The Interdisciplinary Center, Herzlia
Efi Arazi School of Computer Science
M.Sc. program - Research Track

Trust is in the Keys of the Beholder:
Extending SGX Autonomy and Anonymity

by
Alon Jackson

M.Sc. dissertation, submitted in partial fulfillment of the requirements for the M.Sc. degree, research track, School of Computer Science,
the Interdisciplinary Center, Herzliya

May 2017

This work was carried out under the supervision of Prof. Eran Tromer of the Blavatnik School of Computer Science, Tel Aviv University, and Dr. Tal Moran of the Efi Arazi School of Computer Science, the Interdisciplinary Center, Herzliya.

Acknowledgments

First, I would like to thank Prof. Eran Tromer for the opportunity to start this research. The first steps are always the hardest, especial when entering a state of the art security technology that was mostly undocumented and obscure during the first phases of the research. Thanks for focusing me on the tough questions and for the endless hours, even when in opposite time zones. I would like to express special gratitude to Dr. Tal Moran whose door was always open whenever I ran into trouble and helped me solve some of the most challenging chapters of this work. This work could not have been completed without the valuable expertise of Prof. Shay Gueron from the University of Haifa and Intel Corporation, and Prof. Sivan Toledo from Tel Aviv University. Thanks for steering me in the right directions during the research.

This work was supported by:

- The Blavatnik Interdisciplinary Cyber Research Center (ICRC);
- The Check Point Institute for Information Security;
- The Defense Advanced Research Project Agency (DARPA) and Army Research Office (ARO) under Contract #W911NF-15-C-0236;
- The Israeli Ministry of Science and Technology;
- The Israeli Centers of Research Excellence I-CORE program (center 4/11);
- The Leona M. & Harry B. Helmsley Charitable Trust; and by NSF awards #CNS-1445424 and #CCF-1423306.

Any opinions, findings, and conclusions or recommendations expressed are those of the authors and do not necessarily reflect the views of ARO, DARPA, NSF, the U.S. Government or other sponsors.

Abstract

Intel's Software Guard Extensions (SGX) is a recent processor-based security technology that was introduced in Intel's 6^{th} Generation Core processor (microarchitecture code-name Skylake). It provides a trusted execution environment for software modules, in the presence of malicious software, operating systems, hypervisors, and some hardware attacks. SGX enables developers to provide, with relative ease and flexibility, a high level of security for their applications. It has a promising approach to a wide range of security-critical solutions.

In this work we first present an extensive description of the SGX ecosystem. We provide a "two pillar" representation of SGX's security design that helps conceptualize the SGX technology. We highlight the critical role of SGX sealing and attestation primitives, and elaborate especially about the attestation and provisioning process that fell short in former works.

We build on this description to explore the limits of SGX guarantees in term of availability, privacy and trust assumptions; and the implications in case they are violated. Despite marking the extent in which the technology is sound, these fundamental aspects received little attention thus far.

We propose several techniques that, under certain conditions, "emancipate" SGX sealing and attestation primitives from centralized control and trust assumptions. We aim to reduce Trusted Computing Base (TCB) dependencies by decoupling the security of SGX user's from the security the underlying Intel-dependent infrastructure. We show how a trusted sealing primitive can be leveraged to establish alternative attestation schemes and vice versa. We also suggest a different premise for establishing trust, using a software-based attestation scheme which relaxes strong assumptions backing SGX embedded secrets. These solutions remove trust assumptions as well as amend current availability and privacy limitations.

Thus, our work offers powerful tools for reasoning about, attaining and using secure execution environments.

Contents

1. Introduction

Modern system softwares are extremely complex programs with immense amount of code running on highest privilege level on the platform. This includes the operating system and any intermediate software running between in and platforms hardware (e.g. hypervisor, BIOS, etc.). The large code base underlying modem computers opens the door to a variety of vulnerability opportunities. Compromising system software immediately threatens the security of any application running on that platform Trusted Execution Environment (TEE) is security concept that wishes to address these problems by protecting the integrity and demonstrating the authenticity of sensitive system code.

The concept of TEE has been long studied and pursued by the security community and solutions have evolved from embedded TEEs and Fixed-Function TEEs to general-purpose TEEs, such as the SGX. Early solutions were embedded TEEs (e.g. Smart cards [32]) supplied most notably by IBM, as tamper-resistant platforms [14]. These kind of solutions are intended for monolithic applications that are contained and executed completely on the secure platform.

The Trusted Platform Module (TPM) is a fixed-function unit, which means it defines a limited set of operations that are not sufficient for performing arbitrary computation on the TPM. Instead, the TPM was envisioned to attest that the host it is attached to, a general-purpose computer, runs a trusted software stack. The TCB on the TPMs host computer includes a secure boot loader, an operating system, and drivers. Such a TCB does not exist, because it is impractical to analyze and certify the large codebases of modern operating systems, together with their frequent updates. In practice, TPM applications are not capable of performing trusted arbitrary computation.

In contrast to these solutions, general-purpose TEEs realized by secure processors, protect the logic found inside the CPU. This delivers the full flexibility and higher performance of general-purpose computing commercial CPUs. Intel's Software Guard Extensions (SGX) provides a general-purpose Trusted Execution Environment (TEE) for commercial CPUs. This allows users to run program modules in a "virtual black box", in the sense that the module code and data are protected from outside observation and interference during run-time.

Intel introduced SGX in the 6^{th} Generation Intel® Core™ processor (micro-architecture codenamed "Skylake"). Informal and partial information about the platform appears in Intel's white-papers and some previous publications [41, 12, 33, 34, 37].

SGX is designed to provide a secure execution environment on a platform by: a) isolating a protected application (called an "enclave") from all other processes that run on the system, at any privilege level. This is achieved by a hardware based access-control mechanism; b) protecting the system memory against some hardware attacks. This is achieved by a dedicated hardware unit called the Memory Encryption Engine

[31] (MEE).

The capabilities of SGX are realized by new instructions that allow an enclave to obtain and use keys that are unique to its cryptographic identity and to the platform that it is running on. The root of trust for these capabilities is a unique "master" secret embedded during production into each SGX-supporting processor. The master key is never exposed to software, and never leaves the die. Its derivative keys are made available only to the rightful owning enclaves.

Some of the SGX instructions and the external supporting infrastructure are used exclusively by enclaves, and help facilitate enclaves' essential primitives of sealing and attestation. Attestation can be used by an enclave to prove to an external entity: a) its cryptographic measurement; b) that it is running on a genuine SGX processor, under the SGX restrictions. This allows a secret's owner to establish trust in that enclave and hence provide it with a secret or retrieve a trustworthy answer from it. Other SGX instructions can be used by the enclave software in order to encrypt secrets and store them on any untrusted location for future use.

SGX provides protection against any software or hardware component out of the specific running enclave and out of the processor's package. Sealing and attestation are essential for enclaves to receive sensitive inputs and produce reliable outputs, enclaves depend heavily on the critical role of these primitives in order to thrive in their hostile environment. This observation is the basic motivation for closely examining the core trust assumptions and dependencies of sealing and attestation in the SGX technology. Although marking the extent in which SGX guarantees are sound, these assumptions received little attention thus far. The significance of this assessment is further augmented in light of the promising role envisioned for SGX as a commodity security standard.

Clearly, different users may afford different trust assumptions and dependencies, and different application may wish to ensure specific degrees of anonymity and autonomy for their operations. We thus set to ask whether, and how, software means can be implemented to benefit currently available hardware and enjoy SGX novel capabilities while relaxing some dependencies on the account of other –independently chosen and user-controlled ones.

We provide methods to implement independent enclave primitives based on different set of dependencies. For each case we show how a trusted primitive (e.g. attestation) can be leveraged to implement an independent sibling (e.g. sealing). We then conclude by raising a hypothetical approach that suggests the setup assumptions related to SGX embedded secrets may be speared by a different premise. We show how this approach can be bootstrapped to implement both primitives without relying on Intel's embedded secret, hence providing an "independent-enclave". This allows to enjoy SGX novel capabilities while reducing both manufacturer's responsibly and user's dependencies.

Related work. Prior to the release of SGX as a product, Intel published several informative documents [41] [1] [12] that clarified the concepts, the SGX concepts, enclave's security grantees and its new capabilities. The Software Development Manuals (SDM) [2] [3] defined the SGX instructions formally their integration in the 6th gen ISA and provided a technical overview of and their usage for building, and running SGX enclaves.

Based on these documents, researchers explored various possible SGX-based application designs [33] [39] [53] [16] [23] [40]. These proposals demonstrate the substantial potential of SGX for commodity application security. OpenSGX [35] provided an open source SGX emulation environment for exploration of SGX-based application design space. Barbar, Smith & Wesson's paper [15] provided some formal security framework that definies hardware-based trusted computation inspired by SGX paradigm. A very comprehensive account of SGX was provided by [21, Jan. 31, 2016]. It summarized the publicly available information on SGX (at that time), together with a comparative survey of other trusted hardware solutions. Revised versions of this work were was posted in [21, Feb., 14, 2016], [21, Aug., 12, 2016], to fix some inaccuracies.

Previous reports on SGX included some misconceptions regarding the overall SGX provisioning and attestation protocols, and their rationale. For example, [21, Feb., 14, 2016] mistakenly assumed that the EPID private keys are generated by Intel's provisioning server, and are known (and saved) to that server. Therefore, they raised a privacy concern. In reality, this is not the case (as explained in Section 2): a private key (for joining a group) is generated randomly on the platform, while the provisioning server only signs a blinded version of that key, which is never exposed externally. This confusion was the result of insufficiently detailed documentation of the SGX ecosystem. To address this difficulty, we present here a more accurate description of the SGX provisioning and attestation processes.

Our contributions.

1. Provide an extensive description of the SGX attestation and provisioning ecosystem that fell short in former works.
2. Simplify enclave's security design using a "two pillar" SGX representation and highlight the critical role of its sealing and attestation primitives.
3. Explore the limits of SGX guarantees in term of availability, privacy and trust, and their implications in case they are violated. Despite marking the extent in which the technology is sound, these fundamental aspects received little attention thus far.
4. Offer several alternative sealing and attestation primitives that, under certain conditions, enable users to retrofit SGX advantages while reducing availability, privacy and trust assumptions on enclaves' underlying Intel-dependent infrastructure.

2. SGX overview

2.1. Programing model

SGX programing model introduces a new concept of *enclaves* which are program modules developed especially to run as isolated containers of code and data designed to handle critical application logic. During run time, enclaves trusted boundary is strictly confined to the CPU package and its internals only. An instantiated enclave is isolated from any other software not part of the specific enclave being executed as well as any hardware component on the platform. [1]

Enclaves identity is defined by a SHA-256 hash digest of its loading activity procedure. This includes the information of enclave's code and data, as well as meta-data (i.e.relative locations of each page in enclave's stack and heap regions, its attributes and security flags, et cetera). This cryptographic log of enclave's creation process forms a unique measurement called MRENCLAVE that represents a specific enclave identity.

Independent Software Vendors (ISV) wishing to harden their application with SGX, should first identify sensitive application computation suitable to enclave. Integrity sensitive code such as cryptographic functions or procedures that handle confidential secrets, are some good examples of enclave candidates.

The ISV should provide a certificate alongside every enclave. Enclaves' certificate is called SIGSTRUCT and is a mandatory supplement for launching any enclave. The SIGSTRUCT holds enclave's MRENCLAVE together with other enclave attributes, as elaborated in the next section. SIGSTRUCTs are signed by the ISV with its private key, which was originally signed by an SGX launch authority. Intel is considered the primary enclave launch authority, however other entities can be trusted by the platform owner to authorize launching of enclaves. The respected launch authority is specified by its public key hash signed by Intel and stored on the platform.

The protection that an enclave enjoys does not rely on traditional OS separation of execution privilege levels. Enclaves are executed at the same privilege level of their hosting application and conform to OS traditional resources management such as run time shares, segmentation and paging policies.

Before running enclave logic, the processor changes its execution mode to a new "enclave mode" that protects the enclave from all other software on the platform. The system software is expected to provide enclave management services for untrusted application code to load end run enclaves. Enclaves are thus treated by the hosting software environment as a manageable black box function. This is accomplished by the OS through a new set of SGX opcode instructions.

The new processor instructions introduced by SGX offers enclaves and SGX supporting operating systems with numerous new cryptographic operations. These are divided

into supervisor and user instructions. Supervisor instructions are used by system software mainly to control the initiation of an enclave instance, and manage its teardown and resource allocation. User instructions are used by enclaves' hosting application to integrate its logic flow with enclave functionality. Other SGX user instructions are used by enclaves themselves.

These new processor instructions form the SGX trusted interface which facilitates the full novelty of SGXs capabilities. The root of trust underlying these capabilities derive from platform-unique secrets embedded into each SGX processor. The new processor instructions use these device keys as part of their operation and expose different derivatives of them to the calling software. Crucially, the raw device keys are ensured to never leave the trusted boundaries of the processor, particularly never exposed to any software running on the platform.

An important subset of these functions are exclusively accessible by enclaves and are used to facilitate SGX sealing and attestation. These two primitives are essential for enclaves to receive sensitive inputs and produce reliable outputs. Therefore enclaves depend heavily on the critical role of sealing and attestation primitives in order to thrive in a hostile environment of an untrusted OS. This is the basic motivation the following section, in which we closely examine the core trusts assumptions and dependencies of these primitives in the SGX technology.

The next sub-section portrays a high level description of SGX's fundamental security design. We introduce a two pillar representation that helps establish a good understanding of enclaves' novel capabilities, and grasp the essence of SGX security guarantees.

2.2. Security design: The Two Pillars

We divide enclave's representation to the following two pillars:

1. Enclave's Isolated Execution Environment (IEE).
2. Enclave's trusted security interface.

These two pillars provide the basis on which we describe different security mechanisms later in this section.

2.2.1. Pillar I: Isolated Execution Environment

The IEE provides enclaves' code and data with run time integrity and confidentiality by separating it from any other code that runs on the platform. Namely, other enclaves, other applications, the OS (including higher privileged code such as OS hypervisor or Virtual Machine Monitor (VMM) software). This protection is facilitated by a dedicated firmware (and microcode) that operates within the trusted boundaries of the processor and offer: (1) cryptographic protection for system's external memory via a hardware unit called the Memory Encryption Engine (MEE); and (2) a new hardware enforced access-control mechanism.

Memory Encryption Engine. Due to the limited cache size of modern commodity CPUs, applications' code and data continuously leave CPU boundaries and moved to/from system's main memory (DRAM). For an enclave, this implies leaving its trusted boundaries. This can be exploited by privileged software that normally controls system's main memory, or by a physical attacker (e.g. the platform's owner) who may physically snoop or tamper with platform's bus or DRAM components.

To handle these threats, SGX reserves a configurable DRAM range called the Processor Reserved Memory (PRM) during system's boot up process. Subsequently, all the CPU-DRAM traffic over this range passes through the MEE which protects its confidentiality, integrity and freshness [31].

Hardware enforced access-controls. While inside the processor package, enclave's code and data are not encrypted. Software attacks targeting cache memory may try to reach these clear text pages and thus break enclave's isolation guarantees. Hence, the MEE it not enough to ensure enclave isolation within the boundaries of the processor.

To ensure cache level separation, every context-switch transition to and from enclave execution mode, triggers a flush of enclave's logical core cached Translation Lookaside Buffer (TLB) entries. Additionally, a set of new access-control checks were added to the processor's hardware-firmware implementation. These new access-controls operate on the PRM range and serve a page fault result if any code out of the PRM tries to fetch memory addressees within it. PRM memory addressees are further supervised by the new firmware to isolate pages in accordance to their individual enclave owner. Enforcing that each page request was originated by its owner, ensures that enclaves' IEE also isolates different enclaves from each other. Since programs only use virtual address to manage system's memory, access-control checks are performed merely during address translation (and not for each page request). This design significantly reduces the performance overhead necessary for achieving enclave's IEE. [3][21]

2.2.2. Pillar II: SGX trusted interface

Each SGX-enabled platform has two 128-bit secret keys generated uniformly at random and embedded into write-once fuses during processor's production [1]. The raw form of both fuse keys never leaves the trusted boundaries of the CPU. The SGX trusted interface is realized by a group of new processor instructions that enjoy exclusive access to these device keys and provide new cryptographic operations based on them. Section 2.4 will elaborate more about the production and usage of these keys.

These unique per-die keys provides a cryptographic separation between different SGX platforms. Software can only obtain different derivatives of platform's device keys through the use of the new SGX trusted interface. An important subset of these instructions are restricted for enclave-use only, and bled enclave unique identities (MRENCLAVE or MRSIGNER) into the key derivation process. Hence serviced keys are uniquely binded not only to a specific platform, but also to a specific enclave or group of enclaves, according to the identity chosen. Enclaves can trust these SGX instructions since they are called directly from within the enclave itself and have no intermediate untrusted code serving it. Instructions run as atomic processor operations and their results are served

back directly by the processor to enclave's code.

Another important aspect of the new SGX opcodes is their ability to access any physical memory address of the system. Since these instructions are implemented by microcode and firmware, they do not rely on address translation where SGX access-control is enforced. This allows instructions to provide results based on different isolated enclave data structures without exposing the raw isolated data to the calling software.

The unique properties of SGX instructions enable enclaves to leverage the enclave-only trusted interface to implement novel security operations, especially the two fundamental enclave primitives of sealing and attestation. Understanding these primitives in the SGX context, requires some familiarity with enclave's framework, provided in the following subsection.

2.3. The critical role of sealing and attestation

It would be fundamentally wrong to conceptualize enclaves as trusted software modules rather than software modules that can execute in a trusted IEE. Since it's impossible to debug or monitor enclaves during run-time, their code should be verified by users before execution and only trusted accordingly. However, since the untrusted system software controls enclave initialization and execution, the actual enclave loaded and ran by the system can only be assured with the help of enclave attestation.

The exposed nature of enclaves in rest, also limits their ability to ship and load with hard coded secrets. Consequently, secrets should always be provisioned to an enclave after it has been properly loaded into the IEE. Relying parties willing to ensure SGX security properties for their secrets, must be convinced as to the trustworthiness of an enclave before allowing enclaves to handle confidential information. Once the secret is obtained, an enclave can securely store it for consecutive runs on to untrusted media using the SGX sealing primitive. This is an important capability since attestation is usually a complex and sensitive task that relies the availability and cooperation of several parties, as we describe in Chapter 3.

Sealing and attestation are essential not only for enabling enclaves to receive sensitive inputs, but also to produce reliable outputs with appropriate security guarantees. A Relying party willing to build on enclave's products, must be convinced as to the security guarantees of its origin. Achieving this requires the enclave to either seal its product with an appropriate policy for the relying party to access, or provide an attestation that proves its credibility. To conclude this discussion, enclaves depend heavily on the critical role of sealing and attestation primitives in order to function in a hostile environment of a malicious system.

2.4. Realizing sealing and attestation

This section describes the realization of enclave's sealing and attestation primitives. We start by clarifying how device fuse keys are manufactured and handled by different production actors of the SGX ecosystem. We then describe enclave's sealing capabilities

and provide a detailed explanations of the provisioning and attestation protocols. These details are necessary for assessing the availability, privacy and trust aspects of SGX (Chapter 3), as well as reasoning the methods proposed in Chapter 4.

2.4.1. Installing device root keys

The underlying paradigm of SGX is based on a unique secret generated uniformly at random during production, and embedded into each CPU. Platform's secret consists of two fuse keys: *Root Provisioning Key* (RPK) which is shared with Intel to facilitate future hardware-based attestation, and *Root Seal Key* (RSK) which Intel promises to forget after production so that it is known exclusively by the platform. This enables enclaves to create platform unique values used for both sealing and local attestation, as described shortly.

Understanding how these keys are produced by Intel, and operated on the platform, is essential in order to comprehend enclaves' trustworthiness. Both keys are stored in the same fashion on the platform, but generated and maintained by separate processes and under different guarantees provided by Intel.

Root Provisioning Key. The first fused key created by Intel at manufacturing time, is the *Root Provisioning Key* (RPK). This key is randomly generated on a dedicated Hardware Security Module (HSM) within a special purpose facility called Intel Key Generation Facility (iKGF) which is guaranteed to be a well-guarded offline production facility. Intel is also responsible for maintaining a database of all keys ever produced by the HSM [54]. RPKs are delivered out to different factory facilities, named by Intel's formal publications as the "high volume manufacturing system", to be integrated into processors' fuses.

Intel stores all RPKs as they are the basis of how SGX processors demonstrate their genuinity through an online provisioning protocol detailed in 2.4.5. For this reason, the iKGF also forwards different derivations of each RPK to Intel's online servers as we described later.

Root Sealing Key. The second key burnt into SGX fuses is called the Root Seal Key (RSK). As in the case of the first key, RSK is also guaranteed to statistically differ from part to part. Yet in contrast to the RPK, Intel declares it attempts to erase all production lines residues of this key so that each platform should assume that its RSK value is both unique and known only to itself. Excluding one special key discussed below, all keys provided by enclave's trusted interface base their derivation on platform's RSK [2].

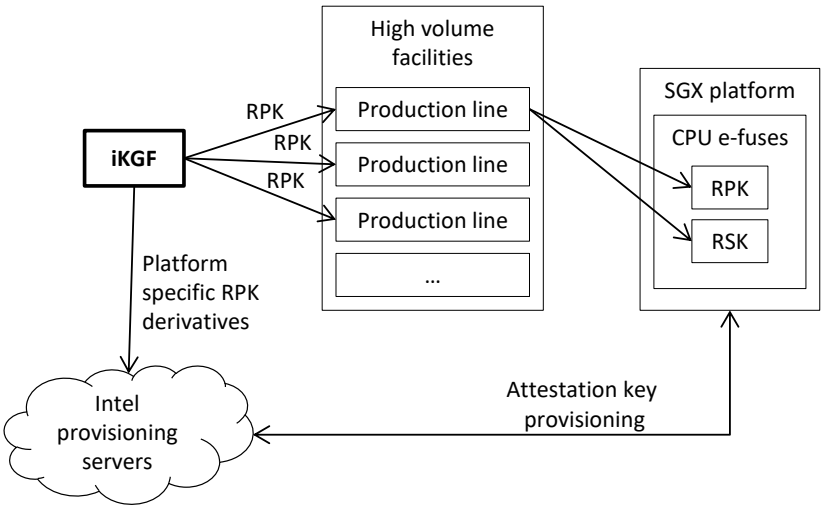

Figure 2.1.: SGX root keys eco-system

Although SGX threat model excludes physical attacks on CPU's package, Intel patents show that at least some preemptive efforts were invested towards tamper proof package architecture [4, 5]. These efforts aim to deter attackers from extracting device keys by raising the cost of performing such attempts. Both keys are stored on e-fuses components which are prone to low cost destructive inspection techniques due to their relatively large feature size. To address this kind of threats, only an encrypted form of the keys is stored on e-fuses. Precise details are undocumented, but recent patents reveal that Physical Unclonable Function (PUF) cells within the CPU package store a symmetric key used to decrypt the e-fuse values during processor's execution [21].

2.4.2. Using root key derivatives.

As described above, a major property of SGX's new instructions is their exclusive access to platform's device keys. Derivatives of device keys are obtained through the use of a new EGETKEY instruction, one of the fundamental services available for enclaves. Here we describe a gently simplified version of EGETKEY derivation specifications which is sufficient to suitably comprehend derivation process and the different characteristics of the resulting keys. Intel's software development manual elaborates this process in more detail [2].

EGETKEY produces symmetric keys for different purposes depending on invoking enclave attributes and the requested key type. There are five different key types, two of which are sealing and report keys available for all enclave. The rest are limited for SGX architectural use only. EGETKEY request includes the type of key requested, e.g. seal, report, et cetera. In case of a sealing key, a sealing policy is specified by the requesting enclave. Sealing policies are described in the following section.

One important EGETKEY derivation parameter is the Security Version Number (SVN). SVNs of different components are specified by the requesting enclave in order to define

the requested key characteristics. Some SVN values used for key derivation are the CPU SVN value which reflects processor's microcode update version, and ISV SVN value used by enclave's author to describe the enclave software version. As depicted in figure 2.3, EGETKEY checks these parameters against the SVN value fixed in the invoking enclave SIGSTRUCT and only allows to obtain keys with SVN values lower or equal to those of the invoking enclave. This key derivation feature is valuable for upgraded versions of the same software to retrieve keys created by former versions. [2]

SGX provides a key recovery transformation process which can be used to dynamically generate keys related to previous SVNs from a newer version key. This is done using a One Way Function (OWF) iteration that transforms newer keys to older ones, assuring that the reverse direction is infeasible. Fig. 2.2 illustrates this process. Using previous SVN values as part of an EGETKEY instruction, enables newer enclave versions to request keys formerly used on the same platform, meaning updated enclave instantiations can decrypt data sealed with keys of their previous versions.

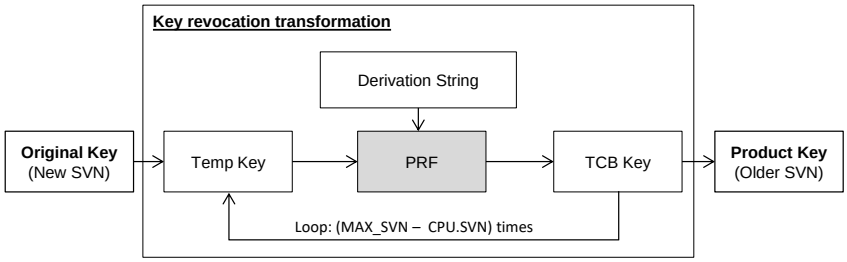

Figure 2.2.: Recovering previous keys from a newer one.

To add user-personal entropy into the key derivation process, a 128-bit value called Owner Epoch is added to the key derivation. This value should be configured by the user during boot time by inserting a password. The value is then stored persistently between power cycles on platform's nonvolatile flash memory. After the platform has booted, Owner Epoch value is stored in a dedicated register and is accessible only by enclave's trusted security interface as an additional parameter for deriving enclave-only products. [2]

The Owner Epoch value must remain the same for an enclave to obtain the same keys used by previous runs. However, it may be desired to change the Owner Epoch value when platform changes hands. This denies access of a new platform user to personal information previously sealed by other users until the correct Owner Epoch password is restored.

EGETKEY instruction internal logic checks the validity of requested parameters against the invoking enclave attributes (e.g. architectural privileges, proper SVN, et cetera). If applicable, it then populates other derivation values such as CPU SVN, Owner Epoch, MRENCLAVE or MRSIGNER (optional) according to invoking enclave properties and the key requested. The global RSK value is included as the root for all EGETKEY derivations, except of the architectural provisioning key described in Section 2.4.5. Finally EGETKEY

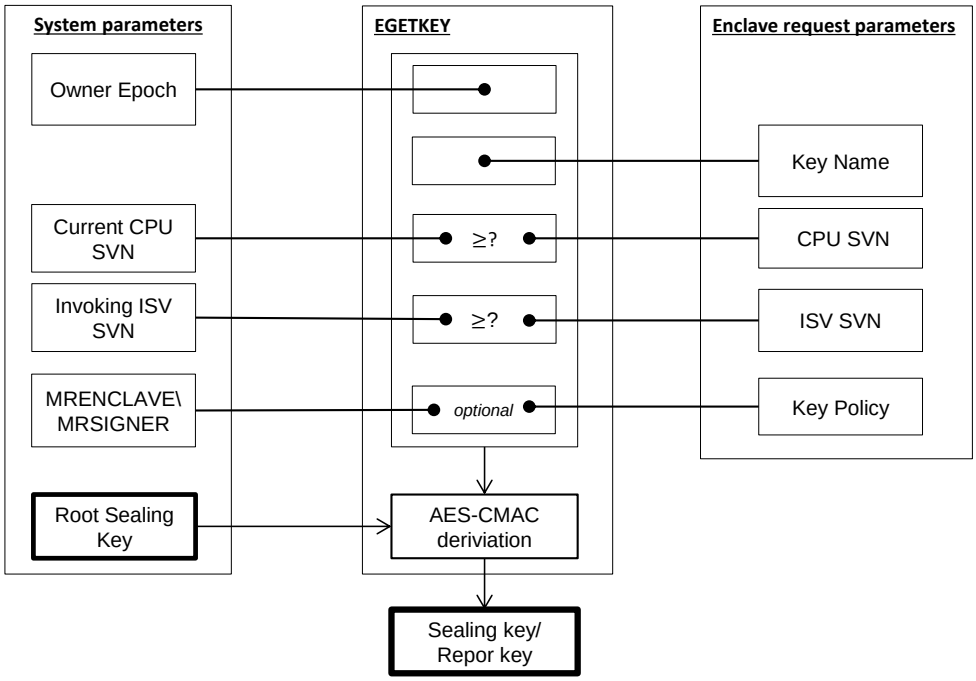

Figure 2.3.: Simplified EGETKEY derivation process

uses an AES-CMAC derivation algorithm to produce the requested 128-bit symmetric key, as illustrated in figure 2.3.

2.4.3. Sealing

Sealing enables enclaves to persistently store sensitive information on untrusted media. Sealing keys come in two flavors, the first allows the migration of secrets between a family of different enclaves (MRSIGNER), and the second between consecutive instances of the exact same enclave (MRENCLAVE). MRSIGNER is a notion introduced by SGX that reflects enclave's sealing authority. This value is represented by a hash over sealing authority's public key and is part of enclave's SIGSTRUCT certificate. The uniqueness property of the sealing key is specified by the requesting enclave through a sealing policy EGETKEY parameter. This can specify MRSIGNER or MRENCLAVE policy, both deriving from the same RSK with different components.

The use of RSK renders the keys from both policies as platform specific. MRENCLAVE policy adds the unique identity of the calling enclave into the key derivation process, resulting with a sealing key which is enclave and platform specific. MRSIGNER policy adds enclave's sealing authority identity to the key derivation. The latter produces a key that can be obtained by any enclave running on the same platform and sharing the same sealing authority (specified by its SIGSTRUCT), i.e developed by the same software vendor.

ISV enclaves are responsible for choosing and implementing the encryption scheme

suitable for their needs when sealing their data. That is, SGX does not provide a complete sealing service, but rather a new security primitive (available exclusively for enclaves) based on EGETKEY features described.

2.4.4. Attestation

Attestation is the process by which a specific software demonstrates its trustworthiness in terms of authenticity and integrity to an external party. In the SGX attestation protocol, an ISV enclave (the prover) running on an SGX platform wishes to prove to a remote relying party (the verifier) its identity, namely its MRENCLAVE, and its correct IEE execution on a genuine SGX processor.

SGX supports two kinds of enclave attestation, local and remote. Local attestation can only be verified by another enclave running on the same platform as the attester, while remote attestations can be verified by any software outside the platform, particularly remote attestation verification does not need to be performed by an enclave.

Using a new EREPORT instruction, enclaves can retrieve a hardware-based assertion describing their software and hardware TCB. The returned report contains enclave's attributes, measurements and ISV additional data. These are adequate for the verifying party to assert what software is running inside the enclave, in which execution environment (including CPU's security level) and which sealing identity will be used by the attesting enclave.

The attesting enclave specifies the MRENCLAVE value of the verifying party and supplies any additional data to be attached to the report. Instruction's internal logic is responsible for populating all other parameters representing the attesting enclave and signing the report with a symmetric key according to the addressee specified. EREPORT outputs the resulting report without revealing the signing key used by the processor. The attester then passes the report to its destination enclave for verification. The verifying enclave uses the EGETKEY instruction with a "report key" type requested to obtain its unique symmetric report key. This key is used by EREPORT to sign all reports generated on that specific platform and destined that for that enclave. Therefore, a local attestation can only be verified by its designated enclave running on the same platform.

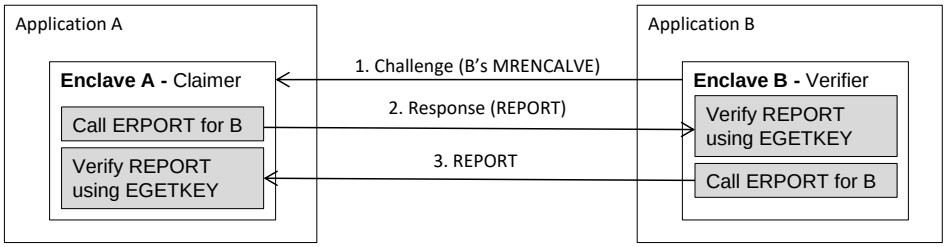

Figure 2.4.: Mutual local enclave attestation protocol.

Apart of the verifying enclave, only the EREPORT internal logic is accessible to its spe-

cific report key. Hence validating report signature inevitably demonstrates the hardware TCB of the attested enclave. If this first validation is successful, the report assertion details can now be trusted as they are assured to be populated by EREPORT using invoker's properties. Witnessing the software TCB of the attested enclave is then done by validating report's details against the expected attestation requirements.

After determining attested enclave trustworthiness, the verifying party is assured as to the authenticity and integrity of any additional data attached by ISV to the report. The verifying party may then reciprocate to the attested enclave in the same fashion. Attaching public keys to report's additional data by both parties can be used to establish a mutually authenticated secure channel between the two enclaves on the platform. [12]

In a remote attestation process, an enclave takes advantage of the local attestation mechanism to obtain a remotely verifiable Quote. This is done with the help of a special purpose enclave called the Quoting Enclave (QE) which is part of a group of SGX architectural enclaves. These enclaves extend the SGX trusted security services where operations are too involved to be implemented as microcode instructions. To transform a local report into a remotely verifiable Quote, the QE uses a platform unique asymmetric attestation key. The Quote can then be verified by a remote party using the corresponding public key. Due to its crucial role in the overall SGX system, remote attestation is explained in detail in the following sections.

2.4.5. Platform Provisioning

In this section we provide a detailed explanation of the provisioning process in which an SGX platform receives its remote attestation key. In the following section we describe the remote attestation protocol. Namely, how the attestation key is used by the platform.

Provisioning in this sense, is the process by which an SGX device demonstrates to Intel its authenticity as well as its CPU SVN and other system components attributes, in order to receive an appropriate attestation key reflecting its SGX genuinity and TCB version. Normally, provisioning is done during platform initial setup phase, but re-provisioning may be performed in the field (after purchase) due to attestation key loss or in cases of TCB recovery. The latter case occurs when releasing an update to remedy critical security issues of certain system components, such as firmware, BIOS or architectural enclave vulnerabilities. In such cases, the attestation key may be replaced to reflect platform renewed TCB security level.

Attestation keys are sensitive assets in the SGX ecosystem. Relying parties trust valid attestation signatures as an Intel signed certificate witnessing platform's authenticity. To facilitate SGX provisioning services, Intel operates a dedicated online provisioning infrastructure. SGX provisioning and remote attestation protocol follow a group signature scheme developed by Intel called Enhanced Privacy ID (EPID) [18]. To implement the EPID provisioning process Intel provides an architectural enclave called the Provisioning Enclave (PvE).

The Provisioning Enclave. The PvE is responsible for conducting the provisioning process on the platform against Intel's online provisioning servers. In this process the PvE demonstrates it has a key that Intel put in a real SGX processor and in return, is

provisioned with a unique platform attestation key for future remote attestations. Both sides implement the EPID scheme Join protocol; the PvE functions as a new joining member and Intel serves as the group membership Issuer issuing new group membership credentials [1].

To prove its authenticity, the PvE uses several SGX privileged key types which are accessible through EGETKEY only by SGX architectural enclaves. Two of these special purpose keys are the Provisioning key and Provisioning Seal key. The uniqueness of SGX architectural enclaves is based on their SIGSTRUCT certificates which are signed directly by Intel (as its MRSIGNER). Architectural enclaves are thus authorized to launch with privileged attributes permitting them to later obtain these restricted key types from the EGETKEY instruction.

The derivation process of the Provisioning key is constructed of two phases. The fist binds platform's embedded RPK to its current hardware TCB level, and the second adds system software properties to the resulting Provisioning key. The first derivation phase occurs prior to PvE's key request, very early during processors boot time. The key transformation mechanism produces a hardware specific TCB key by looping over PRK, according to the current platform SVN patch level which reflects platform's firmware components and affects different SGX instructions. [2] The second phase occurs only on EGETKEY invocation and uses the precomputed hardware specific TCB key as the basis for its derivation. PvE's software elements are reflected by EGETKEY input parameters, as described above. RSK is ignored in this special EGETKEY case, enabling Intel to uniquely identify the platform by pre-computing the expected Provisioning key. Owner Epoch value is also omitted in this case in order to render the same platform-specific key regardless of its current owner. The final result is a unique provisioning key, expressing the TCB level of both hardware and software components of the SGX platform.

A desirable product of this approach is a reduction of usage, and thus the exposure, of RPK itself. Using only OWF derivatives of RPK out of the iKGF and the trusted boundaries of the CPU, prevents the original fuse key from being compromised if any derivatives are leaked. [54]

2.4.5.1. The provisioning protocol

We now describe the provisioning protocol. Figure 2.5 illustrates the initial provisioning process (not a re-provisioning scenario). EPID properties and remote attestation protocol will be discussed shortly.

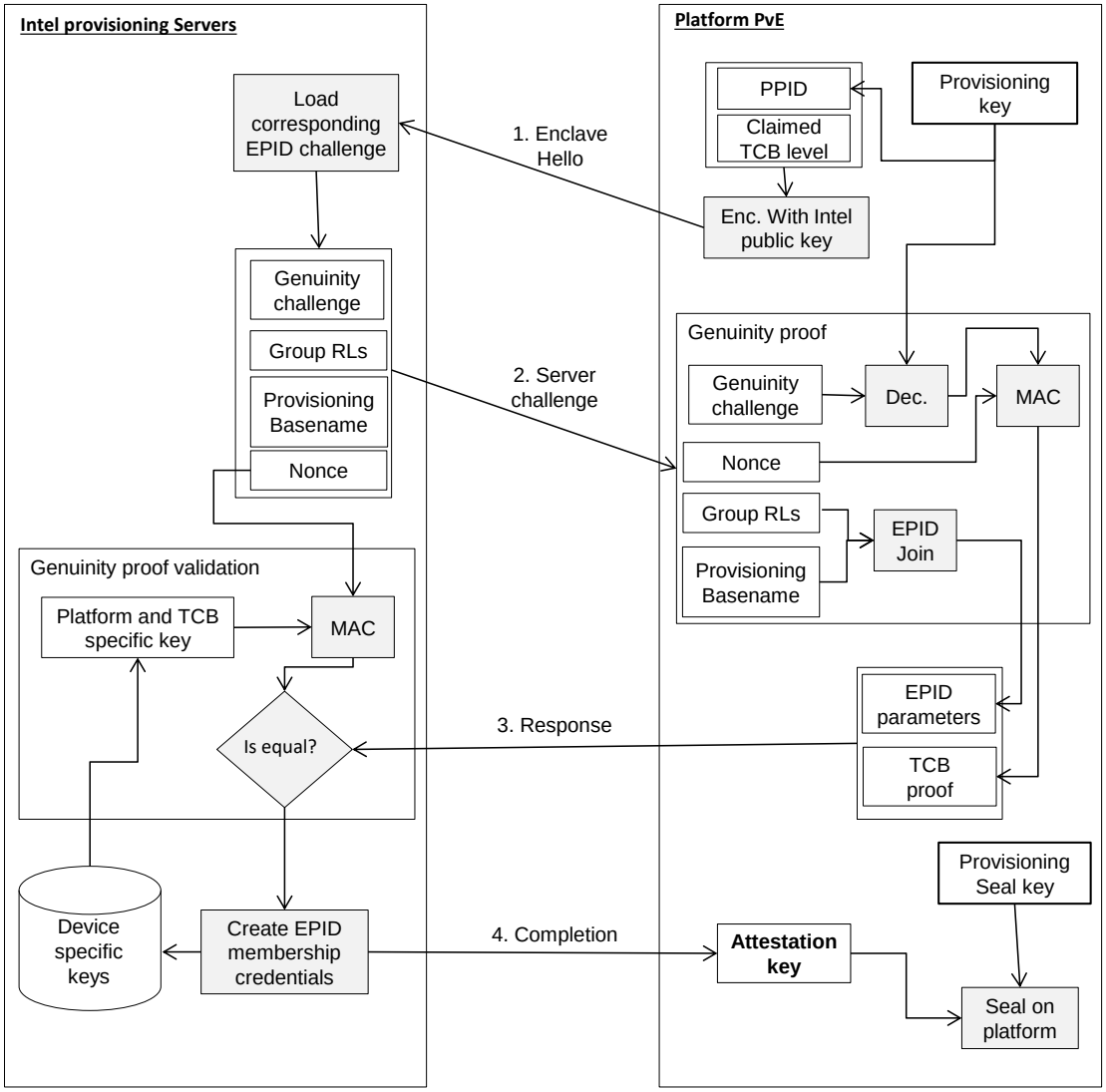

Figure 2.5.: Provisioning protocol - Inital previsioning case.

1. Enclave Hello: After obtaining the hardware TCB specific provisioning key, the PvE generates two values for initiating the provisioning protocol. The first is a OWF product of the provisioning key called the Platform Provisioning ID (PPID). The second reflects platform's claimed TCB level based on its current SVN. Both values are encrypted using Intel's provisioning server public key, and sent to the provisioning server.

2. Server challenge: The PPID value is used by Intel to determine if the platform has been previously provisioned. If so, an encrypted form of a previously generated attestation key (that corresponds to the received PPID) is added to the server's challenge. If not, the server determines the appropriate EPID group for

that platform, and adds the appropriate EPID group parameters together with a liveliness nonce and a pre-computed TCB challenge to the message sent back to the platform.

Since all RPKs are stored by the offline iKGF, it can perform the same hardware and software TCB specific derivation process as performed by the PvE (using EGETKEY) on every individual SGX device to produce its own unique provisioning key. For each SGX platform, the iKGF computes the corresponding TCB specific provisioning key. This is used to encrypt a random value producing a platform specific TCB challenge. All pre-computed challenges are sent to Intel's online servers to support the provisioning protocol. Hence a platform will demonstrate its TCB level by providing a valid response to a specific TCB challenge in the following manner.

3. Enclave response: After the PvE decrypts the received TCB challenge with its provisioning key, it uses it to produce a TCB proof by using the TCB challenge as a key to CMAC the nonce received from Intel. Next, the PvE generates a random EPID membership key and hides it mathematically (according to the EPID protocol) [18] so that Intel's provisioning server cannot learn the unique membership key generated by the platform.

   To facilitate future attestation key retrieval service, the non-hidden membership key is encrypted by the PvE using another special key only obtained by architectural enclaves, called Provisioning Seal key. PvE's privileged key permissions, enables it to request this key from the EGETKEY instruction. Similarly to the Provisioning key, the Provisioning Seal key derivation does not include the Owner Epoch value. Different from the provisioning key, this key follows the general EGETKEY case of using platform's RSK as the root key for derivation. This generates a sealing key which is not affected by the platform changing owners, and is trusted to be exclusively known only by that specific platform.

   If the platform has been formerly provisioned, meaning the ongoing protocol is an attestation key retrieval or a TCB update process, the platform has to also prove it has never been revoked in the past. This is done using the platform's Provisioning seal key to decrypt the backed up attestation key copies obtained from the server, and using them to sign a selected message chosen by Intel.

   Both the hidden and the encrypted EPID membership keys are sent, together with the TCB and non-revoked proofs, as part of platform's response.

4. Completion: On receiving the response, the provisioning server first validates the TCB proof using the corresponding values received from the iKGF and continues the EPID Join protocol on success. The hidden membership key is processed to create a unique certificate signed with the EPID group issuer key and stored together with the encrypted membership key for future re-provisioning events. The final message completing the protocol is then sent by the server containing the signed certificate.

Platform's membership key together with the matching signed certificate, form a unique EPID private key. It is important to note that the attestation key is constructed combinedly by both parties, according to the EPID scheme in a manner that leaves it unknown to the issuer. This promises that no one (including Intel) can forge a valid membership signatures produced by the platform.

5. Finally, PvE encrypts the attestation key with its Provisioning seal key, and stores the encrypted result on the platform for subsequent use. Since EPID groups are categorized by according to TCB levels, Platform's EPID signature can thus be used hereafter ro represent both platform's SGX genuinity and its TCB level.

2.4.6. SGX remote attestation

In contrast to the EPID attestation protocol, the SGX version of it does not allow relying parties to play the role of an EPID verifiers. Instead, Intel provides a worldwide online verification infrastructure, called Intel Attestation Service (IAS). The SGX version of the EPID attestation protocol involves two Intel controlled services and two independent actors. Each couple is divided to prover and verifier: In the first couple the QE and the IAS play the role of prover and verifier respectfully. In the independent couple, the ISV attesting enclave and the service provider play the role of prover and verifier respectfully.

The Quoting Enclave. The QE is an architectural enclave responsible for providing remote attestation assertions called Quotes. Quotes are signed by platform's attestation key. Since this key is sealed using the architectural Provisioning seal key, only only the PvE and the QE can obtain. This priviledge is specified for these two enclaves by a provisioning attribute in their Intel signed SIGSTRUCT.

To enforce Quotes are only served to enclave programs executing within a proper enclave-mode, the QE services are provided using local attestation. The QE receives a EREPORT (a local attestation assertion) from the requesting enclave and verifies it. If the EREPORT is valid, the QE will then sign the local report with platform's attestation key, converting it into a Quote. [12]

SGX service providers. Relying parties are referred to as service providers and do not have to hold SGX enabled hardware. Service providers are expected to register to the IAS and meet a set of Intel defined requirements in order to submit attestation evidence for IAS verification. This registration binds service providers' Transport Layer Security (TLS) certificate to a unique Service Provider ID (SPID), and permits access to the IAS services. Some of these main IAS services are: Verifying ISV enclave Quotes, requesting updated attestation revocation lists and retrieving the assertion information history associated with a Quote (past IAS verification reports). [6]

Remote attestation modes. The QE supports two Quote signature modes with different linkability properties, Fully-anonymous and Pseudonymous Quotes. The linkability property of a Quote is determined by a basename parameter signed using platform's unique attestation key. Using the same attestation key to sign the same basename parameter multiple times yields pseudonymous Quotes that are easily linkable. This mode is used by service providers to keep track of revisiting users and protect against sybil

attacks, while preserving user's privacy. When a pseudonymous Quote is used, the IAS first validates that the basename used is associated to that specific service provider [6]. This role of the IAS enforces user's pseudonymous separation between different service providers. In contrast, by signing multiple signatures on different basenames, it is computationally infeasible to determine whether the Quotes were produced using the same attestation key or not, thus preserving platform's anonymity. Therefore random basenames are used by the QE to sign Fully-anonymous Quotes.

Revocation lists. SGX facilitates three types of Revocation Lists (RLs): Group-RL which holds all revoked EPID groups, Priv-RL listing all revoked private-keys of the same EPID group, and Sig-RL that lists tuples of a basename and its corresponding signature of all revoked members in the same EPID group.

2.4.6.1. Remote attestation protocol.

We now describe the remote attestation protocol illustrated in figure 2.6. To reduce the complexity of this highly involved process, we choose to omit the internal flow of messages between the enclave and its hosting application. Since the hosting software is considered untrusted, we exhibit its presence in the process but consider it as another intermediate layer of communication between the enclave and its external parties. Different insensitive logic may be introduced by the hosting applications to manage the enclave in this process.

1. At first, the ISV enclave sends out an initial request to the desired remote service provider. The request includes the EPID group the platform claims to currently be a member of.

2. If the service provider wishes to serve members of the claimed group, it may proceed by requesting an updated Sig-RL (corresponding to platform's specific group) from the IAS.

3. The service provider then constructs a challenge message that consisting of its SPID, a liveliness random nonce, the updated group Sig-RL and an optional basename parameter - if a pseudonym signature is required.

4. If the enclave supports the requested signature mode, it invokes the EREPORT instruction to create a locally-verifiable report addressed to platform's QE. To establish an authenticated secure channel between the enclave and the service provider, a freshly generated ephemeral public key may be add to the local report's additional data field. This EREPORT, together with service provider's challenge is sent to the QE.

5. The QE verifies the EREPORT using the appropriate report key obtained by EGETKEY, as described in Section 2.4.4 above. On success, the QE invokes EGETKEY again to receive platform's Provisioning Seal Key, this key is then used to decrypt platform's remote attestation key. The attestation key is first used to produce an identity signature by either signing the challenged basename or

a random value, according to the attestation mode requested. If a non-random basename is used, the signature reflects platform's pseudonymous identity; else the identity signature is fully anonymous.

The attestation key is then used to compute two signatures of knowledge over platform's identity signature. The first proves the identity signature was signed with a key certified by Intel. The second is a non-revoked prof that proves the key used for the identity signature does not create any of the identity signatures listed in the challenged Sig-RL. The final Quote is then encrypted using IAS's public key, which is hardcoded in the QE code, and the result is sent back to the attesting enclave.

The resulting assertion, the Quote, holds the identity of the attesting enclave, execution mode details (such as SVN level) and additional data associated by the attesting enclave, as illustrated in figure 2.6. [37]

6. The enclave then forwards the Quote to the service provider for verification.

7. Since the Quote is encrypted, it is verifiable exclusively by Intel. Hence, the service provider simply forwards the Quote to the IAS for verification.

8. The IAS examines the Quote by first validating its EPID proofs against its identity signature. It then verifiers the platform is not listed on the group Priv-RL by computing an identity signature on the Quote basename for *each* private key in the list, and verifying that none of them are equal to Quote's identity signature. This finalizes the validity check of the platform, and the IAS then creates a new attestation verification report as a response to the service provider. The Attestation Verification Report includes the Quote structure generated by the platform for the attesting enclave. [6]

9. A positive Attestation Verification Report confirms the enclave as running a particular piece of code on a genuine Intel SGX processor. It is then the responsibility of the Service Provider to validate the ISV enclave identity and serve an appropriate response back to the platform. [6]

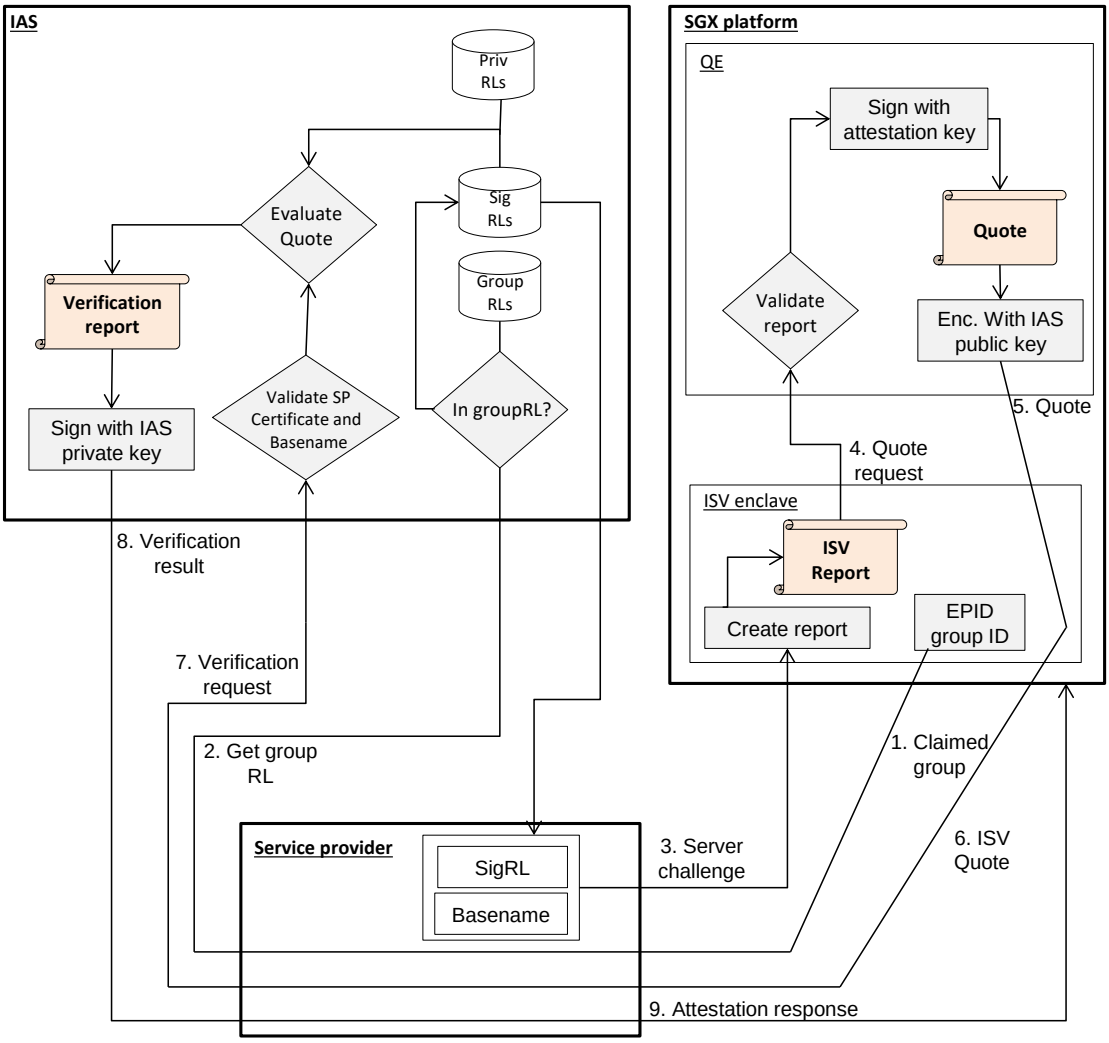

Figure 2.6.: Remote attestation protocol.

3. Availability, privacy and trust assessments

SGX holds several intrinsic assumptions regarding the reliability of platform's unique keys and their derived enclave functionalities. A close examination of the SGX underlying assumptions is essential for consciously trusting its novel capabilities.

This section builds on the description provided above to explore the limits of SGX guarantees in term of availability, privacy and trust. Despite marking the extent in which the technology is sound, these fundamental aspects received little attention thus far.

We start by presenting the availability dependencies of current SGX remote attestation mechanisms, and reason about its necessity in present design. We then examine the privacy degree provided by enclave's anonymous attestation capability, and present several potential attacks to deanonymize SGX users. Finally, we explore the overall trust assumptions underlying SGX security guarantees, and point out the intrinsic dependencies of enclaves' second pillar reliability.

This section provides the main motivation for our solutions presented in the following section that aim to reduce SGX external TCB dependencies.

3.1. Remote attestation availability

By design, the IAS attestation mediator is required as an active party in every remote attestation. In this section, we discuss possible reasons for this design choice and explore some of the problems it causes. In particular, we describe common use-cases in which depending on the IAS is expensive or infeasible.

Intel as an organization plays an extensive role in the SGX attestation ecosystem, providing a worldwide infrastructure to enable both the delivery and use of remote attestation. The EPID attestation scheme used by SGX expands the Direct Autonomous Attestation (DAA) scheme [17] by adding enhanced revocation capabilities. However, the SGX implementation of EPID adds the IAS attestation mediator, which contradicts a major DAA achievement of removing the need for a trusted third party mediating every attestation between the prover (SGX platform) and a verifier (service provider).

Interacting with Intel's online services for each attestation process has several consequences and significantly restricts the usage of SGX based applications. First, attestation can only be carried out when Intel servers are reachable by the relying party. Attestation is Therefore impossible in different situations such as intranet (e.g. enterprise network).

This dependency on Intel server online availability also restricts the potential use of SGX to implement trusted input-output paths between an enclave and the platform

user. Secure and authenticated user-enclave interaction could be realized using direct attestation between the processor and its peripherals (say a laptop and its keyboard or display).

Additionally, critical applications whom wish to use SGX capabilities for sensitive purposes such as governmental or financial business, may not afford such external dependency for their proper operation. Especially as Intel states explicitly that IAS servers may suffer from unplanned downtimes and limited availability. [6]

Finally, attestation verifiers wishing to use the services of the IAS, are bound to perform a prerequisite registration process. This includes commitments to meet some terms of service requirements [7] such as complying with various service level agreement standards and serving the delivery of different Intel applications and updates to any requesting SGX platform. These prerequisites significantly limit the amount of relying parties capable of conducting remote attestation to those whom can comply with the requirements. Additionally, it also eliminates enclaves' ability to establish secure channels with spontaneous remote parties.

Why did Intel design attestation this way? There are two fundamental reasons Intel states to justify current availability restrictions; understanding them is crucial for the assessing alternative SGX attestation mechanisms that are independent Intel servers. [54]

One reason, Intel claims, is to protect service providers from compromised platforms, while at the same time protect the platform's pseudonymity against malicious service providers. To protect service providers, revocation lists are used to declare compromised SGX platforms. As part of a remote attestation protocol, attesting parties compute a proof attesting they are not on such a list.

However, attestation revocation lists could be used by malicious service providers to break user's pseudonymity. This can be done by invoking several attestation requests each with a different basenames hence linking different pseudonym identities of the same attesting platform. Therefore, an attesting enclave needs to assure that a received challenge consists of a suitable basename owned by a legitimate service provider. Since Quotes are always encrypted by the QE with IAS's public key, this validation can be safely delegated to the IAS.

As described in Section 2.4.6 this validation is supported by the IAS registration process that binds service provider's to a distinct SPID and to a unique pseudonymous basename. This role of the IAS enforces user's pseudonymous separation between different service providers.

Another potential attack threatening to degrade platform's privacy, is the "one tuple distant Sig-RL". Since the QE will only create a valid Quote if it succeeds in generating a Sig-RL non-revoked proof, a malicious service provider may deanonymize a platform by challenging it with a rogue group Sig-RL that consists of one tuple or two RLs that are one tuple different. A wrong non-revoked proof, or inability to respond to certain challenges can expose platform's identity by linking it to different pseudonymous signatures.

The IAS thwarts this threat by enforcing the constraint that only the most up-to-

date revocation list can be used with every attestation. The IAS validates that every Quote consists of a valid and up-to-date Sig-RL signed by Intel. Each EPID group has one updated Sig-RL version, a service provider must use the most recent Sig-RL when challenging a platform in order to receive Quote validation results from the IAS [54]. This verification allows the use of revocation lists without exposing platforms to pseudonimity attacks.

Additionally, Intel's involvement in every remote attestation allows it to blacklist (or whitelist) service providers and SGX platform to protect parties from compromised or maliciously exploited remote parties. However, Intel does not specify how such cases are to be detected or what is considered as incriminating evidence for revoking platform's or service provider's attestation capability. Notably, this mechanism can also be used by Intel for commercial purposes.

3.2. Privacy assessment

In this section we wish to examine the privacy aspects of enclaves' anonymous attestation capability. We first assess the degree of anonymity provided by the fully-anonymous attestation mode, and explore passive privacy threats possible by "honest but curios" attestation verifiers. We then describe potential active attacks threatening both users and service providers' as a result of Intel's centralized role in governing every SGX attestation.

3.2.1. Passive privacy threats

In order to consider the degree of anonymity attained from a privacy preserving protocol, such as SGX remote attestation, we use the definition of anonymity as defined by Pfitzmann and Hansen [44] and the degree of anonymity as proposed by Sweeney [55].

Pfitzmann and Hansen define anonymity as: The state of being not identifiable within a set of subjects, called the anonymity set. Understanding the anonymity set characteristics is the key for assessing the anonymity property of any protocol.

Sweeney proposes a k-anonymity model that estimates the anonymity level of every member within a set of users by observing the information revealed when dealing with group members. An anonymous protocol provides k-anonymity protection if the information available regarding any member of a group cannot be distinguished from at least k-1 other members in the group.

EPID groups in the SGX ecosystem. As mentioned in Section 2.4.5, SGX platforms are managed using multiple EPID groups, each reflecting the TCB level of all its members. Intel's white paper [54] shows that EPID groups are further categorized by the specific model of the SGX processor. However this does not mean all SGX platforms of the same TCB level and CPU model are blended together within the same anonymity group. A typical, fully populated group holds between a million to a few million members. This fixes the upper bound of k-anonymity protection provided by the attestation mechanism, even when using the "fully-anonymous" attestation mode.

Since the management of EPID groups directly effects the degree of privacy preserved by SGX remote attestations, users are asked to make several trust assumptions regarding their EPID group in order to maintain a k-anonymity level. For example, users trust Intel to manages groups that are at least k populated at all times, users trust groups will include diverse members. e.g. not partitioned by business, geographical or other categories, et cetera.

Considerations regarding the platform-specific information exposed to service providers should aim to release the minimal set of platform specific attributes that satisfy service providers' requirements. Certainly, many applications would only require to assert that a specific enclave is ruining on a genuine SGX processor patched with an SGX update not under a certain acceptable TCB level. In such cases, SGX attestation discloses excess platform information.

From the service providers' point of view, user's k-anonymity level increases by one only when a collision occurs between customers from the same EPID group. Considering Intel's near-monopoly on desktop processors and considering the limited k-anonymity upper bound of EPID groups (million to few millions), the projection of customers from the same group on a small-medium scale service provider, yields a reasonably low chance of collision. Hence expected k-anonymity level are likely to harm users' privacy expectations. Accordingly, a large scale service provider expands the space for collision, which contributes to users' k-anonymity level.

Additional attestation information disclosure. In addition to the limited privacy provided by EPID groups, enclave remote attestation reveals extra platform-specific information to the service provider. That is, even if all EPID groups are fully populated, platform's actual anonymity degree is less than would appear simply judging by its group size.

As discussed in 2.4.5, re-provisioning is a costly process for Intel as it requires the renewal of platform's TCB specific provisioning credentials. This process involves the offline iKGF facilities to generate new platform specific TCB credentials and securely pass them to Intel's online provisioning servers. Only then, the platform can reach out for re-provisioning.

To minimize re-provisioning scenarios, TCB recovery (issuing new attestation keys) is not mandated for every SVN update to SGX components. Intel is free to decide when re-provisioning is required in order to demonstrate system's TCB level. Therefore, the attestation key owned by a platform is independent of system's patch state and only represents a valid credential previously provisioned to it. In contrast to the provisioning key it self which is derived from current CPU SVN level.

This means that although obtaining an updated CPU SVN value, a platform may still create Quotes that attest to older SGX security versions. Hence service provider's learn not only the EPID group associated with the platform, but also the SVN update of different SGX components of the platform (e.g. the QE SVN) [6]. To grasp the importance the implication of this on platforms' privacy, consider a p amount of patch updates to be uniformly distributes across an EPID group. The privacy protection level gained when using a "fully-anonymous" attention now degrades to k/p-anonymity.

Service provider 1								
EPID 1							EPID 2	EPID 3
SVN 1					SVN 2	SVN 3	...	
PSE 1			PSE 2	PSE 3	...			
IP 1	IP 2	IP 3	...					
etc..								

Table 3.1.: Attesting user quasi-identifiers formed of EPID sub-groups.

1st layer: Service provider's point of view. The only distinction in a true fully-anonymous attestation (Cannot be merged with neighboring columns)
2nd layer: All EPID members (up to few millions).
3rd layer: SVN greater than TCB level
4th layer: PSE information disclosure
5th layer onwards: Network "fingerprints" (users may invest efforts to manipulate these parameters and blur this sub-group distinction)

Moreover, some ISV enclave's can optionally use the services of another architectural enclave called Platform Service Enclave (PSE). In such cases, attestation will convey an additional property descriptor that significantly adds to the differentiation of the attesting platform. The additional property descriptor can hold up to 12 possible values. Most properties disclose software update versions, but some also disclose hardware properties of the platform which reflect stronger platform-identifying properties. Additionally, any subgroup of 11 of the 12 possible properties may be stated in a single attestation. This yields an additional 2047 possible distinct attestation PSE descriptors.

To consider the exact k-anonymity level provided by SGX attestation we must explore all information that can be deduced by service providers. Table 3.1 illustrates the different layers of platform properties that degrade the "fully-anonymous" attestation mode guarantee. This information forms a quasi-identifier of SGX users. The last three layers are examples of the endless measures service providers use nowadays to deduce useful knowledge about their customers.

Arvind and Vitaly explore the shortcoming of k-anonymity privacy guarantees [43]. In their work they use real service providers' data sets and show how attributes associated with a given quasi-identifier may not be sufficiently diverse. They demonstrate this by de-anonymizing users in a Netflix k-anonymity protected data base. They thus conclude that k-anonymity assurance alone, does not provide meaningful privacy guarantees in practice.

Group size considerations. Small groups offer greater management granularity to Intel and requires less work for group manipulation such as revocations. Additionally, small groups also favor with SGX users performance consideration; The computation time of both non-revoked proofs and Sig-RL verification grows linearly in the size of group's Sig-RL. This influences all remote attestation parties in terms of computation complexity and network traffic, as large groups can eventually be accompanied by a large

group Sig-RL. Once an entire group is revoked, its Sig-RL and Priv-RL are no longer required. Therefore group-based revocation is the highest revocation priority among the three revocation methods in terms of performance for attestation parties [45]. On the other hand, the cost of re-provisioning an entire EPID group is a significant concern in Intel's eyes. Hence, the main motivation for Intel to manage smaller EPID groups is to lower the frequency of revoking (and re-provisioning) entire EPID groups.

In a nutshell, SGX attestation implements a trade-off between user privacy and the cost of revoking entire groups. It seems that several million members is the balance Intel chose between the two.

3.2.2. Active privacy threats

We now wish to focus on the responsibility Intel holds as the governor of every SGX remote attestations in terms of user privacy.

PPID Linkability. As part of a new TCB level provisioning process, the iKGF uses its stored RPKs to generate new PPID value for each SGX part expected to be provisioned. During the remote provisioning process, Intel uses the TCB-specific PPID to authenticate the remote platform before conducting the EPID Join protocol. Although the EPID scheme assures that the issuer does not learn its members' private key, in our case Intel does learn members' pseudonym signature over Intel's basename, as described in Section 2.4.5.

The provisioning-time pseudonym gives Intel the freedom to revoke any platform without relying on a service provider handing over incriminating evidence to justify the revocation. Some publications refer to this type of EPID revocation listing as Issuer-RL, yet it is simply a form of a Sig-RL in which the basename parameter is that of the issuer used during the provisioning protocol.

Users must trust that the link between platform's constant chip ID (RPK) to its present pseudonym is kept confidential and not exploited by the issuer. Exploiting this knowledge can be performed in two ways: (1) Simply challenging a platform with issuer's basename or (2) using Sig-RLs that are one tuple distant, as described in Section 3.1. Sine the latter technique can be performed under a "fully-anonymous" attestation mode, platforms' response to any type of attestation may be directly associated to a specific peace of silicon by Intel.

Collaborating verifiers. By exploiting the low degree of k-anonymity derived from attestation's information disclosure, collaborating verifiers may also break users' expectation of privacy. This can be preformed by a group of service providers, or a corrupt EPID issuer collaborating with one or more service provider.

A common service provider such as a bank or a social media site, will hold many personal information details linked to a specific SGX pseudonym. In the issuer's case, each pseudonym is attached to a specific chip ID, as discussed above.

Platform's quasi-identifiers described in Section 3.2.1 can be used to link different pseudonyms of the same platform with reasonably high certainty. Different SGX pseudonym identifiers can then be coupled along with all user specific details held by the col-

laborating service providers. This breaks users' compartmentalization expectation of pseudonym attestation.

Cases of coupling ambiguity may be resolved actively by the collaborating verifiers by sending consecutive attestation challenges to the same suspected identity.

Lack of deniability. Deniable actions performed anonymously should provide repudiation of origin. In other words, deniability protects a user from the possibility of proving him accountable for a specific action as other users can be found equally accountable for the same action. Although deniability may not always be desired by service providers, examining this property in the SGX remote attestation is valuable for both users and service providers to be aware of.

Naturally, actions linked to a pseudonymous attestation are not deniable as EPID signatures are unforgeable and deterministic. Namely, signing the same basename with the same private key will always result with linkable signatures. However, unlike one might expect of a "fully-anonymous" mode, attestation is still not deniable.

As described in Section 2.4.6, "fully-anonymous" attestations are signed using a random basename generated by the attesting platform. However, SGX attestation always conveys the basename used as it is required for verifying Quotes against Priv-RL. To validate a Quote, its basename is used to generate a new signature for each private key on the Priv-RL. The verifier then asserts that none of the generated signatures are equal to that of the validated Quote.

A verifier who learned a random basename used for a certain incriminating "fully-anonymous" attestation, may challenge a suspected platform with the same basename again. Since no other platform can forge the guilty signature and since EPID signatures are deterministic, the guilty platform cannot deny accountability for his anonymous actions. This kind of attack may be preformed by a service provider registering to the IAS with the incriminating basename, or by Intel itself.

Service provider privacy. Intel's involvement in every remote attestation could also raise some privacy concerns from the service providers' side. Some examples of highly valuable business information can be applications' time of use, or the exact amount of newly registered members for each service. Hence attestation information aggregated by the IAS offers Intel endless valuable insights over all business using SGX.

3.2.3. Privacy summary

Digital privacy has raised great concern in recent years and manufacturers currently invest increasing efforts in distancing themselves from responsibility over customers' privacy. Meeting these expectations calls for privacy-preserving protocols that do not depend on a trusted authority to control or manage users privacy guarantees.

Intel's implementation of EPID hold several privacy issues to be considered by both users and service providers when utilizing SGX services. Intel facilitates and governs all remote attestations hence bears significant responsibility over all SGX relying parties, even well after hardware production.

3.3. Trusting SGX crown jewels

Technologies such as SGX lay substantial responsibilities on its manufacturer. In this section we set to explore the core trust assumptions played in Intel when relying on SGX. We utilize enclaves' two pillar representation from Section 2.2 to exhibit and differentiate the assumptions supporting each pillar. We then continue by analyzing threats and ramifications as a result of different attacker profiles. We assess how these kind of attacks expand the scope of enclaves' TCB boundaries beyond those of the processor itself. Although marking the extent in which SGX guarantees are sound, these trust assumptions received little attention thus far, and may readily stay out of users' notice.

Notably, we do not uncover SGX security implementational bugs or vulnerabilities, but rather provide a close examination of its underlying design choices and dependencies; a crucial inspection for trusting SGX. This section provides another strong motivation for discussing software methods for shrinking enclave's TCB assumptions and overcoming its dependencies on Intel.

3.3.1. Trust assumptions and their implications

We divide the trust assumptions backing SGX into implementational and management assumptions. The first includes the correct and secure desing and functionality of the technology (hardware, firmware and software components). While the second rearguards to Intel's responsibility over the integrity and security measures of RSK and RPK initialization, distribution and usage through the life-cycle of SGX platform. The later (management assumptions) is harder to verify and may easily stay overlooked and unappreciated by relying parties.

Considering our two-pillar interpretation of SGX, users trust the correct functionality of both the enclave isolation mechanism (the first pillar) and the services of the trusted security interface (the second pillar). However, in contrast to the first pillar, which requires only implementational assumptions, the second pillar requires several unfalsifiable assumptions, regarding platforms' initialization and Intel's management. Additionally, compromising each pillar's assumptions requires different means of intervention in order to both violate and exploit the violation. The following discussing aims to mark a line between the two pillars based on their different reliability dependencies, stressing the faithful nature of enclaves' second pillar.

Compromising Pillar I: Isolation. To support enclaves' IEE, the new hardware-enforced mechanisms are trusted to correctly enforce access controls and manage processor context switches from and to enclave-mode. The complementary hardware isolation mechanism is provided by the MEE which is trusted to generate fresh keys for confidentiality and integrity as part of every platform boot.

Since the MEE uses ephemeral keys generated by the platform, enclaves' IEE consists only of implementational assumptions which are theoretically verifiable. For an attacker to break these assumptions, significant active endeavors are required; such as influencing design architecture or intervening on production lines (e.g creating predictable MEE

keys or hidden cache-level access control trap-doors). Moreover, exploiting such IEE back doors requires an attacker to actively compromise a victim platform. Generally, active attacks are noticeable much more than passive ones.

Compromising Pillar II: Security Interface. In contrast to the IEE ephemeral keys, device fuse keys are persist and generated externally by trusted facilities. These are trusted to be completely erased and forgotten (RSK) or securely stored by the manufacturer (RPK). Consumers assume platforms' embedded secret (RSK) exists on their platform alone ,otherwise they don't get confidentiality of "sealed" secrets. While Intel and relying parties assume that no other party has the shared common root key (RPK) for a specific SGX platform, otherwise malicious parties can provide a fake "proof" that they are running code securely on that processor.

Unfortunately, the assumptions that fuse keys are completely erased and forgoten or securely stored by Intel are *unfalsifiable*. Namely, we are unable to evaluate whether these assumptions are true or not. Moni Naor provides a formal treatment to unfalsifiability [42]; In general when we are proving that a system X is secure based on an assumption A, then what we show is that either (i) X is secure or (ii) Assumption A is false. However, the problem with this sort of conditional statement is that it might be hard to demonstrate that A is false, so we dont have good guidelines to decide whether to use the scheme or not.

Recent years have proven that even the most secure of national facilities are not immune to such data breaches. Therefor a massive theft of SGX device keys would not be a unique incident. Some prominent target examples are banking organizations, advanced security industries [8], and even commodity chip manufactures [9]. In the latter incident, billions of embedded unique chip keys were leaked by Advanced Persistent Threats (APT) who successfully penetrated the production facilities. Such incidents had also been known to allow bulk screening surveillance, enabling to later bootstrap targeted attacks against specific users. Note that these are only the *publicly acknowledged* incidents—give their sensitivity, it is likely that many (perhaps most) similar attacks are either undiscovered or were not published.

Even if all platform's security mechanisms are correctly implemented, attacking these second pillar assumptions immediately breaks systems security. Given that SGX is successfully adopted by the industry, APT's are likely to be eager to put their hands in Intel's SGX production lines and online servers. As opposed to the first pillar, breaking these assumptions does not necessitate corrupting manufacturing lines. Violating device fuse key confidentiality may occur well after production and may stay oblivious to manufacturer (as well as to users) for years before being exposed.

In addition to the above differences, the second pillar assumptions also differ from those of the first by the means necessary for exploiting a weakness once an assumption is violated. Knowledge of device root keys may be exploited remotely, without having to gain control of a victim's platform. This also reflects on users' lack of ability to control or detect SGX platform reliability state.

Legal aspects regarding product information disclosure is another rising concern of recent years. Like any other commercial company, Intel is subject to legal subpoenas. In

such a case it would not be the first time civil liberties and the rights for digital privacy are on debate.

To avoid being confronted with such requirements and to refrain from holding responsibility for all SGX sealed data, Intel promises to forget all RSKs produced. Hence, customers are asked to make significant unfalsifiable assumptions about Intel's integrity. Additionally, as RSKs have no future use by Intel, they are handled and cleared in (what Intel states as) "high-volume manufacturing lines" that do not enjoy the security measures guaranteed by the iKGF (in charge of producing and holding RPKs). This makes it even harder to assume security breaches to manufacturing infrastructure will never occur.

3.3.2. Pillar II violation ramifications

After emphasizing the faith-based nature of enclaves' second pillar, we now discuss potential ramifications of its violation. We dived the implications of different cases in witch the confidentiality of each of the fuse keys is compromised, and the case of both RSK and RPK are compromised.

Attacker profile / Attacker knowledge	**Remote** (Without compromising victim's device)		**Local** (Compromising victim's device)	
	Passive (Eavesdropping platforms' network traffic)	Active (Challenging the platform or impersonating it against a 3rd party)	Passive (Eavesdropping memory on the platform)	Active (Manipulating platform execution)
RSK	Decrypt sealed data put on untrusted channels or storage. (with Owner Epoch minor struggle)	Same as passive	Decrypt sealed data on victim's platform. (Extracting Owner Epoch from flash)	Decrypt attestation key: → Emulate enclave IEE. → Impersonate the platform.
RPK	None	Authenticate against Intel as a genuine SGX device (Emulate enclave IEE)	None	Same as remote active
Both fuse keys	All above	All above + impersonate an existing SGX platform.	All above	All

Figure 3.1.: Potential attacks according to attackers profile and the knowledge he is armed with.

Root Sealing Key compromise implications. One could argue that an answer to RSK confidentiality concerns is provided by SGX's option to configure the Owner Epoch

value. As illustrated in figure 2.3, both RSK and Owner Epoch secrets are used to derive all EGETKEY products. Hence users are able to influence SGX derived keys generated on their platform by adding another degree key entropy, chosen independently of Intel.

However, Owner Epoch is prone to all weaknesses commonly associated with user passwords. Repeated usage and easily rememberable phrases are just some of these common vulnerabilities. These often cause passwords to have low degree of entropy or make it possible to deduce them from elsewhere. Additionally, in practice users are not always aware of such advanced security features and rarely configure them. Moreover, two of the first Original Equipment Manufacturers (OEMs) to deliver SGX supporting BIOSs did not include an option that enables users to configure platform's Owner Epoch value. Instead, a default value is configured by the OEM during platform manufacturing, adding more concern to the Owner Epoch inadequacy.

Against an attacker armed with RSK knowledge, sealed data is protected merely by a password. This threat is not imposed solely by local active attackers who may compromise user's platform. Applications building on SGX sealing guarantees to transmit sensitive data over untrusted channels are exposed to threats imposed by remote passive attackers. Such passive attackers could eavesdrop transmissions of sealed data and decrypt the content with relative ease. In the local active attacker case, the job may be even easier. Since the Owner Epoch is stored on the platform's non-volatile flash memory, local attackers operating on privileged system software may be able to read the Owner Epoch value from the flash chip. This allows the attacker bypass the struggle of cracking the Owner Epoch value.

In addition to producing RSK-based sealing keys, RSK knowledge allows an attacker to generate fake local attestations and potentially even forge Quotes. As described in Section 2.4.4 local attestation reports are signed by the processor using a symmetric key available through EGETKEY only by the destined attestation enclave (the verifier). Since the only secret parts in producing these EGETKEY products are RSK and Owner Epoch, a local active attacker is able to sign any local attestation report on a specific platform. Even more alarming, is the ability to produce remotely verifiable attestation Quotes. The QE uses local attestation to verify a requesting enclave running on the platform and creates Quotes based on EREPORT products. An attacker can attest to the QE using a forged local attestation report and retrieve a "genuine" Quote from signed by the QE using platform's valid attestation key. Alternatively, such an active attacker is also able to simply decrypt platform's attestation key by using RSK to create the appropriate provisioning seal key. The attestation key can then be used to forge Quotes even out of the victim platform.

The most troubling implication of forging Quotes is attackers ability to convince remote parties that sending their sensitive code and data to the him will stay protected under SGX security guarantees. This kind of attack completely breaks the reliability of enclaves.

Root Provisioning Key compromise implications. Like the RSK, the RPK can also be exploited outside of its originating platform. Knowledge of the RPK (together with precise understanding of SGX key derivation and PvE logic) is enough to authenti-

cate as a genuine SGX device and receive a fresh attestation key from Intel's provisioning service. Unlike RSK, RPK is used solely during the provisioning phase to demonstrate the processor's authenticity to Intel's provisioning servers.

RPKs alone are only useful for an attacker if they have not yet been provisioned. In other words, RPK can be used to conduct a setup provisioning process, but not re-provisioning (attestation key retrieval or TCB update). Re-provisioning requires decrypting backed up attestation keys previously provisioned to that specific RPK in order to prove the platform has never been revoked in the past. This is normally performed by the PvE using platform's RSK with the appropriate SVN requirement for each TCB key. Unlike RSK, holding a RPK copy of a specific chip instance does not necessarily threaten that specific platform. Therefor, despite iKGF's security guarantees protecting RPKs', it may be less attractive to gather RPKs in large quantities (in contrast to RSK).

The RPK in its raw form is not the only asset allowing to maliciously obtain SGX attestation keys. RPK derivatives and TCB specific values, such as PPID and random TCB challenges (presented in figure 2.6) are also sufficient for preforming a credible provisioning process. This approach may be cost-effective in comparison to stealing RPKs. Breaking into online servers is should be an easier target of compromise than breaking into the iKGF production lines. Yet the disadvantage of such method is that RPK derivatives are TCB specific. Since the SGX key recovery transformation refers to lower SVN keys as products of more OWF transformations, these assets may become obsolete as SVNs become outdated. Hence this method requires repeated access to Intel's servers periodically according to TCB updates released, in contrast to a one-time security breach endeavor for obtaining a bunch of original RPKs.

Compromising Both device keys. The main advantage of knowing both RSK and RPK of the same platform is the ability to retrieve previously provisioned attestation keys. RPK is used to authenticate against the IAS and RSK is used to decrypt the previous provisioned attestation key backed on Intel's servers. Retrieving attestation keys allows an attacker to impersonate an existing SGX platform. This opens a new optional threat of authenticating against a service providers that had previously served the victim (e.g. victim's bank).

3.3.3. Concluding trust assumptions implications

Vulnerabilities are in the nature of every technology life cycle, especially new and complex as the one in hand. To deal with such cases SGX has a TCB recovery mechanism incorporated into its architecture. This enables to patch platforms when implementational security vulnerabilities are revealed. As described in Chapter 1, Intel chose not to implement an endorsement key scheme as proposed by past TCG specifications for the TPM remote attestation standards [10]. Instead, SGX makes use of replaceable attestation keys which are decoupled from platform's persistent fused keys. This allows for both attestation key revocation and re-provisioning to be remotely managed by Intel.

However the attacks discussed in this section are products of SGX design choices. Namely, these threats cannot be thwarted by security updates. A complex eco-system, such as the provisioning and remote attestation infrastructure, inherently expands the

attack surface of the technology. Although revocation capabilities are built in as part of the SGX attestation mechanism, it is not clear how (and if) Intel plans on identifying and accusing a suspected remote platform as compromised. Especially in cases where compromised keys are used passively by the attacker as discussed above.

Since fuse key confidentiality assumptions define the scope under which SGX is viable, enclaves' TCB expand out of the trusted boundaries of the processor. Their TCB scope includes the security measures of Intel's online server, manufacturing lines, legal obligations regarding production information disclosure, et cetera.

Software APT attacks are a primary motivation for integrating SGX into applications' security designs. Yet, in present reality it seems evident that APTs will invest extensive efforts in pursuing enclaves' crown jewels, i.e SGX fuse keys. It is especially common for certain organizations not to trust foreign hardware manufacturers. In particular, organizations handling sensitive information or critical operations, such as governmental, financial or health institutes. These potential applications cannot afford to have their security boundaries extend out of their supervision. In particular when considering the unfalsifiable nature of enclaves' security assumptions discussed. In such cases, SGX loses its ability to assure security against its ultimate adversary.

Table 3.1 summarizes the potential ramifications of violating enclaves' second pillar trust assumptions discussed in this section. Potential attacks are presented according to attackers profile and the knowledge he is armed with.

4. Enclave Emancipation

In this section we propose new techniques for creating alternative attestation and sealing primitives that unshackle users of current enclave limitations and faith-based trust assumptions. Unlike the fixed "trust Intel's servers" assumption, our new protocols allow users to choose where they place their trust and offer different capabilities and guarantees when using sealing and attestation.

We start by proposing several independent attestation primitives that build on a trusted sealing primitive. We then change the underlying trust assumption, and show how an independent sealing primitive can be established given a trusted attestation primitive. Finally, we propose using a timing-based scheme to establish a completely independent enclave that uses only alternative attestation and sealing primitives.

4.1. Independent Attestation

Current SGX attestations depends on the availability of an SGX service provider, which in turn depends on the availability of the IAS. Here we propose several methods for independent remote attestations. We divide our methods according to their IAS-dependency, as summarized in Table 4.1. In addition to overcoming availability limitations, our proposals enable users to choose different trusted attestation authorities as well as enhance limited privacy guarantees of present SGX attestation.

Both the online and Antarctica models in this section depend on a trusted sealing primitive. Namely, relying on RSK confidentiality. This assumption seems to be a

Attestation properties:
Availability - Depends on IAS for each attestation?
Privacy - Enables enhanced privacy?
Trust - Relies on a trusted service provider?

Alternative Attestation Proposals:	IAS-dependent provisioning (The online model)			IAS-independent provisioning (The Antarctica mode)
	1	2	3	4
Availability	NO	YES	NO	NO
Privacy	YES	YES	YES	YES
Trust	YES	NO	NO	YES

Table 4.1.: Alternative attestation sum-up

weaker than depending on RPK's confidentiality, since RPKs are stored by Intel (and its derivatives are transferred and held on Intel's online servers) in contrast to RSKs that never leave the iKGF and are assumed to be forgotten by Intel.

4.1.1. IAS-dependent Provisioning - The Online Model

We first provide three attestation methods for the "online model" in which one-time IAS-dependent provisioning process is leveraged to enable multiple future independent attestations. These proposals utilize current SGX services together with dedicated service provider to solve some of present attestation trust, availability and privacy limitations.

The first method builds on a *trusted* service provider and allows future remote attestations which are IAS-independent. The second allows to loose the trust played on the service provider, but depends on the IAS-availability for attestation. The third proposal uses proof of knowledge to enable attestations which are both IAS-independent and do not depend on a trusted service provider. All three methods provide enhanced privacy to attesting platform

4.1.1.1. Trusted service provider

To lose Intel availability dependency we introduce a Certificate Authority (CA) mediator between Intel and the attesting SGX platform. The CA is in charge of provisioning alternative attestation keys which are independently verifiable by any external party. In this method the CA is considered trusted by relying parties. Different service providers may implement the following CA role, and a single platform may be provisioned many times by different CAs. This allows the platform to later attest to verifiers that only respect certain CAs.

The CA supports a predefined set of enclaves, some function as Alternative Provisioning Enclaves (AltPvE) and others as Alternative Quoting Enclaves (AltQE). Users who wish to be certified by a certain CA service provider, will run the corresponding AltPvE. The AltPvE conducts a standard SGX remote attestation with the CA. On receiving a Quote, the service provider requests verification from the IAS and validates that the AltPvE MRENCLAVE stated in the Quote is supported. If successful, it then provisions the AltPvE with an attestation key. Any group scheme can be implemented and managed by the CA to provide the appropriate attestation privacy (including an EPID scheme, as used in present attestation scheme). We consider certifying a platform as adding it to CA's group and provisioning it with a membership credential.

As in the original PvE design, the membership credential is sealed by the AltPvE under a MRSIGNER policy, hence allowing the AltQE to later decrypt for attestation. This enables the CA to deliver different AltQEs without the need to contact the IAS and update the group management databased. However, since MRSIGNER is used to share the attestation key, any enclave developed by the AltPvE developer can obtain it, hence both AltPvE and AltQE must be developed by the trusted CA. Accordingly, CA's will only service enclaves developed by themselves, otherwise the CA loses control over the exact code that can obtain and handle the attestation keys it provisions.

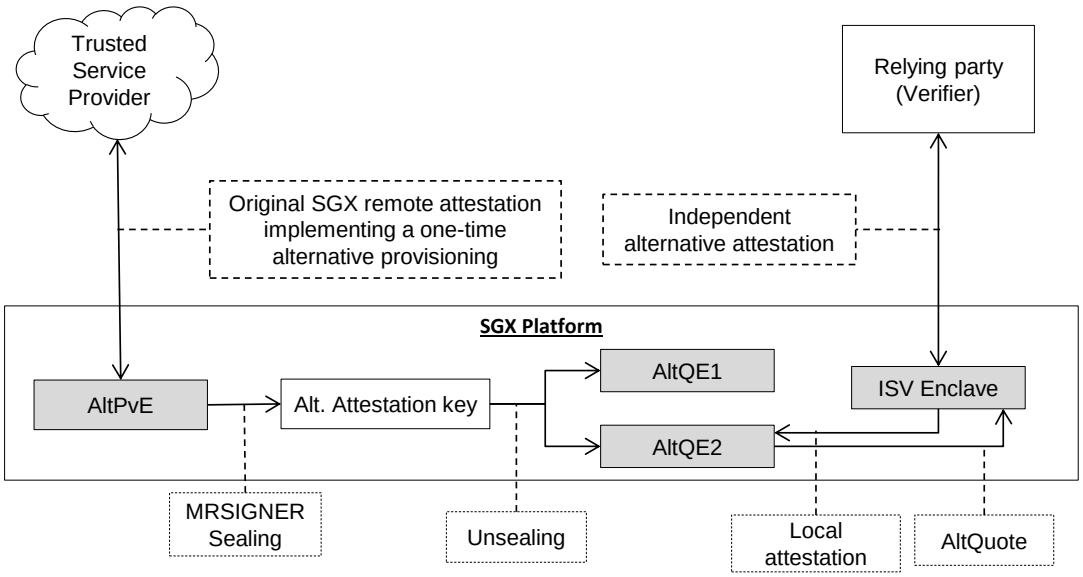

Figure 4.1.: Alternative attestation based on a trusted service provider.

The AltQE will serve ISV enclaves running locally on the same platform with Alternative Quotes (AltQuote) signed with its attestation key. To receive an AltQuote, an ISV enclave conducts a local attestation (which does not require IAS availability) against the AltQE. The AltQuote may include any subset of information stated in the local attestation report about the attesting enclave, and any auxiliary data added by the AltQE. Relying parties can then verify AltQuotes independently of the IAS or CA's availability.

The same attestation key may be used by different AltQE versions of the same CA, each providing different assertion information. This allows for ISV enclave's to independently manage their anonymity level by requesting an AltQuote from different AltQEs. Alternatively, the AltQE can receive an anonymity descriptor as part of its AltQuote requests and only sign the requested enclave properties.

4.1.1.2. Untrusted service provider with IAS dependency

As described in 2.4.6, only service providers that meet Intel's requirements for IAS registration can verify Quotes. This is enforced using a service provider certificate provisioned by Intel during IAS registration. Therefore service providers in present SGX attestation mechanism are trusted to hold an exclusive role in validating enclaves, acting accordingly and optionally forwarding the correct ISA result.

In this method we wish to keep the IAS availability dependency, while removing the trust and availability dependencies on the service provider. This allows two parties to attest each other without trusting a service provider mediator. To a chive this, we suggest a dedicated *untrusted* service provider that delegates its verification process to

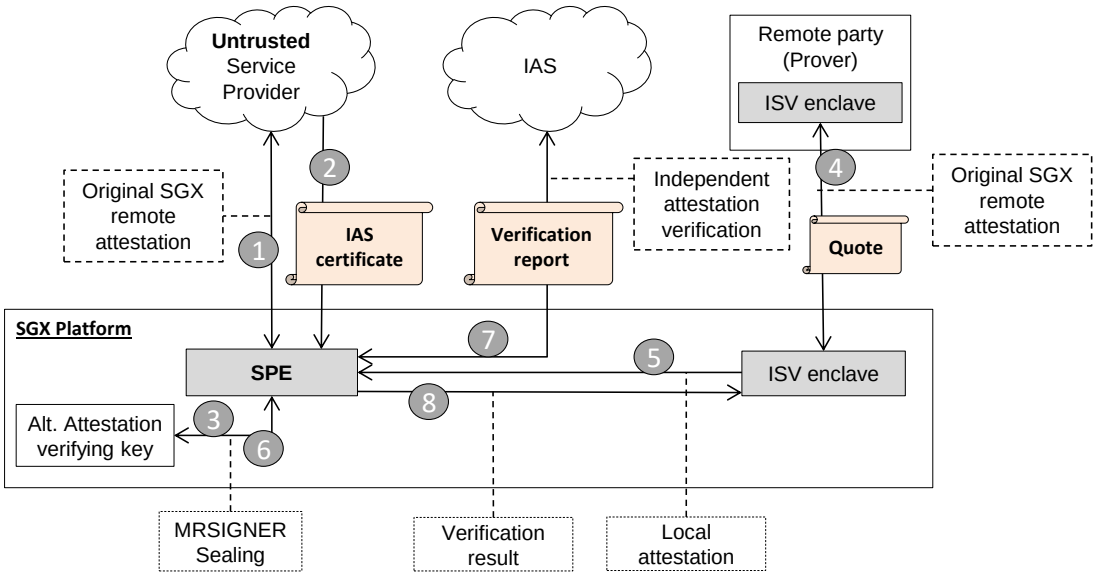

Figure 4.2.: Alternative attestation based on an untrusted service provider.

a local Service Provider Enclave (SPE) running on an SGX verifying party.

To setup a client platform, the platform contacts the service provider and downloads its SPE. The platform then runs the SPE locally and preforms a standard remote attestation with the service provider. If successfully verified the service provider can securely send a copy of its IAS certificate to the SPE. The certificate can then be sealed by the SPE under an MRENCLAVE policy and stored locally for future use.

Fig. 4.2 illustrates both the setup phase (steps 1-3) and an independent attestation following it (steps 4-8). After the setup phase, the SPE can serve any ISV enclave running locally on the platform with Quote verification services in the following manner:

4. The platform receives a Quote created by a remote enclave.
5. The Quote is forwarded to platform's SPE.
6. The SPE unseals its IAS certificate.
7. The SPE directly contacts the IAS and sends the Quote for verification.
8. The SPE validates the IAS verification report and outputs accordingly.

To enhance prover's privacy and prevent the verifier from learning all information stated in the Quote, the SPE can output only partial properties of the attesting enclave Quote, as described in the AltQE case. The SPE code should be published to all relying parties, and only trusted accordingly (to ensure it validates Quote correctly, only discloses desired identity properties, does not leak attestation secrets, etc). A good approach for

this would be to publicly publish SPE's source code. ISV enclaves can then be developed to accept a known set of MRENCLAVEs which are that are widely accepted and trusted.

Since the PSE is needed to hold service provider's IAS certificate, this method can only be used by verifiers running on SGX hardware. The next technique keeps the use of an untrusted service provider, but provides an attestation evidence can be verified by any external party without depending on IAS availability

4.1.1.3. Untrusted service provider without IAS dependency

In this method we utilize a dedicated *untrusted* service provider acting as a CA. The CA is used to conduct a provisioning process that results with an alternative attestation key that is certified directly by Intel and hence can be verified independently by any party knowing Intel's (ISA's) public key. In order to remove the trust played in the CA we mark two of its properties to be treated: (1) Its ability to develop new enclaves that can obtain platforms' private attestation keys. (2) Its role in verifying platforms and managing attestation key provisioning.

To tackle these challenges we (1) combine the PvE and QE functionality into one enclave, hence eliminating the trust played in its MRSIGNER, namely its developer and sealing identity. (2) Utilize the IAS verification report for provisioning alternative attestation keys.

Sharing the same attestation key between two enclaves, requires sealing the key under a MRSIGNER policy, as is the case in the previous method (and in the current SGX remote attestation) mechanism. This necessarily introduces trust on enclave's developer, the MRSIGNER, for supporting the attestation scheme. This is the case since any enclave signed by the MRSIGNER can access the provisioned attestation key on a specific platform.

To eliminate the trust dependency put in the enclave developer, the same enclave should facilitate both the provisioning and the quoting services. A MRENCLAVE policy can then be used to seal the attestation key and bind it uniquely to that specific enclave only. This ensures that no unexpected code can access the key in the future. This method utilizes one enclave, an Alternative Provisioning and Quoting enclave (AltPQ). Its code should be published to all relying parties, and only trusted accordingly.

In order to completely remove the trust put in the service provider, users can no longer depend on it to function correctly when validating and provisioning platforms. Since only IAS registered parties can verify Quotes, we wish to use service providers' access to the IAS verification API and validate the result within the trusted AltPQ enclave running on the provisioned platform. As described in Section 2.4.6, the verification report result sent back from the IAS is signed by Intel and includes the Quote requested. Therefore, the AltPQ (running on the provisioned platform) can use Intel's public key to verify the correct functionality of the service provider.

To provision a platfom with an alternative attestation key, an AltPQ instance is run locally the client platform and contacts the dedicated service provider. The AltPQ generates a fresh key-pair and includes the public key in the additional data section of the Quote sent to the service provider. If the IAS verification report is positive and valid,

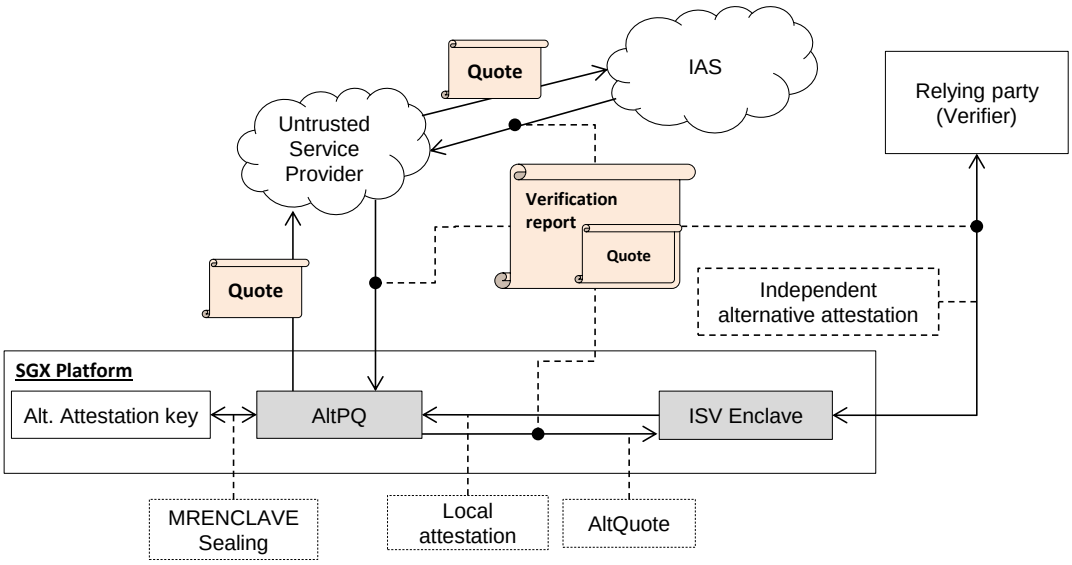

Figure 4.3.: Alternative attestation based on an untrusted service provider without IAS availability

it then serves as a witness proving the trustworthiness of the public key attached to the Quote. The corresponding secret key can now be used as an effective alternative remote attestation key. To finalize the provisioning phase, the AltPQ seals its attestation key using an MRENCLAVE policy, and stores it locally together with its IAS verification report that facilitates as its public key certificate.

After being provisioned, the AltPQ can serve any ISV enclave running on the platform with AltQuotes in the same manner as did the AltQE. The IAS verification report (the attestation certificate) is sent together with the AltQuote so that any party knowing Intel's pubic key can validate the AltQuote independently. However, this technique cannot be leveraged to implement anonymous attestations since the attestation certificate is platform unique and must be revealed for relying parties to verify the AltQuotes. To add attestation privacy to this scheme, we propose a zero-knowledge proof is used by the AltPQ.

4.1.1.4. Anonymous AltQuotes

With every attestation preformed, the provisioned attestation key (Sk) can be used by the enclave to generate a new ephemeral key-pair (Pk',Sk'). The enclave can then generate a zero-knowledge proof of knowledge of an attestation key (Sk), who's corresponding public key (Pk) is included in an attestation certificate and Sk was used to generate the ephemeral public key Pk'. This proof can then be delivered together with an AltQuote signed by Sk'. Additionally, the generated key-pair (Pk',Sk') may be stored for future reuse, enabling pseudonymous attestations.

Such a zero-knowledge attestation scheme adds considerable computation complexity

for each attestation preformed. We thus propose to delegate the proof of attestation certificate to an AltQuote Verification Service enclave (AltQVS). The AltQVS, how's privacy is not required, will replaces the function of a zero-knowledge proof.

The AltQVS runing on the verifier platform, will publish an AltQuote holding its public key together with an appropriate attestation certificate (signed by Intel). The AltPQ runing on an attesting platform, creates a new AltQuote, encrypts its using AltQVS's public key and sends it to the AltQVS. The AltQVS will then validate the AltQuote and signs a new anonymized-AltQuote (that discloses only a desired subset of properties of the original AltQuote). The anonymized-AltQuote can then be verified by any party knowing both Intel and AltQVS's public keys.

The AltQVS may run on a verifying SGX platform, or on a dedicated service provider equipment with SGX. In the later case, the service provider implements the CA role played by the service provider in the first method described above Section 4.1.1.1. Only this time, the service provider does not need to be trusted as its sensitive functionality is implemented within the AltQVS. Since the AltQVS's code is publicly available, relying parties are assured as to the exact logic used by the CA to provision platforms.

Interestingly, simple group schemes that normally allow signature opening by the group manager, may be used the CA to guarantee provisioned platform anonymity against the CA. This can be achieved by omitting the Opening procedure from AltQVS's implementation. AltQVS's management key should be sealed under the MRENCLAVE policy to assure it is used only by a known limited amount of procedures implemented by the AltQVS.

4.1.2. IAS-independent Provisioning - The Antarctica Model

The last alternative attestation approach allows to completely omit any IAS dependency. This method spares the need to ever contact Intel's servers, as if all SGX platforms arrive on Antarctica with no internet connection (hence the "Antarctica model"). This approach can be utilized in isolated scenarios, such as enterprise intranets, by organizations that wish to reduce the trust played in foreign manufacturers, or simply by users whom do not wish to *faithfully* trust the IAS security and integrity.

This method completely loses the trust put on Intel's management of RPKs. Like in the IAS-dependent methods, here we still rely on a trusted sealing primitive, namely on the RSK, and on a trusted CA. In contrast to the former methods, the trusted CA authority does not function as a provisioning middleman between the platform and the IAS.

For the CA to replace Intel's role in authenticating SGX platform, it has to make several new assumptions and limitations. Since this approach does not rely on Intel's servers, CA's are not asked to assume that security breaches to Intel's servers will not occur, nor do they need to depend on its availability or integrity for provisioning. Instead, CA's make the assumption of an SGX platform "clean setup phase". This includes trusting Intel's iKGF and the supply chain delivering the platform to the CA. Crucially no assumption is made by the CA regarding platforms' embedded RPK. Finally, this method is limited to cases in which the CA has physical access to the platform during

its"clean setup phase".

This method accounts for CA's which are generally responsible for platform's first hardware and software setup stages as part of customers' supply chain. Common candidates for such constraints are platforms' OEMs or corporate IT departments. In these cases the CA trusts the platform has as a genuine SGX processor and is thus willing to provision it with a respective attestation key.

As part of platform's setup process, the CA runs an AltPvE with a CA secret sining key as input. The AltPvE creates a provisioning certificate by generating a new key pair, creating a local attestation report holding its public key and signing it with CA's secret key. AltPvE's secret key is then sealed under a MRSIGNER policy for future use as a remote attestation key.

After the setup process, a provisioned platform can run CA's AltQE that unseals platform's attestation key and signs AltQuotes. AltQuotes can then be sent together platform's provisioning certificate to any verifying party knowing CA's public key.

Since SGX does not offer trusted input-output paths from and to enclaves without depending on attestation, the AltPvE may be threatened by malicious software interrupting communication between it and its operator (the CA). Hence as part of platform's setup, the CA should first load a trusted OS for running the AltPvE before installing

4.2. Independent sealing

In this section we set to establish an independent sealing primitive given trusted attestation. This scheme can be useful for users who wish to decouple the security of sealed data from the security of platform's fuse keys. Here we drop the assumption that Intel forgets (and does not reveal) platform's RSK. Additionally, our proposal adds a new security feature to enclaves' sealing, which enables users to remotely revoke the ability to unseal data.

4.2.1. Distributed Sealing Key

We propose an independent sealing method that utilizes a secret sharing scheme implemented by an enclave on the platform. We follow a basic secret sharing scheme that includes protocols to split, share and reconstruct a secret. A simple xor-based split and reconstruction can be sufficient.

A platform Sealing Service Enclave (SSE) internally generates a Distributed Sealing Key (DSK) and implements the following two protocols to share the DSK secret with n remote parties:

1. Setup - Generate, split and share a fresh random DSK.
2. Init - Reconstruct platform's DSK by obtaining their shares of n shareholders.

Protocol notes

Setup:

- Generate a random secret (DSK) and a random key-pair (Sk,Pk).
- Split to n shares and seal each share with RSK.
- Distribute shares and Pk to n remote parties.
- Seal dealer's share and Sk under RSK and store it locally on the platform.

Init:

- Preformed once every boot.
- Attest to each remote party to receive its share.
- Unseal share (including dealer's share from local storage) and construct DSK from all shares.

This method ensures the DSK in its full form never leaves enclave's IEE and is only present during runtime as only the SSE can obtain and reconstruct all shares. Hence this ensures that the DSK is never exposed on any environment other than its originating enclave.

Remote parties can be realized by a variety of sources such as a web-service or portable smart devices. As no individual share has use on its own, an attacker must compromise all shareholders to obtain a DSK-sealed secret. Redundancy can be added by using a (t,n)–threshold scheme (such as Shamirs Secret Sharing [50]).

After constructing DSK, the SSE can then provide alternative sealing services to local enclaves on the platform. The SSE exposes an interface of two functions: independent sealing (iSeal) and independent unsealing (iUnseal). Both functions receive a tuple of the data to be sealed/unsealed and a sealing policy to follow. Requests are past using local attestation and the SSE uses the MRENCLAVE or MRSIGNER value from the local report according to the policy requested. This value together with DSK are used to produce a request specific sealing key. ISV enclaves should first seal their data based on the original RSK before requesting the SSE service. Since iSeal and iUnseal follow the sealing policy requested, SGX unique sealing properties of forward secret migration and sealing policy are preserved.

However, DSK does come with a cost as it cannot be implemented by standalone platforms. Since DSK relies on external assistance, it requires contacting at least one external party once every boot cycle. This caveat is inherent in our initial goal to isolate user's secrets from the platform's underlying infrastructure. As a secret must reside elsewhere, no method aiming to secure persistently stored secrets against an active attacker can circumvent this drawback.

Additionally, this method can enable users to remotely revoke the ability to unseal data by contacting share holder. This kind of protection can be valuable in cases where the platform is suspected to be compromised or stolen.

4.3. Independent Enclave

Previous sections leveraged a trusted sealing primitive to implement an independent attestation primitive and vice versa. Attestations protocols discussed thus far (both conventional SGX and out alternative ones) rely on a shared secret between the prover and the verifier. Every SGX chip holds a hardware-based secret (the RPK) shared with the iKGF which is used to conduct attestation key provisioning. In turn, the attestation key is also secured by the SGX hardware (specifically the RSK) for future remote attestations. Any hardware-based attestation schemes must assume the confidentiality of the shred secret installed within each platform and held by the prover (e.g. Intel) remains uncompromised. Therefore proofs of knowledge techniques alone are inadequate witnesses in our attempt to achieve a completely independent attestation primitive.

In this section we propose using a different premise for establishing trust, we suggest that the strong assumptions backing SGX embedded secrets may be relaxed at the expense of others, which do not depend on Intel's integrity and security. We propose a time-based approach is used to construct an alternative attestation that can then be leveraged for our independent sealing primitive, thus constructing both enclave primitives without relying the original ones. This completely losses enclave's dependency on the strong device key confidentiality assumption.

Before discussing the implementation of software-based attestation on SGX, we briefly present software-based attestation fundamentals and prior works. In contrast to former proposals, the time-based approach makes a considerable theoretical leap, proposing an idea that might not be implementable on present hardware. Hence we provide a formal treatment to help extract the exact properties required to realize this approach and bridge the theoretical leap. We also propose some candidates that may be implemented by Intel to facilitate this approach. Finally, we show how software-based attestation can be bootstrapped to enable our independent sealing method presented in Section 4.2.

4.3.1. Software-based attestation.

In a software-based attestation scheme the adversary has access to all information available to protocol parties. In particular, this approach does not assume any secret is held by the attesting party. Instead, this approach leverages side-channel witness, typically the time it takes for the prover to compute a response.

Software attestation employs the common approach of challenge-response protocols, where the verifier challenges a potentially compromised prover with respect to expected memory (code and data) content. The prover runs an attestation function that aims to provide the verifier with a proof witnessing the correct execution of the expected program. The attestation function is run on a random challenge received from the verifier to produce a verifiable response within a given time interval. The time interval accommodates a predetermined threshold standing in between the maximum computation time of an honest case and the minimal computation time of any software trying to simulate the attestation function on the same hardware.

This approach may resemble proofs of work techniques, however proofs of work in

general are not suitable for software attestation since they are usually less efficient and not designed to achieve the optimality requirements as desired by the software attestation scheme.

Prior works. A large body of literature is dedicated for designing and analyzing software-based schemes — see, e.g. [13, 19, 51, 52, 20, 25, 26, 29, 30, 46, 36, 24, 38, 48, 47, 49, 11]. Many prior works implement this approach on a variety of computing platforms, from general propose modern Intel processors [38, 48] and other complex environment processors such as smartphones [36], to those of embedded micro-controllers [20, 19]. Some of which provide empiric results showing the use of this technique in different scenarios of hardware and network configurations.

Attestation function general structure. The attestation function is designed to be the optimal implementation for computing the its result on a given processor. Therefore producing a proper response within an expected time is a trustworthy witness ensuring both the integrity of the code producing the result and that the execution was not emulated. Namely, the exact code ran directly on the expected hardware.

The attestation function uses a computational intensive logic that is tailor-made to exploit a specific processor execution throughput properties and other microarchitectural behaviors. Most software-based attestation proposed use self-checksumming function which computes a checksum over the entire attestation function instruction sequence. The checksum is specifically crafted such that any modification to the code necessarily incurs a timing overhead. Using a different random challenge as the basis for every checksum computation ensures the adversary is limited to produce the expected result within the bounded time interval.

A substantial condition for attestation function optimality is that all its code fits within processors Level 1 cache (L1) and that fetching and writing to this cache has distinguishable low-latency relative to higher memory levels on the system. This condition assures that any alternative implementation will have to fetch its code and data from higher intermediate cache memories, inevitably resulting in some pipeline queuing latency compared to the honest case.

Once a marginal time advantage is achieved, it is then amplified intensively enough to ensure a large enough time gap to compensate for any honest case delay that can be advantaged an adversary (such as propagation latency of sending the proof to the verifier). Most works, use an iteration parameter that determines the number of checksum loops that is required for computing the proof. The number of iterations is chosen according to the link latency between the two parties and according to the marginal advantage of a the optimized checksum code. To promise an expected time witness, current schemes are not designed to be performed across different networks. Commonly, the two parties are restricted to be within the same Local Area Network (LAN).

After the checksum result is sent to the verifier, the attestation function measures a hash over an arbitrary executable payload. The payload runs immediately after the attestation function is completed. After the verifier verifies the checksum time restriction and value (witnessing the integrity of the attestation function), he then verifies the identity of the expected payload.

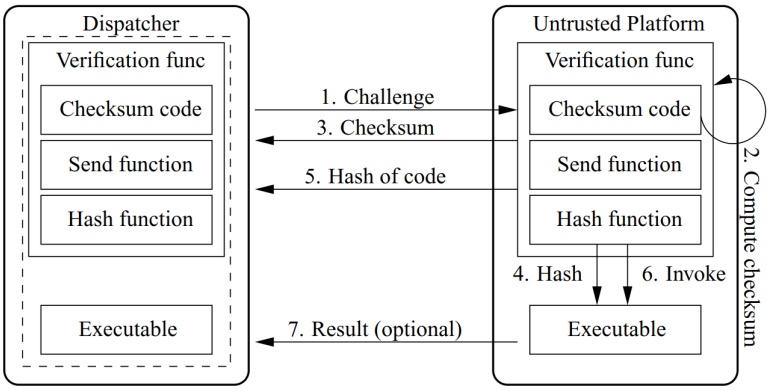

Figure 4.4.: Pioneer's attestation function design [48].

To ensure the adversary cannot subvert the execution flow and run a malicious payload instead of the attested one, the attestation function is assumed to execute in an isolated environment, out of adversary's reach. Current schemes run the attestation function in system's highest privilege level (OS kernel, or even a SMM module) and temporarily manipulate interrupt handlers to assure no software can manipulate execution transition to the expected payload.

Attestation function conditions. For the attestation function to guarantee the expected code ran directly on the expected hardware, the following conditions must hold:

1. The exact model and configuration of prover's processor is known to the verifier.
2. The attestation function consists of an optimal implementation of the checksum logic.
3. The attestation function executes in an isolated environment, out of adversary's reach.
4. L1 cache of the processor can fit all the implementation of the attestation function code, and L1 has a distinguishably lower latency relative to higher memory levels on the system.

Crucial, we note that the first term implies this approach provides security only against pure-software attacks. However, software-based attacks are mostly much easier to launch than hardware-based ones. Therefore conditioning the prover platform is untampered can be a reasonable assumption for a variety of scenarios.

Completeness and Soundness. In essence, the correctness of this approach follows from the fact that a honest prover always succeeds in computing the correct response within the expected time interval. The security to this approach promises that the probability of an adversary to produce a correct response using any code different of the expected is negligible.

4.3.2. "Enclavazing" software-attestation

Here we trade the assumption of SGX device keys confidentially, with a weaker adversarial model that excludes hardware threats. Despite the fact that software-based attestation has been found applicable for a variety of processors, reasoning such an approach for the SGX case is not straightforward. Enclaves hold some caveats as well as some significant advantages for software-attestation.

We first discuses the limitations of utilizing current software attestation designs for enclave attestation. Then we reason about the possibility of realizing a new software-attestation implementation for SGX. After a formal treatment of our proposal, we explain how enclaves satisfy the basic software attestation conditions describe above, and show how enclave specific properties can be leveraged to achieve greater functionality with less assumptions, in comparison to previous software-based designs.

Using existing frameworks. Simply adapting an attestation function to suit the SGX processors does not satisfy enclave attestation needs. Present schemes assure that a specific code runs directly on the expected hardware, but do not provide any assertion as to the execution mode of the processor. Therefore present attestation function designs do not assure the verifier that the attested enclave was properly initialized and executed within a genuine IEE.

One solution could be to leverage current designs as a building block for ensuring proper enclave execution. In this approach, the attestation function is executed out of an enclave as an additional step of platform's initial boot up process. Before loading the OS, the attestation function is ran and attests the execution of a payload that properly initializes an enclave. The enclave can then generate a key-pair and establish a secure channel with the verifier. After the OS is loaded, the verifier can communicate securely to the attested boot time-enclave. This way a new root of trust is established once for very boot cycle without relying on platform's hardware security features. The verified enclave can then be used as platform's AltQE, as described in Section 4.1.1.1.

However, this approach might be limited to operating systems that are customized to load without destroying the boot-time enclave. Hence, implementing a new software-based verification function, customized for running within an enclave will provide a much stronger and practical tool.

Leveraging SGX unique properties. Here we wish to run a software-attestation scheme within an enclave. To motivate this approach, we provide three aspects in winch enclave software-attestation deviates from former schemes. These highlight the advantages of running software-based attestation function within the IEE and the challenges of implementing such a function.

Attestation function isolation Current schemes satisfy the isolation requirement by running the attestation function in system's highest privilege level and temporarily manipulate interrupt handlers. A proper enclave enjoys the SGX IEE protection, hence ruining the attestation function within an enclave satisfies the isolation requirement on user-mode, meaning no special OS support is required to enable enclave software

attestation. In this sense, *enclaves may be an ideal vessel for performing software-based attestations.*

However, since enclave's isolation does not provide any guarantee as to execution continuity. This means the OS can take control over attestation flow at any time. This is a unique challenge of our approach, that was not confronted in former schemes.

Payload identity and integrity The common approach in present software-attestation literature for binding the payload measurement uses a hash function as part of the attested attestation function code. After sending the attestation response, the prover measures the payload and sends the result to the verifier. Since no interrupts are expected, a correct response also assures the verifier as to the exact code ran to produce the second message (the payload measurement). Since enclaves do not provide any guarantee as to execution continuity, the adversary (e.g. a malicious OS) can halt enclave's execution after the attestation response is sent and send a rouge payload measurement.

Generating an ephemeral key-pair and adding its verification key to the checksum computation is not enough to guarantee the authenticity of the second message (signed by the corresponding signing key). Since only the second message declares the identity of the payload, a malicious enclave may leak the verification key after running a correct attestation function, hence allowing code out of the IEE to send a signed rouge payload measurement.

To deal with attestation function interrupts, we can leverage the MRENCLAVE value precomputed during enclave initialization. Instead of using a hash function as the final step of the attestation function, we add a prefix code to the verification function that simply calls for enclave's MRENCLAVE value. Together with the ephemeral verification key, both values are included as the basis for the attestation function checksum. This binds the attested payload identity to the first response message sent by the prover (the attestation proof).

Interrupting the prefix code cannot be leveraged to attack the attestation as the adversary cannot modify the prefix code content or forge any message sent by the prover (the prover simply waits to receive a challenge). Considering the verifier trusts the MRENCLAVE not leak any attestation secret, if the adversary interrupts after the response is sent, he cannot forge any message sent by the enclave.

If the timing test succeeds (and the response value is correct), the verifier can then use prover's ephemeral public key to establish a secure channel. This ensures that the rest of the communication is performed against the verified enclave.

Proving enclave execution In contract to former designs, in our case it is not sufficient for the attestation function to prove a known piece of code is run directly on hardware. Enclave software-attestation requires proving also that the code is executed in enclave mode. Hence we require that computing enclave's attestation function within a proper enclave must execute faster than any other computation possible of running out of the IEE (on that specific platform) and generating the equivalent result.

In the following section we assume the use of such a attestation function and provide

a formal treatment to reason about its exact properties and use. We then propose some concrete examples that *could* be implemented to satisfy our goal of proving enclave execution mode.

4.3.3. Generic enclave software-attestation protocol

To capture the security of an attestation scheme, we define a stronger version of an authenticated channel that we call an *attested IEE-channel*: an attested IEE-channel is, loosely speaking, an authenticated channel to a single instance of the program running in an IEE, that also guarantees that the program is actually running in an IEE (and not an adversarial simulated one, for example).

Attestated IEE-channel. A protocol $\Pi_{\mathbf{prog}}$ that provides an attested channel for a payload-program **prog** consists of three algorithms: $(V, P_{\mathbf{init}}, P_{\mathbf{prog}})$ and runs in two phases:

1. **Attestation Phase**: V and $P_{\mathbf{init}}$ run an interactive protocol. At the end of this phase, V outputs either vk or $\perp$ (where vk is a verification key for a public-key signature scheme) and $P_{\mathbf{init}}$ outputs (vk, sk).
2. **Execution Phase**: In this phase, $P_{\mathbf{prog}}(vk, sk)$ runs interactively by "wrapping" **prog**; when activated with input x, it activates **prog** with input x. When *prog* outputs a message m, $P_{\mathbf{prog}}$ outputs $(m, \text{Sign}sk(m))$.

Attestated IEE-channel parties.

- Prover - An enclave P that consists of both algorithms $P_{\mathbf{init}}$ and $P_{\mathbf{prog}}$.
- Verifier - Any external party out of the proving enclave running algorithm V.

Adversarial capabilities. We consider only software attacks preformed by the adversary (we trust the integrity of prover's hardware). The adversary has oracle access to enclave instances; that is, the adversary has only oracle access to the combined $(P_{\mathbf{init}}, P_{\mathbf{prog}})$ program running in every IEE (query and get results, but not break the enclave's IEE). P and V communicate via the prover's OS which includes all software stack out of the enclave's boundaries. The adversary can corrupt the OS. Particularly, can halt and execute the prover enclave, as well as create multiple instances of any enclave.

Definition 1 (Attested IEE-Channel)**.** *A protocol provides an* attested IEE-channel *to a payload-program* ***prog*** *against a class of adversaries $\mathcal{C}$ if for every adversarial prover $P^* \in \mathcal{C}$, given oracle access to $P_{\boldsymbol{prog}}$ (running in an IEE), the probability that P^* wins Fig. 4.5 is negligible.*

We formally define security using a game-based definition:

Figure 4.5.: Attestation Security Game for $\Pi_{\mathbf{prog}} = (V, P_{\mathbf{init}}, P_{\mathbf{prog}})$

The game is between a verifier V and the adversary P^*. Like the Attested IEE-channel protocol itself the game has two phases:

1. **Attestation Phase**: V and P^* run the protocol interactively until V outputs. P^* receives oracle access to n IEE instances containing $(P_{\mathbf{init}}, P_{\mathbf{prog}})$, where n is polynomial in the security parameter. Denote $(P^{(i)}_{\mathbf{init}}, P^{(i)}_{\mathbf{prog}})$ the i^{th} instance of the IEE. V outputs out_V
2. **Execution Phase**: P^* continues to execute (with oracle access to the IEE) until it outputs a message m^* and a signature σ^*.

Let $(vk^{(i)}, sk^{(i)})$ be the output of $P^{(i)}_{\mathbf{init}}$ (they can be $\perp$ if $P^{(i)}_{\mathbf{init}}$ did not have output). Let $\left\{m_1^{(i)}, \ldots, m_t^{(i)}\right\}$ be the set of responses the instance of $P^{(i)}_{\mathbf{prog}}$ receives from **prog** during the execution of the game. The adversary wins the game if $out_V \neq \perp$ (the verifier accepts in the attestation phase) and at least one of the following conditions is satisfied:

1. **Faked Key:** $out_V \notin \left\{vk^{(i)}\right\}_{i=1}^{n}$ or
2. **Forged Signature:** There exists $i \in [n]$ such that $out_V = vk^{(i)}$, $m^* \notin \left\{m_1^{(i)}, \ldots, m_t^{(i)}\right\}$ and Verify$out_V(m^*, \sigma^*) = 1$.

4.3.3.1. Time-based attested IEE-channel

In essence, time-based attestation relies on an "enclave-accelerated" function f. This function will map an MRENCLAVE value, we denote mr, and an arbitrary string to an "hard-to-compute" response. Loosely speaking, what we mean by "hard to compute" is that f running in a proper IEE context must execute faster than any other possible implementation of f on that specific platform. For completes of this approach, computing f out of the IEE should be feasible in time to verify a correct result.

Enclave-Acceleration Security Game. To capture the idea that f is fast to compute in the proper enclave, but significantly slower in any other setting, we define a security experiment between a verifier V and the adversary P^*. In this experiment, the adversary is given oracle access to a polynomial number of arbitrary IEE instances. The adversary's goal is to quickly compute $f(mr, x||vk)$ correctly for a random challenge string x, with vk an arbitrary string chosen by the adversary. The catch is that the adversary cannot use any IEE with identifier mr. For the formal definition, see Fig. 4.6.

Figure 4.6.: $\tau_{\text{MaxHonest}}$-Enclave-Acceleration Game for f

The game is between a verifier V and the adversary P^*, where P^* has oracle access to a polynomial number of arbitrary IEE instances.

1. V sends random challenge ch to P^*
2. P^* computes $resp$ and sends it to V
3. P^* sends mr, vk to V

P^* wins the experiment if all the following conditions are satisfied:

1. $resp = f(mr, ch \parallel vk)$
2. The time between steps (1) and (2) of the security experiment is at most $\tau_{\text{MaxHonest}}$
3. P^* did not make any oracle queries to IEE instances with the identifier mr.

We can now formally define the properties we need from the function f:

Definition 2 (Enclave-Accelerated Function)**.** *We say f is $(\tau_{MaxHonest}, \tau_{MinAdv})$-enclave accelerated against a class of adversaries $\mathcal{C}$ if for every input (mr, x), code running in the context of an IEE with identifier mr can compute f in time at most $\tau_{MaxHonest}$ (with overwhelming probability), while no adversary in $\mathcal{C}$ can win Fig. 4.6 with more than negligible probability.*

Definition 3 (Acceleration Gap)**.** *We say f has a (t, Δ) acceleration gap against a class of adversaries $\mathcal{C}$ if there exist (a, b) such that f is (b, a)-enclave accelerated against $\mathcal{C}$ and $a - tb \geq \Delta$.*

We use a generalized (t, Δ)-gap rather than simple $(1, \Delta)$ (the straightforward notion) in order to more easily specify the requirements from f if the underlying commitment scheme is (m, t)-extractable for $t > 1$ (e.g., an extractor that rewinds the committer may require $t = 2$).

Computation time and Malleability of f. A naïve attempt to construct a protocol using f's computation advantage within an IEE, would be to have the verifier simply bound the time the prover has between receiving a challenge ch and responding with $f(ch||vk)$. Intuitively, the prover will only be able to respond fast enough if f is being computed inside the IEE (and thus the signing key corresponding to vk is not under the adversary's control). Unfortunately, this intuition is slightly misleading; the reason is that f may be *malleable*: given $f(x)$, it might be possible to compute $f(x')$ for a related x' much more quickly than computing $f(x')$ from scratch. Since the adversary can use the IEE to compute $f(ch||vk)$, it might be able to use the result to compute $f(ch||vk')$ under the verifier's time limit (without violating the security assumptions on f).

We solve this issue by having the prover first *commit* (using a non-malleable commitment scheme) to f's result and then provide the actual result after a long-enough delay that the adversary cannot use it to cheat. For technical reasons (as will be seen in the proof of Theorem 1), we also require the commitment scheme to be *efficiently extractable*.

Efficiently-Extractable Commitment. A commitment scheme consists of two interacting parties, Commit and Verify, and works in two phases, *commitment* and *opening*. In the commitment phase, Commit and Verify interact, and at the end Verify either rejects or outputs a commitment state σ. In the opening phase, Verify receives σ as input. Commit sends the committed input and an opening proof to Verify, and Verify either accepts or rejects. Without loss of generality, we can assume that in the opening phase the committer sends the entire input (i.e., the message that was committed to and the random coins) to the verifier.

We require the commitment scheme to be both *binding* and *hiding* (see [28] for formal definitions).

We also require an additional property, for which we use a modified definition of *extractable commitment* (see [22]). The difference from "regular" extractable commitment is that we also measure the exact running time of the extractor (which we need to bound the running time in our reductions).

Loosely, we would like to have, for every adversarial committer that produces a valid commitment and that can later open its commitment, an extractor that can output the committed value using only the committer's code.

To model this, we think of the adversary as split into two parts: the committer (Commit*) and the opener (Open*); each one participates in the corresponding phase of the commitment scheme execution. Commit* receives as input random coins c^*, while Open* receives as input the adversary's view (c.f. Definition 4) and outputs an opening message.

Definition 4 (Adversary's View)**.** *The adversary's* view *of the commitment phase consists of her input c^* and all the messages sent by the verifier.*

Note that the entire view is known to the adversary at the end of the commitment phase (since the verifier does not send messages in the opening phase).

Definition 5 (Good Opening)**.** *We say an opening message m is* good*, relative to a view View if, conditioned on View, the probability that the verifier will accept m in the opening phase is a non-negligible function of the security parameter.*

Definition 6 (Good View)**.** *We say an adversary's view View is* good *if Open***(View) is a good opening.*

Definition 7 (Extractable View)**.** *We say E* successfully extracts *from a view $View_{Commit^*}$ if E(View) gives a good opening with all but negligible probability (in the security parameter)*

Let $T_{\text{Commit}^*}(\text{View})$ be the running time of Commit* on view View.

Definition 8 (Efficiently-Extractable Commitment)**.** *A commitment protocol (Commit, Verify) is* (m, t)-extractable *if for every adversarial committer (Commit***, Open***) there exists an algorithm E such that for every good view View, E successfully extracts from View, and E(View)'s expected running time is bounded by $t \cdot T_{Commit^*}(View) + m$.*

Note that if we remove the running-time requirement, the definition becomes trivial (the extractor can simply run Open* itself). However, if Open* has running time longer than $t \cdot T_{\text{Commit}} + m$, this property may no longer hold.

Efficiently-Extractable Commitment in the Random Oracle Model In the random oracle model, the "trivial" commitment scheme is also $(\varepsilon, 1)$-extractable for some small ε. This is because the extractor can intercept all calls to the random oracle, so it can extract the committed value from a single execution of the committer (the ε is the additional time it takes to store the commitment values in a table and lookup the results in that table).

P_{init} generic protocol. Given f which is $(\tau_{\text{MaxHonest}}, \tau_{\text{MinAdv}})$-enclave accelerated, we can now construct a generic P_{init} algorithm. Before presenting the protocol, we first consider all protocol components that could be leveraged by the adversary to gain time advantage. We define the time boundaries of protocol properties and components which are part of P_{init}'s time-sensitive section, but are not necessarily accelerated when run within the IEE.

For P to succeed in Protocol 1, we define a time-challenge threshold α. α is determined by the computation time of prover's time-critical section and the communication latency between protocol parties.

Computation time. P_{init}'s time-sensitive section includes both f and the (m, t)-extractable commitment. We use an f with a (t, Δ) acceleration gap. By definition, this guarantees the existence of $(\tau_{\text{MinAdv}}, \tau_{\text{MaxHonest}})$ such that f is $(\tau_{\text{MinAdv}}, \tau_{\text{MaxHonest}})$-enclave accelerated and $\tau_{\text{MinAdv}} - t \cdot \tau_{\text{MaxHonest}} > \Delta$.

Denote $\theta_{\text{MaxHonest}}$ be the maximum computation time for generating a commitment within an IEE. For clarity we assume a non-interactive commitment scheme is used. Our protocol can be generalized to use multi-round commitment schemes.

Communication time. The communication channel between V and P incurs transmission latency for every message sent. We denote the upper and lower bound of this delay time through $\delta_{\max}$ and $\delta_{\min}$ respectfully. The adversary can see every message on the channel and modify communication contents arbitrarily. However, the adversary cannot send massages faster than $\delta_{\min}$.

We thus define the time-challenge success threshold to be:

$$\alpha = \tau_{\text{MaxHonest}} + \theta_{\text{MaxHonest}} + 2 \cdot \delta_{\max}$$

Protocol 1 P_{init}

procedure PROVER
  Generate (vk, sk) key pair for public-key signature
  Send vk to Verifier
  Wait to receive ch from V
  Compute $resp_2 \leftarrow f(mr, ch \parallel vk)$
  $resp_1 \leftarrow Commit(resp_2)$
  Send $resp_1$ to Verifier
  Wait time α ▷ this can be done by computing a function that takes time at least α if the IEE does not have access to a "secure" timer
  Send $resp_2$ to V
end procedure
procedure VERIFIER
  Wait to receive vk from V
  Choose random challenge $ch \leftarrow \{0,1\}^k$
  $t_0 \leftarrow currenttime$
  Send ch to P
  Receive $resp_1$ from P
  $t_1 \leftarrow currenttime$
  Receive $resp_2$ from P
  Accept iff $resp_1$=Commit($resp_2$) and $resp_2 = f(mr, ch \parallel vk)$ and $(t_1 - t_0) < \alpha$.
end procedure

Payload restriction. Finally, in order to get meaningful security, we will have to restrict the program payload's behavior in some way; for example, a payload that "leaks" all the IEE memory (including attestation secrets) to the adversary would make the attestation step meaningless, since the adversary could then sign arbitrary messages without additional help from the IEE..

We define a sufficient criterion that ensures payloads cannot do this:

Definition 9 (Well-behaved payload)**.** *A payload* ***prog*** *is well-behaved if there does not exist an input x to* ***prog****, such that when* ***prog*** *is loaded into an IEE together with* $P_{\textbf{\textit{init}}}$

and $P_{\textbf{prog}}$ on input x reads or writes the memory area of its attestation-layer algorithms $P_{\textbf{init}}$ and $P_{\textbf{prog}}$ (i.e the proxy).

In particular, a well-behaved payload cannot access the signing key except via the signatures generated by the wrapping code $P_{\mathbf{prog}}$ (the verification key and signed messages might be passed to the code by the external adversary).

Theorem 1. *If f has a (t, Δ)-acceleration gap, **prog** is a well-behaved payload and $\Delta > t \cdot (\theta_{MaxHonest} + 2(\delta_{max} - \delta_{min})) + m$ then Protocol 1 is secure according to Definition 1.*

Proof. We show by reduction that if there exists an adversary A that breaks the attestation security Definition 1 then one of the following must hold:

1. There exists an algorithm B_1 that breaks the security of the signature scheme,
2. there exists an algorithm B_2 that breaks Definition 2 ,
3. there exists an algorithm B_3 that breaks the binding of the commitment scheme, or
4. there exists an algorithm B_4 that breaks the hiding of the commitment scheme.

Which of the cases holds depends on the behavior of A.

***A* wins the security game by Condition 2 of Fig. 4.5 (forged signature).** If A wins by Condition 2 with non-negligible probability p, we will show how to construct B_1 that can output a forged signature with probability p/n, where n is the bound on the number of IEE instances to which A can have access. Given a challenge verification key vk, B_1 runs as follows:

1: $i \xleftarrow{\$} [n]$ ▷ Choose a random IEE instance to use vk
2: $vk_i \leftarrow vk$
3: **for** $j \in [n] \setminus \{i\}$ **do** ▷ Generate signing keys for remaining instances
4: $(sk_j, vk_j) \leftarrow Gen(1^\kappa)$
5: **end for**
6: Execute A, simulating n IEE oracles, where instance $j \in [n]$ has verification key vk_j; for queries to IEE $j \neq i$, use sk_j to sign messages in the **prog** phase. For instance i, use the unforgeability game's signing oracle to sign messages.
7: **if** A outputs m^* such that $m^* \notin \left\{m_1^{(i)}, \ldots, m_t^{(i)}\right\}$ and $\text{Verify}_{out_V}(m^*, \sigma^*) = 1$ **then**
8: output m^*
9: **end if**

Since **prog** is a well-behaved payload, it cannot output anything correlated with $sk_i \in \{sk_1, \ldots, sk_n\}$, with the exception of data received in previous rounds (this can be correlated with the signing key, since the verifier has the verification key and signed messages). Hence by induction on the round number, we can simulate $P_{\mathbf{prog}}$ using the signing oracle (we don't need to simulate $P_{\mathbf{prog}}$ for the message for the first round, since we get it from the verifier which we can fully execute, and for subsequent rounds we

again don't need to simulate since we compute the input by executing the verifier with the output of previous rounds).

Since all the verification keys are identically distributed, A's choice of which verification key to attack cannot depend on the choice of i. Thus, if B_1 reaches step 8, then A must choose vk_i with probability at least $1/n$. When this occurs, B_1 wins the unforgeability game. Thus, B_1 will win the unforgeability game with probability at least p/n.

***A* wins the security game by Condition 1 of Fig. 4.5 (faked key).** When A wins the game against the verifier of Protocol 1, *all* of the following conditions must have been satisfied:

1. $resp_2$ is a valid opening for the commitment $(resp_1)$
2. $resp_1 = f(mr, ch||vk)$ and
3. $t_1 - t_0 < \alpha$ (the response was fast enough).

For every A, we can define a "commitment adversary" $A_{\text{Commit}}, A_{\text{Open}}$ as follows:

1. We first generate n signature key pairs, and use them to emulate n IEE instances.
2. We execute A, simulating the verifier until step 13 (after A sends the verification key vk).
3. A_{Commit} is defined to be A, with the initial memory state from the previous step "hard-wired", and terminating execution after sending the first response message to the verifier; the random coins of A_{Commit} consist of the random coins of A and the challenge ch. The output of A_{Commit} is its entire memory state.
4. A_{Open} is defined to be A, setting its memory state to its input, and with execution starting just after sending the first response message to the verifier and ending after sending the second response message.

Since our commitment scheme is extractable, for every $(A_{\text{Commit}}, A_{\text{Open}})$ as defined above, there exists an extractor E.

Consider the following algorithm B_2:

1: Let $c^* \xleftarrow{\$} \{0,1\}^*$ ▷ Choose random coins for A
2: Execute $A(c^*)$, simulating the verifier until step 13.
Let vk be the verification key received by the simulated verifier at step 12.
3: Send vk to the real verifier (of Fig. 4.6).
4: Wait to receive the challenge ch from the real verifier.
5: Execute E, using the random coins of A as input.
▷ Note that given ch, the input to A_{Commit} is fully defined
Let $resp^* = (msg, opening)$ be the output of E.
6: Send msg to the real verifier.

We say an execution (i.e., choice of random coins) for B_2 is a *winning execution* if the interaction of A and the simulated verifier results in A winning the security game by Condition 1. For a winning execution, A must run in time less than α. Since every "real" instance of IEE of $P_{\textbf{init}}$ will wait for at least α time before opening the commitment, no IEE instance of $P_{\textbf{init}}$ will need to be simulated beyond step 7 of Protocol 1.

This means that in a winning executions we can replace the real IEE instances with "fake" instances: in a fake instance of $P_{\textbf{init}}$, the computation of f in step 5 is replaced by $resp_2 \leftarrow 0$. Denote $B_2^{(j)}$ the algorithm B_2 where the first j IEE instances of $P_{\textbf{init}}$ are replaced with fakes (e.g., $B_2 = B_2^{(0)}$). We consider the following two subcases:

Case 1: Conditioned on a winning execution of B_2, the output transcripts of B_2 and $B_2^{(n)}$ are indistinguishable. In this case, we can use B_2 to either break the security of f or the binding property of the commitment scheme.

Suppose $A(c^*)$ wins the security game by Condition 1 with non-negligible probability p (over the coins of the verifier). When A wins in this way, one of the following must hold:

Case 1.1: Conditioned on a winning execution of B_2, the probability that $msg = f(mr; ch||vk)$ is overwhelming (where msg is the value sent in step 6). In this case, we use B_2 to break the security of f. Let τ be the time it takes for B_2 to execute steps 4–6. Since these steps consist of a single execution of E, $\tau < t \cdot T_A + m$, where t, m are the extraction parameters of the commitment scheme and T_A is a bound on the time it takes to run A_{Commit}. Since A wins, $T_A < \alpha - 2\delta_{\min}$. Moreover, by the definition of α, f is $(\tau_{\text{MaxHonest}}, \tau_{\text{MinAdv}})$-accelerated, and $\tau_{\text{MinAdv}} - t \cdot \tau_{\text{MaxHonest}} > \Delta$, which in turn implies $t \cdot \tau_{\text{MaxHonest}} < \tau_{\text{MinAdv}} - \Delta$. Thus,

$$\begin{aligned}
\tau &< t \cdot (\alpha - 2\delta_{\min}) + m \\
&= t \cdot (\tau_{\text{MaxHonest}} + \theta_{\text{MaxHonest}} + 2(\delta_{\max} - \delta_{\min})) + m \\
&< \tau_{\text{MinAdv}} - \Delta + t \cdot (\theta_{\text{MaxHonest}} + 2(\delta_{\max} - \delta_{\min})) + m \\
&< \tau_{\text{MinAdv}}
\end{aligned}$$

(where the last inequality is since $\Delta > t \cdot (\theta_{\text{MaxHonest}} + 2(\delta_{\max} - \delta_{\min})) + m$).

Since B_2 computed f's result in less than τ_{MinAdv}, B_2 broke our assumption that f is $(\tau_{\text{MinAdv}}, \tau_{\text{MaxHonest}})$-enclave accelerated.

Case 1.2: Conditioned on a winning execution of B_2, the probability that $msg \neq f(mr; ch||vk)$ is non-negligible. We denote this case as algorithm B_3: A winning execution of B_2 that breaks the binding property of the commitment scheme.

By the extractor property, $(msg, opening)$ is a valid opening for the commitment. However, since $msg \neq f(mr; ch||vk)$ it would not be accepted by the verifier. Since this is a winning execution of B_3, the value $resp_2$ sent eventually to the verifier must be both a valid opening of the commitment. Since

$resp_2$ is equal to $f(mr; ch||vk)$, we know that $resp_2 \neq msg$. We have found two different opening values for the commitment, hence broke the binding property of the commitment scheme with non-negligible probability.

Case 2: Conditioned on a winning execution of B_2, the output transcripts of B_2 and $B_2^{(n)}$ *are* distinguishable. That is, there exists a distinguisher D that can guess whether we are running B_2 or $B_2^{(n)}$ with non-negligible advantage δ. In this case, we can define an algorithm B_4 that will break the commitment hiding with advantage δ/n^2. First, by a standard hybrid argument there exists some $j \in \{0, \ldots, n-1\}$ such that D can distinguish $B_2^{(j)}$ and $B_2^{(j+1)}$. We define B_4 to be:

1: Generate a random verification key vk
2: choose $ch \xleftarrow{\$} \{0,1\}^\kappa$
3: Let $msg_0 = f(mr; ch||vk)$ and $msg_1 = 0$. Send these as the challenge messages to the commitment challenger.
4: Generate n signature key pairs, and use them to emulate n IEE instances of the $P_{\textbf{init}}$ type.
5: Execute B_2, simulating the first $j-1$ the IEE instances "fake" instances and the last $n-j$ instances as "real". For the j^{th} instance, return the commitment challenge instead of running the computation in step 5.
6: Output the output of D on the transcript of B_2.

Note that the execution of B_2 in step 5 is exactly $B_2^{(j)}$ if the challenge message was msg_0, and $B_2^{(j+1)}$ if it was msg_1. Since D can distinguish the two cases with advantage at least δ/n, we break the hiding property of the commitment with this advantage.

Concluding, one of the following statements must have occurred: A either broke the forgeability game, violated our initial assumption that f is $(\tau_{\text{MaxHonest}}, \tau_{\text{MinAdv}})$-enclave-accelerated, broke the commitment scheme binding or hiding properties (or broke the IEE). Each one of these contradicts at least one assumption about A.

Hence their does not exist an adversary that can break Definition 1 over $P_{\textbf{init}}$ protocol. In other words, $P_{\textbf{init}}$ is secure according to Definition 1 if f has a (t, Δ)-acceleration gap, $\Delta > t \cdot (\theta_{\text{MaxHonest}} + 2(\delta_{\max} - \delta_{\min})) + m$ and **prog** is an well-behaved payload.

□

4.3.3.2. Amplifying the Acceleration Gap

Since f may provide an insufficient acceleration gap to compensate for prover's communication latency and commitment computation time, we wish to amplify the prover's marginal advantage. We propose using an iteration parameter l, as described in Section 4.3.1, that determines the number of loops over f needed for an success threshold α.

Depending on the concrete realization of f, it may have a (t, Δ)-acceleration gap that when iterated ℓ-times over itself (running over its previous output), the result may not

have increased the acceleration gap, or worse, even decreased it.To provide an amplified enclave-accelerated function f' that builds on any f, we show a construction of f' under the random-oracle model. We denote $H(x)$ as a random-oracle query over input x. We define two time-bounds for the time it takes to call one oracle query: $\gamma_{\text{MaxHonest}}$ - the maximum time for a query from within an IEE, and γ_{MinAdv} - the minimum time for querying the oracle out of an IEE, where $\gamma_{\text{MinAdv}} < \gamma_{\text{MaxHonest}}$.

We thus define f' as follows:

Algorithm 2 $f'(mr, ch \parallel vk, \ell)$

1: $result_1 \leftarrow f(mr, ch \parallel vk)$
2: **for all** $i \in \{2, \ldots, \ell\}$ **do**
3: $\quad ch_i \leftarrow H(result_{i-1})$
4: $\quad result_i \leftarrow f(mr, ch_i \parallel vk)$
5: **end for**
6: **return** $result_\ell$

Theorem 2. *If f has a (t, Δ)-acceleration gap then Algorithm 2 has a $(t, \ell(\Delta - (t \cdot \gamma_{MaxHonest} - \gamma_{MinAdv})) + t \cdot \gamma_{MaxHonest} - \gamma_{MinAdv})$-acceleration gap.*

Proof. Denote $(\tau_{\text{MaxHonest}}, \tau_{\text{MinAdv}})$ the parameters for which f is $(\tau_{\text{MaxHonest}}, \tau_{\text{MinAdv}})$-accelerated and that satisfy $\tau_{\text{MinAdv}} - t \cdot \tau_{\text{MaxHonest}} \geq \Delta$ (as promised by the (t, Δ)-acceleration gap).

Completeness: During the execution of f', f is called a total of ℓ times and H is called $\ell - 1$ times. Thus, the total time for evaluating f' is bounded by

$$\tau'_{\text{MaxHonest}} \stackrel{def}{=} \ell \cdot \tau_{\text{MaxHonest}} + (\ell - 1) \cdot \gamma_{\text{MaxHonest}} = \ell \cdot (\tau_{\text{MaxHonest}} + \gamma_{\text{MaxHonest}}) - \gamma_{\text{MaxHonest}} \ .$$

Soundness: We show by induction that A cannot evaluate f' faster than

$$\tau'_{\text{MinAdv}} \stackrel{def}{=} \ell \cdot (\tau_{\text{MinAdv}} + \gamma_{\text{MinAdv}}) - \gamma_{\text{MinAdv}} \ .$$

Together with the upper bound $\tau'_{\text{MaxHonest}}$, this implies that f' is $(\tau'_{\text{MaxHonest}}, \tau'_{\text{MinAdv}})$-accelerated, and hence we can find its time gap Δ':

$$\begin{aligned}
\tau'_{\text{MinAdv}} - t \cdot \tau'_{\text{MaxHonest}} &= \ell \cdot (\tau_{\text{MinAdv}} + \gamma_{\text{MinAdv}}) \\
&\quad - \gamma_{\text{MinAdv}} - t \cdot (\ell \cdot (\tau_{\text{MaxHonest}} + \gamma_{\text{MaxHonest}}) - \gamma_{\text{MaxHonest}}) \\
&= \ell \cdot (\tau_{\text{MinAdv}} - t \cdot \gamma_{\text{MinAdv}} - t \cdot \gamma_{\text{MaxHonest}} + \gamma_{\text{MinAdv}}) + t \cdot \gamma_{\text{MaxHonest}} - \gamma_{\text{MinAdv}} \\
&= \ell(\Delta - (t \cdot \gamma_{\text{MaxHonest}} - \gamma_{\text{MinAdv}})) + t \cdot \gamma_{\text{MaxHonest}} - \gamma_{\text{MinAdv}}
\end{aligned}$$

Base case: $\ell = 1$. In this case f' consists of a single execution of f, so the induction hypothesis follows directly from the fact that f is $(\tau_{\text{MaxHonest}}, \tau_{\text{MinAdv}})$-accelerated.

Induction step: We show that if our theorem holds for ℓ then it also holds for $\ell + 1$. Suppose our theorem holds for ℓ but not for $\ell + 1$. In other words, A can execute $f'_{\ell+1}$ in time less than $(\ell + 1) \cdot (\tau_{\text{MinAdv}} + \gamma_{\text{MinAdv}}) - \gamma_{\text{MinAdv}}$.

Consider the sequence of calls A makes to H. Note that if for some $i \in \{1, \ldots, \ell\}$, A does not query H on $result_i$ then A will fail to compute f' with overwhelming probability. Intuitively, this is because $H(result_i)$ is independent of f and of H's value on any other query, so in effect A must compute f given no information at all about the challenge. If A can do that, it could just as easily precomputed the response (before receiving the challenge), and so could break the security of f (Fig. 4.6).

Formally, we can prove this by induction on ℓ:

Base case (for nested induction): $\ell = 2$. In this case A did not query H on $result_2$, hence had to guess ch_3 or directly guess $result_3$ with non-negligible probability. The first guess would mean A broke the random oracle, the second means A computed f' without knowing ch hence broke our assumption that f' $(\tau_{\text{MaxHonest}}, \tau_{\text{MinAdv}})$-enclave-accelerated.

Induction step (for nested induction): If A cannot compute f' without querying H on $result_\ell$, then A cannot compute f' without querying H on $result_{\ell+1}$. Suppose our theorem holds for ℓ but not for $\ell + 1$. Meaning, A can compute f' without querying H on $result_{\ell+1}$. Then we can construct algorithm B that breaks our assumption that f' is $(\tau_{\text{MaxHonest}}, \tau_{\text{MinAdv}})$-enclave accelerated.

Consider the following algorithm B as the prover of Fig. 4.6:

1: Let $resp$ be A's result for $\ell + 2$ f' iterations.
2: Wait to receive ch_v from the verifier.
3: Send $resp$ to the verifier.

If we assume A returns the correct final result, but cannot query H on $result_{\ell+1}$, then A had to guess correctly ch_2 (or guessed f' result). This is the same as assuming A guessed ch_v and computed the correct result for f' before even receiving ch_v from the verifier.

B satisfies all three condition of Fig. 4.6:

1. Since we can assume A "guessed" the correct f' result with non-negligible probability, B's response is equal to $f(mr, ch_v \parallel vk)$ with non-negligible probability.

2. B responded immediately after receiving ch_v (in less than $\tau_{\text{MaxHonest}}$).

3. Nor A or B made any oracle queries to IEE instance.

Hence B wins the Fig. 4.6 with non-negligible probability, breaking our initial assumption that f' is $(\tau_{\text{MaxHonest}}, \tau_{\text{MinAdv}})$-enclave-accelerated.

This induction result also shows that the queries must be *in order* (i.e., A must query H on $result_j$ before querying on $result_{j+1}$). For $i \in \{1, \ldots, \ell\}$, we denote t_i the time at which A queries H on $result_i$.

If A computed $f'_{\ell+1}$ in time less than $(\ell + 1) \cdot (\tau_{\text{MinAdv}} + \gamma_{\text{MinAdv}}) - \gamma_{\text{MinAdv}}$, then either:

Case 1: $t_{\ell+1} - t_\ell < (\tau_{\text{MinAdv}} + \gamma_{\text{MinAdv}})$. In this case, we either computed f faster than τ_{MinAdv} or H faster than γ_{MinAdv} (the computation of f and the call to H must be sequential because H is atomic—we model it as an oracle that must be given a complete query, and then returns a complete response after time γ_{MinAdv}).

Case 2: $t_\ell < \ell \cdot (\tau_{\text{MinAdv}} + \gamma_{\text{MinAdv}}) - \gamma_{\text{MaxHonest}}$. In this case, we contradict the induction hypothesis. Since we assume our theorem holds for ℓ, we know that f'_ℓ cannot be computed in time less than $\ell \cdot (\tau_{\text{MinAdv}} + \gamma_{\text{MinAdv}}) - \gamma_{\text{MinAdv}}$.

We've proven $\tau'_{\text{MaxHonest}}$ to be the upper bound for computing f' in the honest case, and τ'_{MinAdv} the best adversarial computation time. Since $\tau_{\text{MaxHonest}} < \tau_{\text{MinAdv}}$, then $\tau'_{\text{MaxHonest}} < \tau'_{\text{MinAdv}}$. Hence f' is $(\tau'_{\text{MaxHonest}}, \tau'_{\text{MinAdv}})$-enclave accelerated.

We've also shown that $(\tau'_{\text{MaxHonest}}, \tau'_{\text{MinAdv}})$ satisfies $(\tau'_{\text{MinAdv}} - t \cdot \tau'_{\text{MaxHonest}} \geq \Delta')$, hence we conclude that Algorithm 2 has a $(t, \ell(\Delta - (t \cdot \gamma_{\text{MaxHonest}} - \gamma_{\text{MinAdv}})) + t \cdot \gamma_{\text{MaxHonest}} - \gamma_{\text{MinAdv}})$-acceleration gap for every function f that has a (t, Δ)-acceleration gap.

□

4.3.4. Can Intel build an enclave-accelerated function?

Implementing a function that executes faster within an enclave greatly depends on SGX's internal implementation. Despite the lack of SGX micro-architectural documentation, we propose several candidates that *could* be implemented to realize an enclave-accelerated function.

Using existing hardware. Here we propose using a checksum function that records processors execution-state as implemented by [38]. This is achieved by including environmental feature flag registers that indicate the execution mode of the processor. Since CPUs register file is the only memory that operates at the CPUs clock frequency, it can be used without latency penalty. Adding carefully chosen registers will incorporate processor's enclave execution mode status into the checksum result, in addition to reflecting the attestation function integrity.

Intuitively, enclaves incurs computation overhead that could easily overpower the marginal time difference essential for the attestation function. This includes both security mechanisms added to support enclave's IEE: the MEE encryption-decryption process and the new memory access controls enforced in enclave-mode. However, SGX unique properties described in section 2 suggest that running a function that fits within processor's L1 cache, satisfies the optimality requirement of present software-attestation methods. The MEE only plays a part in enclave execution when evicting and fetching pages from/to the main memory. Additionally, to reduce IEE performance overhead, SGX access control checks are performed merely during the address translation process and not for each page request. In other words, enclave's code and data are stored in plain text when within processor's boundaries. Hence it should be possible to follow Pioneer's technique to construct an optimal attestation function that fits within the L1 cache of the processor.

The lack of SGX internal implementation prevents us from pointing out the exact registers to be introduced to the checksum calculation. However, we believe that if Intel wanted to, it should be easy to include execution context-registers to the checksum function for witnessing enclave-mode.

Additionally, it may also be possible to leverage SGX's ability to restrict access of certain processor opcodes exclusively to code running within an enclave. Existing enclave-only opcodes may include one which is already enclave-accelerated. However present documentation does not provide enough details to ensure this property. We leave this open question for future work.

Modifying SGX firmware. Since enclave opcodes are implemented largely by microcode, it could be possible to implement a new enclave-only opcode (or a variant of an existing one) that fulfills the role of an enclave-accelerated function, as part of a firmware update.

Regular opcodes, that are not microcode based, consists of up to 4 micro-ops. Microcode-based opcodes are used to implement complex instructions that may consists of dozens (or hundreds) of micro-ops. Like L1 cache, microcode is also core-specific. Moreover, microcode spears the fetch operation needed to load code from L1 and is loaded directly into processors decode pipeline operation [21]. Hence, it may be possible to construct a recursive microcode function that cannot be accelerated when implemented out of the microcode. Specifically, if enough microcode space is available, the self-checksumming loop described in Section 4.3.1 as part of the attestation function, may be implemented as an enclave-only microcode-based opcode.

Existing microcode-based opcodes can also satisfy our enclave-accelerated needs by modifying them slightly to get a unique variant which is enclave-only. Consider a set of dedicated micro-ops that are used exclusively to implement a hardware-accelerated instruction. It should be possible to construct a unique use of these hardware-accelerated micro-ops for realizing a enclave-only and enclave-accelerated instruction. As in current enclave-only instructions, the microcode will use the MRENCLAVE of processor's correct execution as part of its computation in order to bind the identity of the invoking enclave to its result.

Modifying SGX hardware. Processor's architecture may be utilized to compute a highly parallelizable procedure much faster than any software implementation of that procedure. Implementing such a function as an enclave-only opcode can satisfy our enclave-accelerated requirements. One candidate for such a function is the Goldreich's one-way function [27]. Goldreich's function can be computed by a constant low depth circuit hence can be implemented using a small number of hardware logical gates to be computed in parallel by the processor. This will provide a hardware-unique efficiency for a specific OWF function chosen.

Considering such an instruction requires a small amount of CPU cycles, we can then enumerate all possible opcode combinations that require the same amount of CPU cycles, and prove that none of them can compute the equivalent result. Therefore only an enclave can generate the correct result under a certain time threshold.

4.3.5. Software-based enclave attestation: immediate applications

Close-proximity usages. A major shortcoming of all software attestation schemes, is their extreme dependency on low-latency message transmission, which commonly trans-

lates to physical proximity. However, we wish to note some exemplary cases where enclaves may significantly benefit from a close-proximity attestation scheme that does not depend on hardware secrets.

One example is using a small computing device (such as a portable smart USB stick) to validate an expected enclave is properly running on a platform. Another useful example, is for smart peripherals, such as a screen or a keyboard, to establish a trusted Input-Output path between an enclave and platform's user, bridging any potentially malicious software or hardware between them.

Independent Enclave. Software-based enclave attestation can be bootstrapped to enable an independent enclave sealing primitive as described in Section 4.2. Neighboring platforms on the LAN (or peripherals) can be used to hold a share of enclave's DSK. For each boot-cycle the enclave reconstructs its DSK by attesting to its share holders. The result is an enclave that allows other enclaves on the platform to reform both attestation and sealing primitives without relying on the original SGX sealing and attestation primitives. Crucially, DSK is persistent between platform power cycles. This completely losses enclave's dependency on the strong assumption of embedded device key confidentiality. DSK can then be used as a platform specific "virtual fuse key" to facilitate alternative attestations against remote verifiers.

Previous software attestation works where limited to test their schemes on the same ethernet segment to reduce the impact of attestation amplification exhaustion. Using a platform specific "virtual fuse key" within an enclave, SGX enables us to leverage a boot-time software-attestation to conduct Intel-independent attestations against any remote verifier.

Application for organizational networks. Consider an organization that strictly controls all platforms connected to the organizational network. The network administrator wishes to remotely verify an enclave running on any platform within the organizational boundaries.

To utilize our timing-based attestation framework in this scenario, we add an assumptions that network latency in the organization is at least α (attestation success threshold) smaller than the latency of communication across organizational boundaries. Additionally, we assume the administrator manages all hardware in the organization such that all platforms in the organization are of the same or older generation of the attesting SGX platform. Introducing a stronger computer in the organization is referred to as an attack on the network.

Last, in this scenario several platforms may collude to beat the attestation threshold by computing f parallely. Hence if f is realized by exploiting processor's parallel computation features, its amplification must be implemented such that each parallel computation cannot be amplified independently of the rest. This ensures a colluding group of platforms must incur additional communication latency to compute the correct attestation result.

5. Conclusion and future work

5.1. Conclusion

There is growing recognition that trusted execution environments such as SGX provide critical security services in a world with growing dependence on computing and communication. In particular, attestation and sealing are key services provides by these environments; they enable a wide range of security applications. However, like other security services (e.g., authentication using cryptographic certificates), attestation and sealing require a whole ecosystem to function effectively. The specific way that this ecosystem is constructed entails a certain set of capabilities and a certain set of risks. For SGX, Intel choices are fairly conservative; they limit usage scenarios but protect participants from many risks, and position them selfs as liable for securing, facilitating and governing the usage of this technology.

In this work we provided an extensive description of the SGX ecosystem and a simplified "two pillar" representation of enclave's security design. We explored the limits of SGX guarantees entailed by Intel's design choices in term of availability, privacy and trust. In light of these assessments, we provide several proposals to "emancipate" enclave's sealing and attestation primitives. In addition to reducing the SGX Intel-dependent TCB, our proposals also provide enhanced privacy and availability capabilities that rely less on Intel and more on user-chosen implementation.

5.2. Future work

SGX software development kits were released by Intel late towards the end of this research. It is in our intention to implement prof of concepts for our alternative attestation proposals. Especially, creating alternative Quotes that do not depend on an trusted service provider or on IAS availability for verification, as described in .

Regarding our time-based proposal, experiments can be conducted to explore the possibility of an existing enclave-only opcode that satisfies our enclave-accelerated function definitions. Additionally, interesting work can be done together with Intel on realizing such a function; implementing a tailor-made SGX attestation function, implementing a microcode procedure or designing the minimal hardware changes needed for our goals.

Bibliography

[1] Intel. SGX Tutorial, ISCA 2015. http://sgxisca.weebly.com/, June 2015.

[2] Intel. Intel Software Guard Extensions Programming Reference (rev2), Oct. 2014. 329298-002US.

[3] Intel. Intel Software Guard Extensions Programming Reference (rev1), Sept. 2013. 329298-001US.

[4] http://www.google.com/patents/US20140093074.

[5] http://www.google.com/patents/US8885819.

[6] Intel Software Guard Extensions: Intel Attestation Service API, version 2. https://software.intel.com/sites/default/files/managed/7e/3b/ias-api-spec.pdf.

[7] Intel SGX Product Licensing https://software.intel.com/en-us/articles/intel-sgx-product-licensing.

[8] Lockheed Martin breach https://www.theguardian.com/world/2009/apr/21/hackers-us-fighter-jet-strike.

[9] Billions of Gemalto embedded keys stolen https://www.theguardian.com/us-news/2015/feb/19/nsa-gchq-sim-card-billions-cellphones-hacking.

[10] Trusted computing group. https://www.trustedcomputinggroup.org/.

[11] L. v. D. P. K. A. Seshadri, A. Perrig. Swatt: software-based attestation for embedded devices. CyLab, Carnegie Mellon Univ., Pittsburgh, PA, USA, 2004. IEEE.

[12] I. Anati, S. Gueron, S. Johnson, and V. Scarlata. Innovative technology for CPU based attestation and sealing. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, volume 13, 2013.

[13] F. Armknecht, A.-R. Sadeghi, S. Schulz, and C. Wachsmann. A security framework for the analysis and design of software attestation. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 1–12, New York, NY, USA, 2013. ACM.

[14] T. W. Arnold and L. van Doorn. The IBM PCIXCC: A new cryptographic coprocessor for the IBM eserver. *IBM Journal of Research and Development*, 48(3-4):475–488, 2004.

[15] M. Barbosa, B. Portela, G. Scerri, and B. Warinschi. Foundations of hardware-based attested computation and application to sgx. Cryptology ePrint Archive, Report 2016/014, 2016. http://eprint.iacr.org/2016/014.

[16] A. Baumann, M. Peinado, and G. Hunt. Shielding applications from an untrusted cloud with haven. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 267–283, Broomfield, CO, Oct. 2014. USENIX Association.

[17] E. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. In *Proceedings of the 11th ACM Conference on Computer and Communications Security*, CCS '04, pages 132–145, New York, NY, USA, 2004. ACM.

[18] E. Brickell and J. Li. Enhanced privacy id from bilinear pairing. Cryptology ePrint Archive, Report 2009/095, 2009. http://eprint.iacr.org/2009/095.

[19] C. Castelluccia, A. Francillon, D. Perito, and C. Soriente. On the difficulty of software-based attestation of embedded devices. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 400–409, New York, NY, USA, 2009. ACM.

[20] Y.-G. Choi, J. Kang, and D. Nyang. Proactive code verification protocol in wireless sensor network. In *Proceedings of the 2007 International Conference on Computational Science and Its Applications - Volume Part II*, ICCSA'07, pages 1085–1096, Berlin, Heidelberg, 2007. Springer-Verlag.

[21] V. Costan and S. Devadas. Intel SGX explained. Cryptology ePrint Archive, Report 2016/086, 2016. http://eprint.iacr.org/.

[22] G. D. Crescenzo. *Equivocable and Extractable Commitment Schemes*, pages 74–87. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[23] C. F. C. G. M. P. G. M.-R. M. R. Felix Schuster, Manuel Costa. Vc3: Trustworthy data analytics in the cloud. Technical report, February 2014.

[24] A. Francillon, Q. Nguyen, K. B. Rasmussen, and G. Tsudik. Systematic treatment of remote attestation. Cryptology ePrint Archive, Report 2012/713, 2012. http://eprint.iacr.org/2012/713.

[25] J. A. Garay and L. Huelsbergen. Software integrity protection using timed executable agents. In *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security*, ASIACCS '06, pages 189–200, New York, NY, USA, 2006. ACM.

[26] R. Gardner, S. Garera, and A. D. Rubin. On the difficulty of validating voting machine software with software. In *Proceedings of the USENIX Workshop on Accurate Electronic Voting Technology*, EVT'07, pages 11–11, Berkeley, CA, USA, 2007. USENIX Association.

[27] O. Goldreich. Candidate one-way functions based on expander graphs. *Electronic Colloquium on Computational Complexity (ECCC*, page 2000.

[28] O. Goldreich. *Foundations of Cryptography: Volume 1.* Cambridge University Press, New York, NY, USA, 2006.

[29] V. Gratzer and D. Naccache. Alien vs. quine. *IEEE Security and Privacy*, 5(2):26–31, Mar. 2007.

[30] L. Gu, X. Ding, R. H. Deng, B. Xie, and H. Mei. Remote attestation on program execution. In *Proceedings of the 3rd ACM Workshop on Scalable Trusted Computing*, STC '08, pages 11–20, New York, NY, USA, 2008. ACM.

[31] S. Gueron. A memory encryption engine suitable for general purpose processors. Cryptology ePrint Archive, Report 2016/204, 2016. `http://eprint.iacr.org/`.

[32] M. Hendry. Smart card security and applications. Artech House, 2001.

[33] M. Hoekstra, R. Lal, P. Pappachan, V. Phegade, and J. Del Cuvillo. Using innovative instructions to create trustworthy software solutions. In *HASP@ ISCA*, page 11, 2013.

[34] Intel. Intel 64 and IA-32 architectures software developer's manual. *Volume 3a: System Programming Guide*, September 2015.

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

[36] M. Jakobsson and K.-A. Johansson. Practical and secure software-based attestation. In *Proceedings of the 2011 Workshop on Lightweight Security & Privacy: Devices, Protocols, and Applications*, LIGHTSEC '11, pages 1–9, Washington, DC, USA, 2011. IEEE Computer Society.

[37] S. Johnson, V. Scarlata, C. Rozas, E. Brickell, and F. Mckeen. Intel software guard extensions: EPID provisioning and attestation services, April 2016.

[38] R. Kennell and L. H. Jamieson. Establishing the genuinity of remote computer systems. In *Proceedings of the 12th Conference on USENIX Security Symposium - Volume 12*, SSYM'03, pages 21–21, Berkeley, CA, USA, 2003. USENIX Association.

[39] S. Kim, Y. Shin, J. Ha, T. Kim, and D. Han. A First Step Towards Leveraging Commodity Trusted Execution Environments for Network Applications. In *Proceedings of the 14th ACM Workshop on Hot Topics in Networks (HotNets)*, Philadelphia, PA, Nov. 2015.

[40] Y. Li, J. McCune, J. Newsome, A. Perrig, B. Baker, and W. Drewry. Minibox: A two-way sandbox for x86 native code. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 409–420, Philadelphia, PA, June 2014. USENIX Association.

[41] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar. Innovative instructions and software model for isolated execution. In *HASP@ ISCA*, page 10, 2013.

[42] M. Naor. On cryptographic assumptions and challenges. In *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 96–109. Springer, 2003.

[43] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In *2008 IEEE Symposium on Security and Privacy (S&P 2008), 18-21 May 2008, Oakland, California, USA*, pages 111–125, 2008.

[44] A. Pfitzmann and M. Hansen. A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management. http://dud.inf.tu-dresden.de/literatur/Anon_Terminology_v0.34.pdf, Aug. 2010. v0.34.

[45] X. Ruan. *Platform Embedded Security Technology Revealed: Safeguarding the Future of Computing with Intel Embedded Security and Management Engine*. Apress, Berkely, CA, USA, 1st edition, 2014.

[46] A. Seshadri, M. Luk, and A. Perrig. Sake: Software attestation for key establishment in sensor networks. *Ad Hoc Netw.*, 9(6):1059–1067, Aug. 2011.

[47] A. Seshadri, M. Luk, A. Perrig, L. van Doorn, and P. Khosla. Scuba: Secure code update by attestation in sensor networks. In *Proceedings of the 5th ACM Workshop on Wireless Security*, WiSe '06, pages 85–94, New York, NY, USA, 2006. ACM.

[48] A. Seshadri, M. Luk, E. Shi, A. Perrig, L. van Doorn, and P. Khosla. Pioneer: Verifying code integrity and enforcing untampered code execution on legacy systems. *SIGOPS Oper. Syst. Rev.*, 39(5):1–16, Oct. 2005.

[49] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla. Using SWATT for verifying embedded systems in cars. In *Proceedings of Embedded Security in Cars Workshop (ESCAR)*, Nov. 2004.

[50] A. Shamir. How to share a secret. In *Communications of the ACM 22, Volume 22 Issue 11*, page 612613, 1979.

[51] M. Shaneck, K. Mahadevan, V. Kher, and Y. Kim. *Remote Software-Based Attestation for Wireless Sensors*, pages 27–41. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[52] E. Shi, A. Perrig, and L. V. Doorn. Bind: A fine-grained attestation service for secure distributed systems. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, SP '05, pages 154–168, Washington, DC, USA, 2005. IEEE Computer Society.

[53] M. Shih, M. Kumar, T. Kim, and A. Gavrilovska. S-NFV: securing NFV states by using SGX. In *Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization, SDN-NFV@CODASPY 2016, New Orleans, LA, USA, March 11, 2016*, pages 45–48, 2016.

[54] C. R. E. B. F. M. Simon Johnson, Vinnie Scarlata. Intel software guard extensions: Epid provisioning and attestation services, 2016. `https://software.intel.com/en-us/blogs/2016/03/09/intel-sgx-epid-provisioning-and-attestation-services`.

[55] L. Sweeney. Achieving k-anonymity privacy protection using generalization and suppression. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):571–588, Oct. 2002.

A. SGX ecosystem flowchart

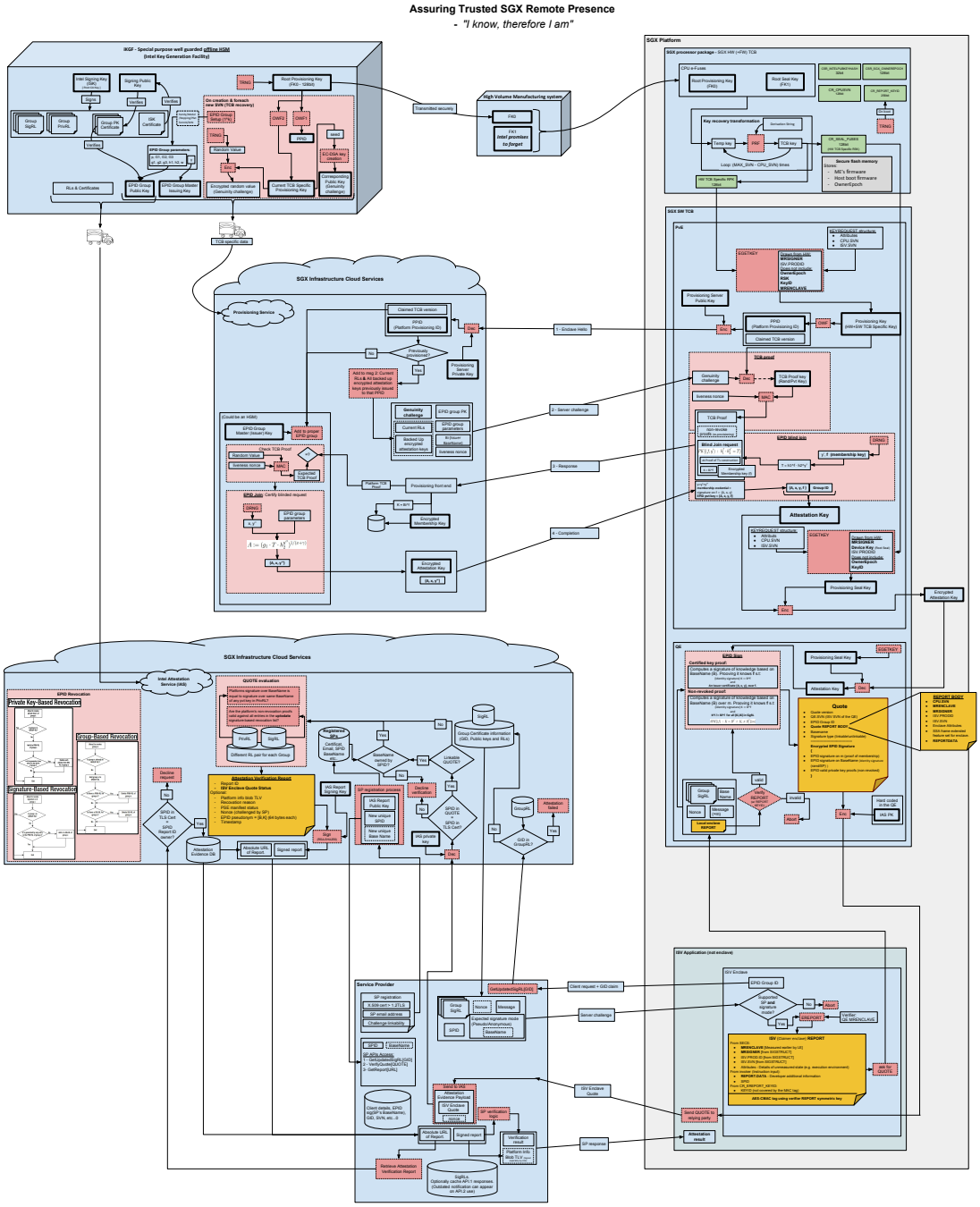

Figure A.1.: Full Flowchart

Bibliography

[1] Intel. SGX Tutorial, ISCA 2015. http://sgxisca.weebly.com/, June 2015.

[2] Intel. Intel Software Guard Extensions Programming Reference (rev2), Oct. 2014. 329298-002US.

[3] Intel. Intel Software Guard Extensions Programming Reference (rev1), Sept. 2013. 329298-001US.

[4] http://www.google.com/patents/US20140093074.

[5] http://www.google.com/patents/US8885819.

[6] Intel Software Guard Extensions: Intel Attestation Service API, version 2. https://software.intel.com/sites/default/files/managed/7e/3b/ias-api-spec.pdf.

[7] Intel SGX Product Licensing https://software.intel.com/en-us/articles/intel-sgx-product-licensing.

[8] Lockheed Martin breach https://www.theguardian.com/world/2009/apr/21/hackers-us-fighter-jet-strike.

[9] Billions of Gemalto embedded keys stolen https://www.theguardian.com/us-news/2015/feb/19/nsa-gchq-sim-card-billions-cellphones-hacking.

[10] Trusted computing group. https://www.trustedcomputinggroup.org/.

[11] L. v. D. P. K. A. Seshadri, A. Perrig. Swatt: software-based attestation for embedded devices. CyLab, Carnegie Mellon Univ., Pittsburgh, PA, USA, 2004. IEEE.

[12] I. Anati, S. Gueron, S. Johnson, and V. Scarlata. Innovative technology for CPU based attestation and sealing. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, volume 13, 2013.

[13] F. Armknecht, A.-R. Sadeghi, S. Schulz, and C. Wachsmann. A security framework for the analysis and design of software attestation. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 1–12, New York, NY, USA, 2013. ACM.

[14] T. W. Arnold and L. van Doorn. The IBM PCIXCC: A new cryptographic coprocessor for the IBM eserver. *IBM Journal of Research and Development*, 48(3-4):475–488, 2004.

[15] M. Barbosa, B. Portela, G. Scerri, and B. Warinschi. Foundations of hardware-based attested computation and application to sgx. Cryptology ePrint Archive, Report 2016/014, 2016. http://eprint.iacr.org/2016/014.

[16] A. Baumann, M. Peinado, and G. Hunt. Shielding applications from an untrusted cloud with haven. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 267–283, Broomfield, CO, Oct. 2014. USENIX Association.

[17] E. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. In *Proceedings of the 11th ACM Conference on Computer and Communications Security*, CCS '04, pages 132–145, New York, NY, USA, 2004. ACM.

[18] E. Brickell and J. Li. Enhanced privacy id from bilinear pairing. Cryptology ePrint Archive, Report 2009/095, 2009. http://eprint.iacr.org/2009/095.

[19] C. Castelluccia, A. Francillon, D. Perito, and C. Soriente. On the difficulty of software-based attestation of embedded devices. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 400–409, New York, NY, USA, 2009. ACM.

[20] Y.-G. Choi, J. Kang, and D. Nyang. Proactive code verification protocol in wireless sensor network. In *Proceedings of the 2007 International Conference on Computational Science and Its Applications - Volume Part II*, ICCSA'07, pages 1085–1096, Berlin, Heidelberg, 2007. Springer-Verlag.

[21] V. Costan and S. Devadas. Intel SGX explained. Cryptology ePrint Archive, Report 2016/086, 2016. http://eprint.iacr.org/.

[22] G. D. Crescenzo. *Equivocable and Extractable Commitment Schemes*, pages 74–87. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[23] C. F. C. G. M. P. G. M.-R. M. R. Felix Schuster, Manuel Costa. Vc3: Trustworthy data analytics in the cloud. Technical report, February 2014.

[24] A. Francillon, Q. Nguyen, K. B. Rasmussen, and G. Tsudik. Systematic treatment of remote attestation. Cryptology ePrint Archive, Report 2012/713, 2012. http://eprint.iacr.org/2012/713.

[25] J. A. Garay and L. Huelsbergen. Software integrity protection using timed executable agents. In *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security*, ASIACCS '06, pages 189–200, New York, NY, USA, 2006. ACM.

[26] R. Gardner, S. Garera, and A. D. Rubin. On the difficulty of validating voting machine software with software. In *Proceedings of the USENIX Workshop on Accurate Electronic Voting Technology*, EVT'07, pages 11–11, Berkeley, CA, USA, 2007. USENIX Association.

[27] O. Goldreich. Candidate one-way functions based on expander graphs. *Electronic Colloquium on Computational Complexity (ECCC*, page 2000.

[28] O. Goldreich. *Foundations of Cryptography: Volume 1.* Cambridge University Press, New York, NY, USA, 2006.

[29] V. Gratzer and D. Naccache. Alien vs. quine. *IEEE Security and Privacy*, 5(2):26–31, Mar. 2007.

[30] L. Gu, X. Ding, R. H. Deng, B. Xie, and H. Mei. Remote attestation on program execution. In *Proceedings of the 3rd ACM Workshop on Scalable Trusted Computing*, STC '08, pages 11–20, New York, NY, USA, 2008. ACM.

[31] S. Gueron. A memory encryption engine suitable for general purpose processors. Cryptology ePrint Archive, Report 2016/204, 2016. `http://eprint.iacr.org/`.

[32] M. Hendry. Smart card security and applications. Artech House, 2001.

[33] M. Hoekstra, R. Lal, P. Pappachan, V. Phegade, and J. Del Cuvillo. Using innovative instructions to create trustworthy software solutions. In *HASP@ ISCA*, page 11, 2013.

[34] Intel. Intel 64 and IA-32 architectures software developer's manual. *Volume 3a: System Programming Guide*, September 2015.

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

[36] M. Jakobsson and K.-A. Johansson. Practical and secure software-based attestation. In *Proceedings of the 2011 Workshop on Lightweight Security & Privacy: Devices, Protocols, and Applications*, LIGHTSEC '11, pages 1–9, Washington, DC, USA, 2011. IEEE Computer Society.

[37] S. Johnson, V. Scarlata, C. Rozas, E. Brickell, and F. Mckeen. Intel software guard extensions: EPID provisioning and attestation services, April 2016.

[38] R. Kennell and L. H. Jamieson. Establishing the genuinity of remote computer systems. In *Proceedings of the 12th Conference on USENIX Security Symposium - Volume 12*, SSYM'03, pages 21–21, Berkeley, CA, USA, 2003. USENIX Association.

[39] S. Kim, Y. Shin, J. Ha, T. Kim, and D. Han. A First Step Towards Leveraging Commodity Trusted Execution Environments for Network Applications. In *Proceedings of the 14th ACM Workshop on Hot Topics in Networks (HotNets)*, Philadelphia, PA, Nov. 2015.

[40] Y. Li, J. McCune, J. Newsome, A. Perrig, B. Baker, and W. Drewry. Minibox: A two-way sandbox for x86 native code. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 409–420, Philadelphia, PA, June 2014. USENIX Association.

[41] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar. Innovative instructions and software model for isolated execution. In *HASP@ ISCA*, page 10, 2013.

[42] M. Naor. On cryptographic assumptions and challenges. In *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 96–109. Springer, 2003.

[43] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In *2008 IEEE Symposium on Security and Privacy (S&P 2008), 18-21 May 2008, Oakland, California, USA*, pages 111–125, 2008.

[44] A. Pfitzmann and M. Hansen. A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management. http://dud.inf.tu-dresden.de/literatur/Anon_Terminology_v0.34.pdf, Aug. 2010. v0.34.

[45] X. Ruan. *Platform Embedded Security Technology Revealed: Safeguarding the Future of Computing with Intel Embedded Security and Management Engine*. Apress, Berkely, CA, USA, 1st edition, 2014.

[46] A. Seshadri, M. Luk, and A. Perrig. Sake: Software attestation for key establishment in sensor networks. *Ad Hoc Netw.*, 9(6):1059–1067, Aug. 2011.

[47] A. Seshadri, M. Luk, A. Perrig, L. van Doorn, and P. Khosla. Scuba: Secure code update by attestation in sensor networks. In *Proceedings of the 5th ACM Workshop on Wireless Security*, WiSe '06, pages 85–94, New York, NY, USA, 2006. ACM.

[48] A. Seshadri, M. Luk, E. Shi, A. Perrig, L. van Doorn, and P. Khosla. Pioneer: Verifying code integrity and enforcing untampered code execution on legacy systems. *SIGOPS Oper. Syst. Rev.*, 39(5):1–16, Oct. 2005.

[49] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla. Using SWATT for verifying embedded systems in cars. In *Proceedings of Embedded Security in Cars Workshop (ESCAR)*, Nov. 2004.

[50] A. Shamir. How to share a secret. In *Communications of the ACM 22, Volume 22 Issue 11*, page 612613, 1979.

[51] M. Shaneck, K. Mahadevan, V. Kher, and Y. Kim. *Remote Software-Based Attestation for Wireless Sensors*, pages 27–41. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[52] E. Shi, A. Perrig, and L. V. Doorn. Bind: A fine-grained attestation service for secure distributed systems. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, SP '05, pages 154–168, Washington, DC, USA, 2005. IEEE Computer Society.

[53] M. Shih, M. Kumar, T. Kim, and A. Gavrilovska. S-NFV: securing NFV states by using SGX. In *Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization, SDN-NFV@CODASPY 2016, New Orleans, LA, USA, March 11, 2016*, pages 45–48, 2016.

[54] C. R. E. B. F. M. Simon Johnson, Vinnie Scarlata. Intel software guard extensions: Epid provisioning and attestation services, 2016. `https://software.intel.com/en-us/blogs/2016/03/09/intel-sgx-epid-provisioning-and-attestation-services`.

[55] L. Sweeney. Achieving k-anonymity privacy protection using generalization and suppression. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):571–588, Oct. 2002.

תקציר:

SGX (Software Guard Extensions) הינה טכנולוגית אבטחת מידע חדשה של אינטל אשר הופעה לראשונה בדור השישי של מעבדי החברה (ארכיטקטורת Skylake). טכנולוגיה זו מבוססת על תוסף חומרתי למעבד אשר מאפשר סביבה להרצה מאובטחת של תוכנה תחת איומים של רכיבים זדוניים של מערכת ההפעלה (כולל חלקי הליבה שלה כגון ה-kernel או hypervisors) וכן מפני טווח מסויים של התקפות פיזיות על חומרת המעבד.

SGX מאפשר למפתחים לספק פתרונות תכנה ברמה גבוהה של אבטחה, בקלות וגמישות יחסית. טכנולוגיה זו בעלת עתיד מבטיח בכינון פתרונות חדשניים עבור מגוון רחב של בעיות אבטחה.

בעבודה זו אנו מציגים תיאור מעמיק של מערכת ה-SGX. אנו מציעים ייצוג המבוסס על "שני עיקרים" אשר מסייע בתפיסת עקרונות האבטחה של המערכת. כמו כן, אנו מדגישים את חשיבותם של יכולות הנעילה (sealing) והעדות (attestation) של הטכנולוגיה ומרחיבים במיוחד אודות יכולת העדות המרוחקת של המעבד.

בהתבסס על התיאור המעמיק שנספק, נמשיך ונבחן את הגבולותיה של הטכנולוגיה בהיבטי זמינות (רציפות תפקודית), פרטיות והנחות האמון שהיא דורשת. כמו כן נסקור את ההשלכות הפוטנציאליות במקרים בהם חלקים שונים מהנחות האמון שמשתמשי המערכת נותנים באינטל מופרים. על אף שבחינה זו מסמנת את קצה גבול אמינותה (תפקודה הראוי) של SGX, היבטים אלו קבלו מעט מאד התייחסות והעמקה עד כה.

לבסוף נציע מספר שיטות, אשר בתנאים מסוימים, "משחררים" את יכולות הנעילה ועדות של SGX מהמגבלותיהם הנוכחיות. נראה כיצד ניתן למזער את בסיס האמון הנדרש ע"י הפרדת ההגנה על המשתמש מזו של תשתית הטכנולוגיה (הנשענת על אינטל). נציג מספר פתרונות אשר מנצלים יכולת נעילה אמינה לטובת בניית יכולת עדות בלתי תלוייה באינטל, ולהפך. לסיום נציע גישה שונה לביסוס אמון, עדות שאיננה תלויה במבחן ידע (קיומו של סוד), אלא במבחן זמן. סכמה חלופית זו מקלה על כמה מהנחות האמון המשמעותיות אשר נדרשות כיום כדי להינות מטכנולוגיית SGX. מלבד ויתור על הנחות אמון שונות, כל פתרון שנציע גם "משחרר" את המשתמש מתלות בניהולה התקין של אינטל את המערכת לטובת זמינות ופרטיות השימוש בה.

עבודתנו מציעה כלים חזקים לבחינה ושימוש מושכל בטכנולוגיות לסביבת הרצה מאובטחת, כמו זו המוצעת ע"י SGX.

עבודה זו בוצעה בהדרכתו של פרופ' ערן טרומר מבית הספר למדעי המחשב, אוניברסיטת תל אביב ודר' טל מורן מבית הספר אפי ארזי למדעי המחשב, המרכז הבינתחומי, הרצליה.

המרכז הבינתחומי בהרצליה
בית-ספר אפי ארזי למדעי המחשב
התכנית לתואר שני (M.Sc.) - מסלול מחקרי

האמון בידי בעל המפתח
הרחבת עצמאות ופרטיות SGX

מאת
אלון ג'קסון

עבודת תזה המוגשת כחלק מהדרישות לשם קבלת תואר מוסמך M.Sc.
במסלול המחקרי בבית ספר אפי ארזי למדעי המחשב, המרכז הבינתחומי הרצליה

מאי 2017