an entity that is assumed to be trusted, e.g., a tamper-evident hardware chip. The RoT then measures the trust status of the CoT second element. As the verifier trusts the RoT, the verifier also trusts the RoT measurement of the second element that is now fully part of the CoT. This process is continued to other elements of the CoT (see Figure 2). This approach may be extended to manage trust relationships in the cloud using CoTs that may cross layers (vertically) or domains (horizontally).

2.3 A Simple Model for Horizontal CoTs

In a single layer, and for a single resource, a CoT may be captured by a triple comprising: (1) a sequence of elements in the chain $sq. < x_0, x_1, .., x_n >$ where x is any component contributing to the CoT; (2) an initial trust function itf for the RoT (x_0) : $itf(x_0) \in \{\text{trusted}, \text{assumedtrusted}\}$; and (3) a set of trust functions stf to extend trust over the next element in the CoT [1]:

$$\forall i \in [1..n], \forall f \in stf, f(x_{i-1}, x_i) == \text{true})$$

3. INTEL SGX

Intel's Software Guard Extensions (SGX) is an extension to Intel architecture for generating protected software containers, referred to as *enclaves*. Inside an enclave, software code, data, and stack are protected by hardware-enforced access control policies that prevent attacks against the enclave content.

3.1 SGX Principles

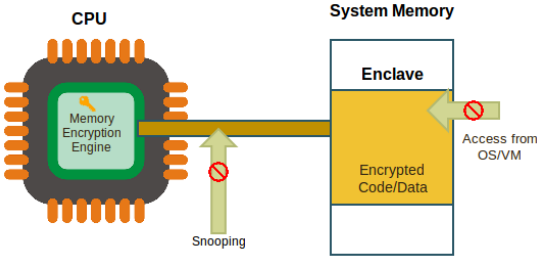

Figure 3: SGX isolated execution

SGX allows part of an application code to run isolated inside an enclave (see Figure 3). The enclave region of the main memory is encrypted. The content is only decrypted inside the CPU using processor-specific keys. The TCB (Trusted Computing Base) is restricted to the CPU and the application running inside the enclave. Even an adversary with extensive control over the hardware cannot access or modify the enclave. The enclave is protected from other software running in the host, including the OS and the hypervisor [1].

In the SGX model, sensitive data is protected within applications. Thanks to this technology, an application may defend its own secrets with a high level of security: even a malware subverting the OS, the VMM, the BIOS, or device drivers cannot steal application secrets.

3.2 SGX Attestation Capabilities

SGX features allows building a CoT based on three elements:

- A RoT for storage materialized by the sealing keys for the enclave software to encrypt and integrity-protect data.
- A RoT for measurement captured by two measurement registers, MRENCLAVE and MRSIGNER. MRENCLAVE returns an identity for the enclave code and data[2]. MRSIGNER returns an identity of an authority over the enclave. These values are recorded when the enclave is built, and are finalized before enclave execution begins. Only the TCB has write access to these registers to reflect accurately the identities available when attesting and sealing.
- A RoT for reporting, equivalent to the report mechanism provided by EREPORT and EGETKEY instructions: an evidence structure called *REPORT* is returned, cryptographically bound to the hardware for consumption by attestation verifiers[3].

With Intel SGX, a trustor can gain confidence that the correct software is securely running within an enclave on the trustee. For this, the SGX architecture produces an attestation assertion that conveys: the identities of the software environment being attested, details of any non-measurable state (e.g. the mode the software environment may be running in), data which the software environment wishes to associated with itself and a cryptographic binding to the platform TCB making the assertion.

To build a CoT, enclaves will need to authenticate one another. The EREPORT instruction provided by the SGX architecture will be useful for this purpose. When invoked by an enclave, EREPORT creates a signed structure, known as a REPORT. The REPORT structure contains the identities of the two enclaves, the attributes associated with the source enclave, the trustworthiness of its hardware TCB, additional information to pass on to the target enclave (e.g., USERDATA), and a message authentication code (MAC) tag.

[1] SGX does not address attacks where an enclave-isolated application legitimately communicates with a corrupt process outside the enclave sending purposely formatted data to crash the application.
[2] This identity is a digest of information related to the enclave (e.g., code, data, memory mapping, security flags).
[3] The REPORT binds the verifier enclave identity with a symmetric key called Report Key, that is only shared between the verifier enclave and the SGX implementation.

3.3 OpenSGX: An Open Platform for SGX

OpenSGX [6] is a fully functional, instruction-compatible Intel SGX emulator to explore the software/hardware design space. It is also a platform to develop enclave programs, providing additional OS components, such as an enclave program loader/packager, and debug and performance monitoring tools.

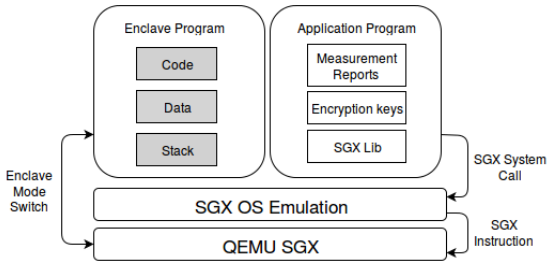

Figure 4: OpenSGX design overview

A packaged program (Wrapper) runs together with the Enclave Program together as a single process in the same virtual address space. Figure 4 shows the memory state of an active Enclave Program (grayed boxes are isolated enclave pages). As Intel SGX uses privileged instructions to initialize and set up enclaves, OpenSGX introduces a set of system calls to service requests from the Wrapper program.

4. COT ATTESTATION PROTOCOLS

Building a CoT implies a series of mutual attestations between neighboring elements of the chain to build persistent links of confidence. Depending whether elements of the chain belong to enclaves within the same SGX platform or to remote platforms, two attestation subprotocols have been specified: intra-platform attestation and remote attestation respectively – a complete CoT establishment being a composition of the two protocols.

4.1 Intra-Platform Attestation Protocol

This protocol establishes trust between two enclaves (A and B) on the same Intel SGX platform. Each enclave authenticates the other. The protocol confirms they both run on the same platform according to the SGX security model. The protocol steps are the following, where A is the Trustee and B is the Trustor (see Figure 5):

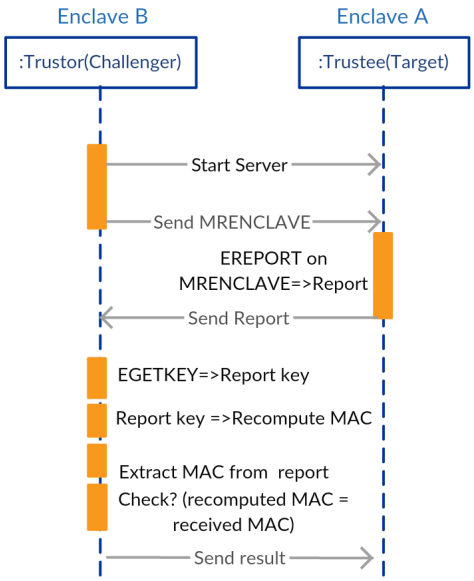

Figure 5: Intra-platform attestation

1. Enclave A obtains the identity of enclave B (MRENCLAVE value).
2. Enclave A invokes the EREPORT instruction to create a signed REPORT sent to enclave B (using the previous MRENCLAVE value) over the untrusted communication channel.
3. When it receives this REPORT, Enclave B calls the EGETKEY to retrieve its Report Key, re-computes the MAC over the REPORT structure, and compares the result with the MAC attached to the REPORT. A match in MAC values means that enclave A runs on the same platform as enclave B.
4. Mutual authentication is achieved by Enclave B creating a REPORT for enclave A, using the MRENCLAVE value from the REPORT it just received.
5. This REPORT is sent to enclave A that can then verify it in a similar manner to confirm that enclave B runs on the same platform as enclave A.

4.2 Remote Attestation Protocol

In case of remote platforms, SGX enables a special enclave called the *Quoting Enclave* to be remotely created. This enclave verifies REPORTs from other enclaves on the remote platform using the intra-platform attestation protocol described above. It then replaces the MAC in those REPORTs with a new MAC computed with the private key of the verifier enclave using a dedicated asymmetric cryptographic scheme. The output of this process is called a QUOTE.

The protocol steps are the following (see Figure 6):

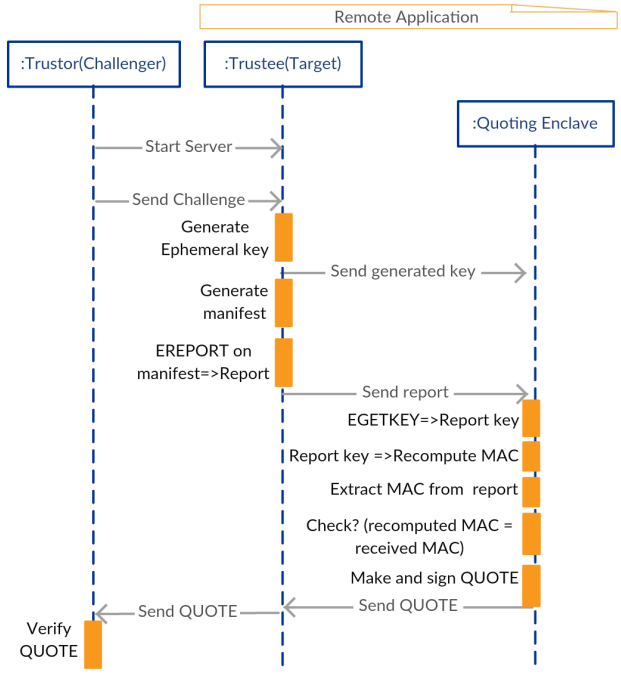

Figure 6: Remote attestation

1. Initially, the Trustor enclave establishes communication with the remote target (Trustee) [4].
2. The Trustor issues a challenge to the target to obtain proof the Trustee can join the CoT. The challenge contains a nonce for liveness purposes. It also contains the Quoting Enclave identity.
3. The target enclave generates an ephemeral public key for the Trustor to use in further communications with the target, and sends the key to the Quoting Enclave.
4. The target enclave generates a manifest that includes a response to the challenge issued in Step 2. A hash of this manifest is included as USERDATA for the EREPORT instruction that will generate a REPORT binding the manifest to the enclave[5].

[4]The trustor start a TCP/IP socket server and the Trustee connects to it. In our protocol, we rely to an asymmetric signing scheme provided by Intel SGX for trust establishment and for further communication we use a dedicated key (the ephemeral key) created on the fly for this purpose. Attacking the channel could always be possible in principle.

[5]By Step 1, the Trustee knows the Trustor is in a remote platform: the REPORT must then be transmitted to the Quoting Enclave for verification and QUOTE construction. This is possible as the Quoting Enclave identity is contained in the challenge generated in Step 2.

Interface for creating keys, signing reports, and checking report integrity:

cot_getkey	Invoked by an enclave. Allows to create a cryptographic key needed for signing a report.
cot_getreport	Takes a generated SGX key. Create, then sign a report.
cot_check	Takes a Report Key. Computes a MAC and compare it with MAC in received report.

Interface for managing communication channels:

cot_make_server	Start a server socket, wait for connection, and return a descriptor.
cot_connect_server	Wrap the connection procedure of a target element to connect with an already started server.
cot_read_sock/cot_write_sock	Customized read/write socket procedures to be used by enclaves programs during attestation.
cot_make_quote	Perform secure hash of the report, generate RSA key for future communications between remotely attested elements, sign the report with the newly created RSA key.

Interface for attestation:

cot_intra_attest_challenger	Called by a challenger enclave. Perform the previously described intra-platform attestation protocol to get an attestation from a target enclave in the same SGX platform.
cot_intra_attest_target	Called by the target enclave. Perform steps expected by the target role to respond to an attestation request.
cot_remote_attest_challenger	Called by the challenger enclave. Start attestation with a remote target enclave specified by an IP address. Perform steps described in the challenger side of the remote attestation protocol.
cot_remote_attest_target	Called by the target enclave. Response to an attestation requested by a remote challenger enclave specified by an IP address. Perform steps described in the target side of the remote attestation protocol.
cot_remote_attest_quote	Called by the quoting enclave co-located with the target enclave. Compute secure hash of received report from target enclave, generate RSA key to send them back to the target.

Table 1: Framework interfaces

5. The target enclave then sends the REPORT to the Quoting Enclave for verification and signing.

6. Playing the role of challenger for the target enclave, the Quoting Enclave retrieves its Report Key using the EGETKEY instruction and verifies the REPORT. The Quoting Enclave creates the QUOTE structure, signs it, and returns it to the target enclave.

7. The target enclave forwards the QUOTE structure and any associated manifest of supporting data to the Trustor enclave.

8. The Trustor uses verifies that the QUOTE was generated by an authentic SGX Quoting Enclave using the asymmetric cryptographic scheme already mentioned. After checking manifest integrity, the Trustor checks the validity of the response the initial challenge.

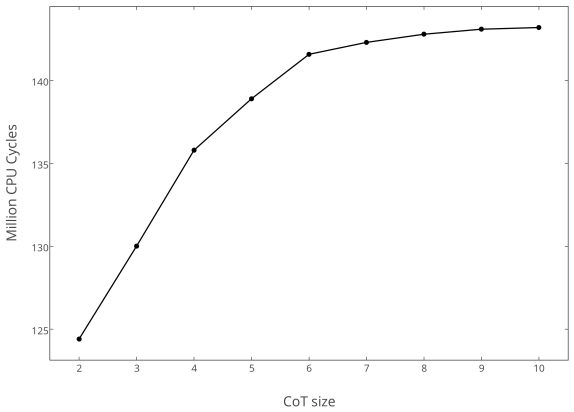

Figure 7: CoT establishment overhead

5. IMPLEMENTATION

The OpenSGX project provides mainly customized libc, lightweight cryptographic libraries and basic wrapper functions for SGX instructions. Our implementation is based on those libraries. It provides a user API to build, verify and study CoTs.

For attestation, SGX allows creating cryptographic keys (EGETKEY) and cryptographic reports (EREPORT) to check the integrity of an enclave during exchanges with other enclaves. We thus specify an interface to create keys, get reports from SGX, and check integrity of reports by comparing the computed report with a received one (see Table 1). An additional interface manages communication channels between elements of a CoT such as network connections between enclaves, and reading/writing to/from enclaves. Finally, a user API is defined based on previously established APIs to perform the main attestation protocol operations. Several procedures are distinguished depending on the type of SGX platform (local or remote) and on the attestation role (challenger or target).

6. EVALUATION

OpenSGX includes performance monitoring features. As it is software-emulated, it cannot provide accurate performance figures. Yet, it may give an idea of potential performance issues. To evaluate the scalability of our protocols, we studied the impact of the number of elements during the establishment of a CoT. We measured the number of CPU cycles consumed by enclaves programs throughout the chain. To capture the effect of the CoT size on computation overhead, we ran the CoT establishment protocol with varying chain sizes (2-10). Results are shown in Figure 7.

We can distinguish a start-up offset ($\sim$ 120 Mcycles) uncorrelated with the CoT size due to some common OpenSGX-related initialization steps. The CoT establishment computation overhead appears as sublinear in terms of CoT size. Those results would tend to show that the proposed protocols could be scalable to larger CoT sizes. This trend could be explained by concurrent execution of attestation between neighboring enclaves, effective remaining costs being due to propagating synchronization throughout the CoT.

7. CONCLUSION

Intel SGX is a serious candidate for isolated software execution, trust delegation and trust management in general. Preliminary scalability results suggest that a hardware RoT could be a cloud-friendly approach. While further enhancements of such techniques need to be followed closely, their wider adoption remains dependent on the will of chipmakers to provide necessary support in terms of APIs. The proposed CoT-building protocols are a first step towards distributed trust management for a plurality of enclaves. Future work includes: (1) extending our framework to manage CoT vertically across infrastructure layers; and (2) integration with a security self-management framework for distributed clouds.

REFERENCES

[1] I. Abbadi. Clouds Trust Anchors. In *TrustCom'12*.

[2] F. McKeen et al. Innovative Instructions and Software Model for Isolated Execution. In *HASP*, 2013.

[3] F. Zhang et al. CloudVisor: Retrofitting Protection of Virtual Machines in Multi-tenant Cloud with Nested Virtualization. In *SOSP'11*.

[4] I. Khan et al. A Protocol for Preventing Insider Attacks in Untrusted Infrastructure-as-a-Service Clouds. *IEEE Transactions on Cloud Computing*, PP(99):1–1, 2016.

[5] J. McCune et al. Flicker: An Execution Infrastructure for TCB Minimization. In *Eurosys'08*.

[6] P. Jain et al. OpenSGX: An Open Platform for SGX Research. In *NDSS'16*.

[7] S. Berger et al. TVDc: Managing Security in the Trusted Virtual Datacenter. *SIGOPS Oper. Syst. Rev.*, 42(1):40–47, 2008.

[8] S. Butt et al. Self-Service Cloud Computing. In *CCS'12*.

[9] T. Garfinkel et al. Terra: A Virtual Machine-Based Platform for Trusted Computing. *SIGOPS Oper. Syst. Rev.*, 37(5):193–206, 2003.

[10] M. Naehrig, K. Lauter, and V. Vaikuntanathan. Can Homomorphic Encryption Be Practical? In *CCSW'11*.