OPERA: Open Remote Attestation for Intel's Secure Enclaves

Guoxing Chen
The Ohio State University
chen.4329@osu.edu

Yinqian Zhang
The Ohio State University
yinqian@cse.ohio-state.edu

Ten-Hwang Lai
The Ohio State University
lai@cse.ohio-state.edu

ABSTRACT

Intel Software Guard Extensions (SGX) remote attestation enables enclaves to authenticate hardware inside which they run, and attest the integrity of their enclave memory to the remote party. To enforce direct control of attestation, Intel mandates attestation to be verified by Intel's attestation service. This Intel-centric attestation model, however, neither protects privacy nor performs efficiently when distributed and frequent attestation is required.

This paper presents OPERA, an Open Platform for Enclave Remote Attestation. Without involving Intel's attestation service while conducting attestation, OPERA is unchained from Intel, although it relies on Intel to establish a chain of trust whose anchor point is the secret rooted in SGX hardware. OPERA is open, as the implementation of its attestation service is completely open, allowing any enclave developer to run her own OPERA service, and its execution is publicly verifiable and hence trustworthy; OPERA is privacy-preserving, as the attestation service does not learn which enclave is being attested or when the attestation takes place; OPERA is performant, as it does not rely on a single-point-of-verification and also reduces the latency of verification.

CCS CONCEPTS

• **Security and privacy → Trusted computing**.

KEYWORDS

Trusted execution environments; Intel SGX; remote attestation; privacy-preserving techniques

ACM Reference Format:
Guoxing Chen, Yinqian Zhang, and Ten-Hwang Lai. 2019. OPERA: Open Remote Attestation for Intel's Secure Enclaves. In *2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19), November 11–15, 2019, London, United Kingdom.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3319535.3354220

1 INTRODUCTION

Intel Software Guard Extensions (SGX) enables a userspace process to create isolated memory regions, called *enclaves*, within which the code and data are protected against accesses from the operating system and other software. Therefore, SGX provides a hardware foundation for supporting security-critical applications even under the assumption of an untrusted operating system or a rogue system administrator. Due to its unique security properties, both academia and industry have paid considerable attention to SGX since its announcement in 2013. So far, various prototype applications (*e.g.*, [3, 17, 28, 29, 33, 40, 41, 47–49]) and development frameworks (*e.g.*, [4, 6, 11, 18, 26, 27, 34, 35, 38, 42, 46]) have been proposed to encourage broader SGX adoption. Recently, SGX has been integrated into commercial products: Microsoft Azure has offered SGX platforms in commercial clouds for confidential cloud computing [31]; a startup company, Fortanix, utilizes SGX to implement hardware security modules and offer runtime encryption solutions [15].

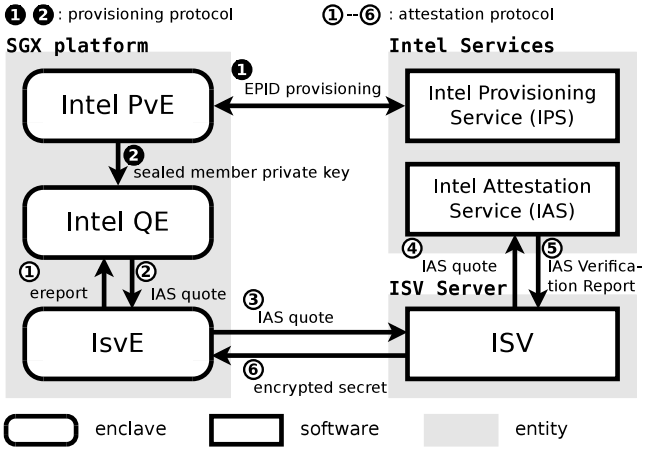


Figure 1: Intel's remote attestation workflow.

While *confidentiality* and *integrity* of Intel SGX are primarily provided by a microcode extension of the memory management unit (MMU) and a hardware Memory Encryption Engine (MEE), remote attestation also plays a crucial role. Remote attestation is a method for software to attest the authenticity of its underlying hardware and the identity of the software to a remote party. Without remote attestation, the trust between SGX enclaves and their users cannot be established. In this paper, we study the issues of trust and privacy of Intel's attestation model, and then propose an open attestation platform as an enhancement for the existing infrastructure.

1.1 Intel's Attestation Service

Intel's attestation model consists of three entities (as shown in Fig. 1): (1) Intel's services, including Intel's provisioning service (IPS) and attestation service (IAS); (2) Independent Software Vendor (ISV)—Intel's term for enclave developers; (3) and the SGX platforms, which run Intel signed enclaves (such as quoting enclaves) and the ISV developed enclave (*i.e.*, IsvE).

Intel SGX adopts the Enhanced Privacy ID (EPID) scheme [10] for remote attestation. EPID uses a group signature that allows an

CCS '19, November 11–15, 2019, London, United Kingdom

ACM ISBN 978-1-4503-6747-9/19/11…$15.00
https://doi.org/10.1145/3319535.3354220

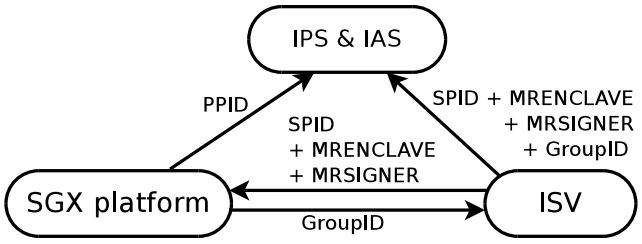


Figure 2: Information exchanges between entities.

SGX platform to generate a signature without leaking the identity of the SGX platform. The EPID scheme involves (1) provisioning the EPID member private key (*i.e.*, attestation key) into the SGX platform through a *provisioning protocol*, and (2) using the EPID member private key to sign the report of the attested enclaves during attestation.

Provisioning protocol. In the EPID provisioning protocol (step ❶ in Fig. 1), the SGX platform first sends a message to IPS that contains the Platform Provisioning ID (PPID) and the trusted computing base (TCB) version. Upon receiving the message, IPS verifies the PPID and *selects* an EPID group for the SGX platform. Intel's current design assigns SGX platforms with the same CPU type and the same TCB version the same EPID group. According to Intel's documentation [22], each EPID group contains millions of machines. IPS then sends back the EPID group public key to the SGX platform. With the group public key, the Intel-signed provisioning enclave (PvE) runs an EPID join protocol with IPS to acquire an EPID member private key, which is sealed in a persistent storage using a seal key, and thus shared with the Intel-signed quoting enclave (step ❷ in Fig. 1).

EPID-based attestation. The EPID-based remote attestation procedure is initiated by the attested enclave (step ①), possibly when requested by the ISV remotely. The attested enclave transfers 64-byte attested data (containing a message or a hash value that the attested enclave would like to associate with the attestation) to the Intel-signed quoting enclave (QE) using a process called *local attestation*. The quoting enclave generates a data structure, called a *quote*, containing the identities of the enclave, including the measurement-based identity MRENCLAVE and the certificate-based identity MRSIGNER, the 64-byte attested data, and a signature of the structure signed with the EPID member private key. The resulting quote is first encrypted with IAS's public key (hardcoded in QE) and then sent back to the ISV (step ② and ③). The ISV forwards the quote to IAS for verification (step ④). The verification results are signed by Intel and encrypted with the ISV's public key before returning to the ISV (step ⑤). Upon a successful attestation, the ISV software provisions secrets into the IsvE to perform confidential computation (step ⑥).

Anonymity considerations. Fig. 2 shows the information exchanges between entities in Intel's attestation model. The primary identifier of an SGX platform is its PPID, which is collected by Intel during the provisioning protocol. PPID uniquely identifies an SGX platform. To have the enclave code executed by the SGX platform, the ISV needs to reveal identity of the enclave signer (*i.e.*, MRSIGNER), the identity of the enclave to be executed (*i.e.*, MRENCLAVE), and the identity of the ISV (*i.e.*, SPID) to the SGX platform. As all these data are included in a quote, they are also collected by IAS as well. The EPID protocol provides a sense of anonymity for SGX platforms during attestation. As only the EPID group identity is included in the quote, neither the ISV nor Intel would learn on which SGX platform did the enclave run.

1.2 Motivation

Instead of allowing ISVs to verify the attestation quotes by themselves, Intel's quoting enclaves encrypt the quotes with Intel's own public key and hence only Intel's attestation service (IAS) is able to decrypt the quotes and verify the attestation. As such, every attestation request must go through IAS to be validated. IAS becomes a trusted third party for ISVs to conduct Internet-based remote attestation.

However, this Intel-centric remote attestation model has several limitations. First, this design leaks sensitive information of the ISVs and enclaves to Intel during verification: (1) which ISV is conducting attestation (through its service provider ID, or SPID) and possibly its IP address (through the TLS connection with IAS), (2) which enclave is being attested (through MRENCLAVE), (3) who signed the enclave (through MRSIGNER), (4) the content of the 64B attested data (embedded in the quote). Therefore, applications that utilize SGX to perform *privacy-sensitive* computation may be hesitant to use Intel's attestation infrastructure. While trusting Intel processors to preserve the privacy of the computation running atop is acceptable in practice, trusting an Internet-facing attestation service that can be compromised by external attackers or rogue insiders would be a different story.

Second, some SGX applications require distributed and frequent attestation. For example, in confidential cloud computing [2, 31], users of these SGX-enabled applications may need to frequently request remote attestation of the SGX applications running in the cloud (*e.g.*, especially in the case of microservice functions, since they are launched and terminated frequently). In privacy-preserving Blockchains that conceal smart contracts in SGX enclaves [13, 50], thousands of nodes may need to perform attestation to verify the computation of others frequently. In these cases, this Intel-centric Internet-based single-point-of-verification model would be insufficient to sustain the high volume of attestation requests. In both examples, moreover, the latency of quote verification (between step ③ to ⑥ in Fig. 1) is too long (as shown in Sec. 5), which negatively impacts the performance of these applications.

Recognizing these limitations, in December 2018, Intel announced a third-party attestation solution [32]. However, this solution is primarily designed as "an on-premise remote attestation capability" that is used for data centers [20]. We will discuss the limitation of Intel's third-party attestation model in Sec. 6.4 when it is used as an Internet-based attestation service, and also discuss how OPERA could help address its limitation. In contrast, in this paper, we aim to design and develop a trusted and publicly verifiable Internet-based attestation platform that are suited for applications to perform distributed, frequent, and privacy-preserving attestation.

1.3 OPERA

In this paper, we propose an Open Platform for Enclave Remote Attestation (OPERA) to address the limitations of the Intel-centric attestation model. It is designed to achieve the following properties:

- *Openness.* The implementation of OPERA's attestation service is completely open; the execution of OPERA is publicly verifiable and thus is trustworthy. Moreover, any ISV could run its own OPERA service.
- *Privacy.* No information regarding the attested enclaves or their developers is leaked to any party other than the SGX platform (who already possesses such information by executing these enclaves).
- *Performant.* The attestation is verified directly by the ISV software or its users, hence eliminating the Internet-based single-point-of-verification and at the same time reducing the latency of verification.

The source code of OPERA is open sourced and available on https://sites.google.com/site/operasgxkb/. We have also discussed with Intel to launch OPERA as a free open service.

Paper outline. We first outline the design of OPERA in Sec. 2 and then detail its design in Sec. 3. The security and privacy guarantees of OPERA are analyzed in Sec. 4. The prototype implementation and its evaluation are described in Sec. 5. We discuss relevant issues in Sec. 6 and related work in Sec. 7, and conclude this paper in Sec. 8.

2 AN OPEN ATTESTATION PLATFORM

In this section, we provide a high-level overview of Open Platform for Enclave Remote Attestation (OPERA), an open remote attestation platform for SGX enclaves. Details of its design will be provided in later sections.

2.1 Threat Model

Intel's services, including IPS and IAS, are trusted to honestly follow the predefined protocols, but may stealthily collect information of the SGX platforms, the ISVs, and the enclaves being attested, during the remote attestation. As the root secrets of SGX platforms are controlled by Intel, we must trust the tech-giant to faithfully establish a chain of trust to identify SGX platforms and their protected software. We also assume that there is no backdoor installed on SGX platforms that could leak any secrets to Intel; firewalls could effectively prevent connections to Intel servers other than the provisioning and attestation traffic.

We assume the SGX platforms, including the entire software stack outside the enclaves, are untrusted, although they are responsible for launching enclaves, supporting their execution and external communication. However, we assume that Intel SGX is secure. Though recently disclosed hardware vulnerabilities have rendered SGX vulnerable to some speculative execution based attacks [12, 43, 44], we assume the confidentiality of the enclaves is guaranteed if the CPU microcode is up-to-date. As such, our remote attestation platform must be able to detect outdated CPU microcode versions, similar to what is offered by IAS. We assume the SGX platforms are operated by a party that is not motivated to collude with Intel.

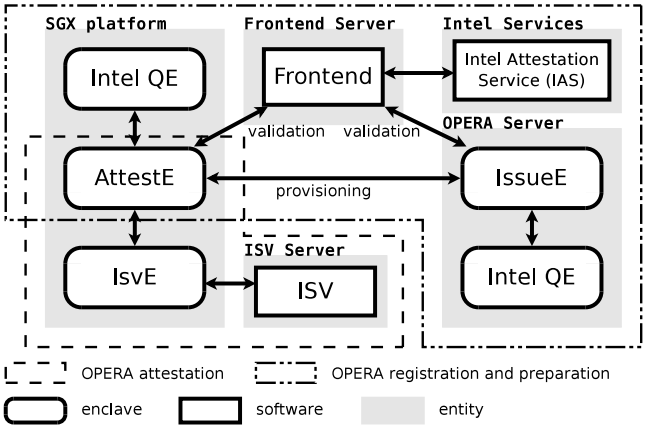


Figure 3: Design overview

OPERA is also untrusted, because it is an open platform that allows anyone to participate. However, we assume OPERA can be publicly verified. The open platform publishes its enclave programs to be hosted either on the SGX platforms or public servers (which are also untrusted). We assume the source code and binaries of all OPERA-provided enclaves can be publicly verified to only perform their specified execution. The design and verification of these enclaves is out of scope. We assume these enclaves can be trusted after performing remote attestation.

Although the primary goal of our design is to preserve the privacy of enclaves and their ISVs during attestation, our design will also preserve the anonymity of the SGX platforms to ISVs, so that ISVs cannot identify on which machine their enclaves are executed from the attestation signatures.

2.2 Design Overview

Components. The overview of OPERA is illustrated in Fig. 3. The main functionality of OPERA is handled by two types of enclaves: the issuing enclaves (`IssueE`) and the attestation enclaves (`AttestE`). An `IssueE` runs on a publicly-accessible server equipped with an SGX processor (*e.g.*, a cloud SGX server). We call such servers OPERA Servers. To preserve the anonymity of the SGX platforms, OPERA adopts the same group signature schemes used by IAS, *i.e.*, the EPID scheme. An `IssueE` is responsible for generating an EPID group issuing key, which is used to issue member keys to `AttestE`s and to produce group verification certificates (`gvCert`). To leverage IAS to validate the authenticity of the `IssueE`, the ISV that launches the `IssueE` must be registered with Intel and the attestation must go through these ISVs. We call an OPERA-participating ISV an OPERA-SV. From Intel's point of view, each OPERA-SV is no different than other ISVs.

The `AttestE` is launched on the SGX platform that requires enclave attestation services. It communicates with the `IssueE` to obtain a unique EPID member private key. The `IssueE` conducts a remote attestation to validate the authenticity of `AttestE`. Then the `AttestE` serves the local attestation requests from the ISV enclaves (called `IsvE`s) and produces quotes (called OPERA quotes) that are signed by its EPID private key. At least one `AttestE` is required

to run on the same SGX platform with the IsvE to be attested; multiple AttestEs can be launched to work with the same IssueE.

Each OPERA-SV must run its own frontend server to communicate with IAS. We discuss in Sec. 6 how frontend servers can be hosted by untrusted parties. The authenticity of both IssueEs and AttestEs are guaranteed by the original Intel attestation procedure, with the help of the OPERA-SV frontend server.

Operations. OPERA operates in three steps: *registration*, *preparation*, and *attestation*. The *registration* step launches, validates, and registers an IssueE to OPERA; the *preparation* step launches an AttestE on one SGX platform, validates its authenticity, and provisions the EPID private key into the AttestE by running a provisioning protocol with an IssueE; the *attestation* step attests the authenticity of an IsvE (*i.e.*, ISV enclave) that run on the same machine as the AttestE. The validation of IssueE and AttestE needs to be run periodically to check the validity of the SGX platform using IAS and establish trustworthiness of these enclaves.

The source code and the binaries of IssueEs and AttestEs are published so that they can be scrutinized to perform the exact desired actions, nothing less and more. For example, the only network activities from an AttestE is through a provisioning protocol or through a self-validation procedure. As long as the code follows these protocols and nothing more, it can be assured that no information regarding the ISVs or the SGX platforms will be collected by OPERA. Furthermore, as each IsvE communicates only with the AttestE, the ISV could assure that its identity will neither be leaked to Intel.

To enable the ISVs to verify the authenticity of IsvE, AttestE, and IssueE directly from the OPERA quotes, OPERA adopts the concept of certificates and accompanies the quotes with a chain of certificates, including an EPID group certificate (gvCert) generated by IssueE that endorse the authenticity of AttestE and a certificate-type structure from IAS that endorses the validity of IssueE. SGX platform's anonymity is preserved by adopting the EPID scheme, as the ISVs cannot differentiate one SGX platform from another by looking at their EPID signatures. Additional cares must be taken for the anonymity guarantees, however, which we will detail in Sec. 3.

3 DESIGN OF OPERA

In this section, we detail the design of the open attestation service. In Sec. 3.1, we describe the *registration* step for setting up one IssueE to OPERA; in Sec. 3.2, we describe *preparation* step for setting up one AttestE on an SGX platform; in Sec. 3.3, we describe the *attestation* step for an ISV enclave; in Sec. 3.4, we explain the revocation process.

3.1 Registration

This step sets up an IssueE and registers it with OPERA. Then a validation procedure is executed to generate a fresh group verification certificate (gvCert) for the IssueE. The validation procedure is executed periodically (*e.g.*, daily) to guarantee that the security level of the SGX processor it runs upon is up-to-date.

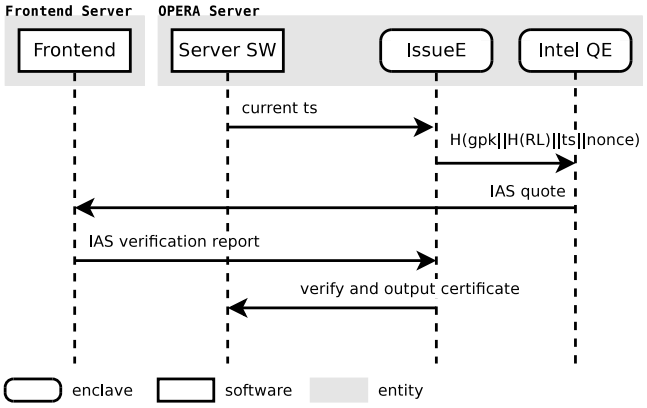


Figure 4: Validation protocol of IssueE

IssueE setup. An IssueE is launched on an untrusted OPERA server. After launching, the IssueE first generates inside the enclave memory an issuing private key and the corresponding group public key. As the issuing private key is protected by SGX, the OPERA Server cannot learn its content. A copy of the issuing private key will be sealed for backup. An issuing private key represents an EPID group and is used to produce EPID member private keys for each AttestE. An IssueE also creates two empty revocation lists during its setup: one for private key revocation and the other for signature-based revocation as described in Intel's documentation [22].

IssueE validation. After setup, the IssueE generates a group verification certificate (gvCert) for the EPID group. The certificate should include the group public key, revocation lists, and a cryptographic proof of their authenticity. OPERA achieves this using Intel SGX's remote attestation with the help of a frontend server run by an OPERA-SV. The IssueE uses Intel's quoting enclave to generate an IAS quote with the hash of the group public key and revocation lists as the report data, and sends the IAS quote to the frontend software to be forwarded to IAS. The IAS verification report (returned from IAS and signed by Intel) can be used as a proof of the authenticity of the public key and the revocation lists. The certificate expires after a short period, *e.g.*, a day [24], so that the IssueE needs to periodically re-validate itself and produce a fresh gvCert. IAS checks if the Intel-issued EPID private key has been revoked or if the CPU is up-to-date, which is essential to the security of OPERA. Therefore, periodic re-validation is important.

Since OPERA servers are not trusted, we need to be careful when designing the validation protocol. Basically, we need a reliable way, *e.g.*, a timestamp to represent the freshness of a gvCert. Note that the IAS verification report contains a timestamp from the IAS, which can be used as the timestamp of the gvCert. However, IAS quotes used to produce IAS verification reports do not include any time information. It is possible for the OPERA server to send an outdated IAS quote to IAS to obtain an IAS verification report from Intel with a new timestamp, so that the OPERA server might be able to refresh an expired gvCert without interacting the IssueE.

One intuitive solution is to maintain an internal timestamp inside IssueE, and include the timestamp in the gvCert. The IssueE could use valid IAS verification reports to update this timestamp.

Table 1: Structure of group verification certificate (gvCert).

Name	Description
Group public key	The public key of the EPID group for verifying EPID signatures
H(PrivRl)	Hash value of private key revocation list
H(SigRl)	Hash value of signature based revocation list
IssueE timestamp	A timestamp provided by the OPERA server and verified by IssueE
Nonce	A nonce generated within IssueE
IAS verification report	Intel signed structure whose report data is the hash of the above five entries, and whose timestamp should be within the same interval of the IssueE timestamp.

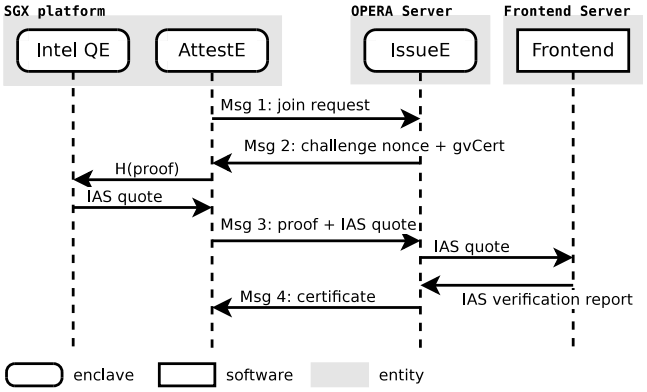


Figure 5: OPERA provisioning flow

However, such a design requires two rounds of Intel's remote attestation for each validation, one for updating the timestamp and the other for generating the gvCert with the updated timestamp.

Hence, we adopted a *commit-and-reveal* method which requires only one round of Intel's remote attestation per validation. The protocol for updating the certificate is shown in Fig. 4. Specifically, when requesting a new certificate,

- The OPERA Server software first passes a current (untrusted) timestamp as input to the IssueE.
- The IssueE then picks a random nonce, calculates the hash value of the group public key, hash values of the revocation lists, the timestamp provided by OPERA server, and the generated nonce. The hash value will then be used to generate a new IAS quote to be sent to IAS.
- After receiving the IAS verification report, IssueE checks the timestamp of the IAS verification report and timestamp given by the OPERA Server software. If their difference is small, *e.g.*, less than 5 seconds, IssueE reveals the nonce and outputs the verification certificate.

The timestamp of the IAS verification report is used as the timestamp of the gvCert. Here the revealed nonce enforces that the timestamp of an acceptable IAS verification report should be close (with a small difference) to the IssueE timestamp present in the gvCert. The structure of the certificate is shown in Table 1.

3.2 Preparation

This step sets up an AttestE on an SGX platform. The validation procedure is executed periodically (*e.g.*, daily) to ensure that the SGX processor is up-to-date.

AttestE setup. An AttestE will be launched on an SGX platform. It will then run a provisioning protocol (as illustrated in Fig. 5) with an IssueE:

- The AttestE sends to the OPERA server an EPID provisioning request (*Msg* 1).
- After receiving the request, IssueE generates a nonce, and then sends it back to AttestE together with gvCert (*Msg* 2).
- The AttestE verifies gvCert, and generates an EPID private key using the nonce received in *Msg* 2. A proof of knowledge [10] is generated to prove that AttestE does possess the private key. The hash value of the proof will be used as the report data to generate an IAS quote. This quote and the proof itself will be sent back to IssueE (*Msg* 3).
- After receiving *Msg* 3, IssueE sends the quote to the OPERA-SV frontend, which forwards the quote to IAS. The returned IAS verification report can be used to authenticate to IssueE that the AttestE runs on trusted SGX CPU and its measurement matches the expected value (hardcoded in IssueE). IssueE calculates a certificate of the EPID private key and sends it back to AttestE (*Msg* 4). The provisioning protocol concludes.

AttestE validation. Similar to IssueE, AttestE also needs to periodically perform an attestation with IAS to validate the authenticity of the underlying SGX platform. Each attestation is performed using a fresh nonce as its report data. The AttestE maintains an internal timestamp to record the most recent time its environment is verified. After receiving the IAS verification report corresponding to the same nonce, AttestE will update the timestamp using the one from the report. This timestamp will be sent to the ISV during attestation to inform the ISV about the time of most recent validation. To prevent the resolution of the timestamp from being abused by ISVs to de-anonymize the SGX platform, the timestamp is discretized (*e.g.*, to date, when the validation is performed daily).

The SGX trusted platform service provides a rich set of functionality for enclaves running on the SGX platform. Although AttestE itself does not use any such services, it records the status of the SGX trusted platform services, *e.g.*, whether the services are up-to-date, so that it is able to provide such information to the attested enclaves that use the services. In addition, the SGX platform also keeps an up-to-date gvCert and revocation lists that are retrieved periodically from the IssueE.

3.3 Attestation

Once the *registration* and *preparation* steps are completed, the SGX platform can allow ISVs to leverage OPERA for remote attestation. When an IsvE needs to attest itself to the ISV or a third party (possibly upon requests), it first generates a report, with a message it would like to communicate with the ISV as the report data, to be transferred to the AttestE via local attestation. The AttestE

Table 2: The structure of the OPERA quote

Name	Description
IsvE's report	The report generated by the IsvE
AttestE timestamp	The timestamp maintained by AttestE
pseManifest status	The status of the SGX trusted platform service
EPID signature	EPID signature of the above three entries

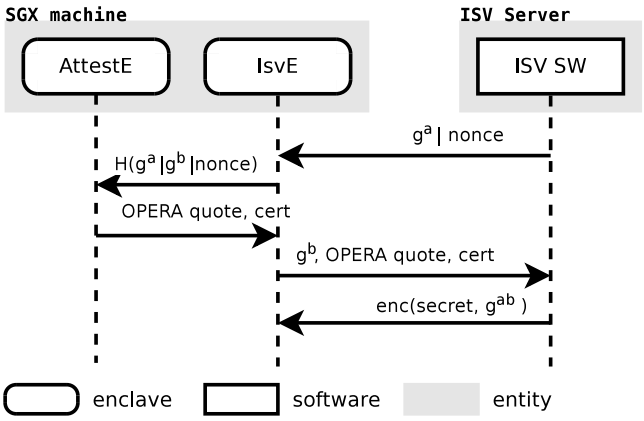


Figure 6: OPERA attestation flow

verifies the IsvE's report and then signs a data structure containing the report, the timestamp of AttestE, and the status of the SGX trusted platform service, using its EPID private key. The resulting quote is called OPERA quote. The structure of the OPERA quote is shown in Table 2.

The OPERA quote, the gvCert, and the revocation lists are sent back to the ISV. The ISV then verifies that (1) the gvCert is valid—the IAS verification report is properly signed by Intel and reflects the integrity of the gvCert; (2) the EPID signature is verified to be valid; (3) the timestamps of the IAS verification report, IssueE and AttestE are up-to-date; (4) The measurements of the IssueE and IsvE are valid; (5) Optionally, the pseManifest status is valid; (6) IsvE's report reflects the integrity of the attestation data. After the verification, the remote party could be assured that the report data is indeed generated by the specific IsvE running on a trusted SGX platform.

We describe an example attestation scheme to demonstrate how to use OPERA to derive a shared Elliptic Curve Diffie-Hellman (ECDH) key between an ISV and its IsvE. The attestation workflow is illustrated in Fig. 6.

Specifically, the attestation is accomplished in the following four steps:

- The ISV software first generates a nonce and an ECDH private/public key pair (a, g^a), and then sends the nonce and ECDH public key g^a to the IsvE launched on a SGX platform.
- The IsvE generates an ECDH private/public key pair (b, g^b), and derives the shared ECDH key g^{ab} following standard ECDH protocol, from its own ECDH private key b and the received ECDH public key g^a such that $g^{ab} = (g^a)^b$. The hash value of the received ECDH public key g^a, its own ECDH public key g^b, and the nonce is used as the report data to generate a report and send it to the AttestE.
- The AttestE verifies the IsvE's report and then generates an OPERA quote. Then the OPERA quote, the ECDH public key g^b, gvCert, and the revocation lists are sent back to the ISV software.
- The ISV software verifies (1) the gvCert, (2) the EPID signature, (3) the timestamps of the IAS verification report, IssueE and AttestE, (4) The measurements of the IssueE and IsvE, (5) (Optionally) The pseManifest status, (6) ISV enclave's report, and (7) that the decrypted nonce using the ECDH shared key derived from its ECDH private key and the received ECDH public key matches the nonce the ISV client generated previously. After the verification, the ISV software could provision secrets to IsvE using a channel encrypted by the ECDH key g^{ab}.

3.4 Revocation

The revocation scheme follows the same manner as the original EPID scheme [22]. One difference between the revocation schemes of OPERA and IAS is that the latter keeps encrypted backup copies of the EPID member private keys and disallows a previously revoked SGX platform from getting new EPID keys. Here, there is no need for OPERA to keep such backup copies, because OPERA relies on IAS to verify the revocation status of an SGX platform. When the SGX platform is revoked, it could not host any IssueE nor AttestE, since the revoked SGX platform will be identified during setup and/or validation processes where valid IAS verification reports are needed.

4 ANALYSIS OF SECURITY AND PRIVACY

In this section, we analyze the security and privacy of OPERA. To bootstrap the analysis, we assume that Intel SGX is not broken and the confidentiality of the enclaves are guaranteed if the CPU is up-to-date. Though recently discovered hardware vulnerabilities have rendered SGX vulnerable to some speculative and out-of-order execution based side-channel attacks [12, 43], we anticipate these vulnerabilities will be addressed eventually with future microcode patches. We also assume the code of IssueE and AttestE are scrutinized and verified so that they behave as expected and do not leak extra information to unauthorized parties.

4.1 Proof of Security

Given our assumption that up-to-date SGX processors protect the confidentiality of their hosted enclaves, we first need to show the protocols of OPERA can verify that the CPUs on which IssueEs and AttestEs run are up-to-date. OPERA relies on the IssueE to verify the trustworthiness of the SGX platform running the AttestE. Specifically, during member private key provisioning, the AttestE will need to provide an IAS quote, with which the IssueE could verify the identity of the requesting AttestE and trustworthiness of the SGX platform running the AttestE. Hence, the IssueE will only provision the member private key to specific AttestEs (specified by its measurement that is hardcoded in its binary) running on a up-to-date CPU. Therefore, the ISV only needs to verify the IssueE is running on an up-to-date SGX platform. Verifying this

$M, N ::=$	terms
x, y, z	variables
a, b, c, k, s	names
$f(M_1, \ldots, M_n)$	constructor application
$D ::=$	expressions
M	term
$h(D_1, \ldots, D_n)$	function application
fail	failure
$P, Q ::=$	processes
0	nil
$\mathsf{out}(N, M); P$	output
$\mathsf{in}(N, x : T); P$	input
$P \mid Q$	parallel composition
$!P$	replication
new $a : T; P$	restriction
let $x : T = D$ in P else Q	expression evaluation
if M then P else Q	conditional

Figure 7: Syntax of ProVerif [8]

is straightforward since an IAS verification report (indicating the identity of the IssueE and the status of the SGX platform running the IssueE) of the group verification certificate is also received by the ISV. Therefore, it is verifiable by the ISV that secrets in the IssueE and the AttestE are well-protected within the enclaves.

Next, we prove that the ISV could verify the identity and integrity of the IsvE using OPERA, and then provision secrets to IsvE which would not be leaked to the adversary. The proof is conducted automatically using ProVerif, an automated tool for analyzing symbolic protocol models. ProVerif uses the π-calculus to model the protocols and to verify a variety of security properties, such as secrecy and authentication. ProVerif [8] takes as input protocols and security properties to be verified, and translates the protocols into Horn clauses. The security properties are translated into derivability queries on these clauses. An example query is whether the attacker could derive a secret given her existing knowledge. If the verification results indicating the queries are not derivable from the clauses, then the desired security properties are proven. Otherwise, a potential attack might be found.

The syntax of the input language of the ProVerif is shown in Fig. 7. *Terms* represent data and messages, *expressions* represent computations on terms and *processes* represent programs. In ProVerif, functions symbols are used to represent constructors and destructors. Constructors are used for building terms. For example, a symmetric encryption scheme can be represented using a constructor senc, and the ciphertext of a message m under a symmetric key k can be represented as the term $\mathsf{senc}(m, k)$. On the other hand, destructors are used for manipulating terms in expressions. The semantic of a destructor $g(M_1, \ldots, M_l)$ are represented as a set of rewrite rules $g(M_1, \ldots, M_l) \rightarrow M'$, where $M_1, \ldots, M_l, M'$ are constructors or variables. For instance, the decryption algorithm of a symmetric encryption scheme can be represented as a destructor sdec, defined by a rewrite rule: $\mathsf{sdec}(\mathsf{senc}(m, k), k) \rightarrow m$.

Each protocol role, (*e.g.*, client or server), can be represented as a process which can send and receive messages (terms) over public channels. Following the Dolev-Yao model [14], ProVerif assumes the protocol is run in the presence of an attacker, who can intercept, synthesize and send any message she has. The keys and nonces generated within a process is considered fresh and unguessable.

To use ProVerif to prove the security of OPERA, we model each enclave (*e.g.*, IssueE and AttestE) as a process. The external world of the enclaves, including the untrusted OS and system programs, is modeled as a Dolev-Yao attacker who cannot directly access the secrets within the enclaves, but handles all communications between enclaves.

However, it is not straightforward to model SGX enclaves. One challenge is to model the SGX instructions that take the calling enclave's measurement as the input. Particularly, the EGETKEY and EREPORT instructions are used for local attestation. As an enclave's measurements are also available to the adversary, the adversary could call EGETKEY within an enclave developed by herself. However, the SGX mechanism prevents EGETKEY using a measurement other than the calling enclave's to derive keys. This mechanism must be modeled properly in our proof. Therefore, we assign a secret, called SGX specification to each type of enclave, which will be used as input when calling enclave specific functions, *e.g.*, EGETKEY, EREPORT. The measurement is also calculated from the secret, and will then be made public. Specifically, we use a constructor $\mathsf{mr}(\mathsf{sgxsp}) : \mathsf{mrenc}$ (where sgxsp is the type of SGX specification secret, and mrenc is the type of enclave measurement) to calculate the enclave measurement. To represent EGETKEY (particularly for deriving report keys) and EREPORT we use two constructors $\mathsf{egetrpkey}(\mathsf{sgxsp}) : \mathsf{rpkey}$ and $\mathsf{ereport}(\mathsf{sgxsp}, \mathsf{mrenc}, \mathsf{bitstring}) : \mathsf{bitstring}$, where rpkey is the type of report key. $\mathsf{egetrpkey}(s)$ derives the report key of the enclave with its SGX specification secrete s. $\mathsf{ereport}(s, r, m)$ generates a report (with message m) from an enclave with SGX specification s to an enclave whose measurement is r. The corresponding destructors $\mathsf{egetreport}(\mathsf{bitstring}) : \mathsf{bitstring}$, $\mathsf{egetmrfrom}(\mathsf{bitstring}) : \mathsf{mrenc}$ and $\mathsf{everif}(\mathsf{bitstring}, \mathsf{rpkey}) : \mathsf{bitstring}$ are defined by:

$$\mathsf{egetreport}(\mathsf{ereport}(s, r, m)) \rightarrow m$$
$$\mathsf{egetmrfrom}(\mathsf{ereport}(s, r, m)) \rightarrow \mathsf{mr}(s)$$
$$\mathsf{everif}(\mathsf{ereport}(s_a, \mathsf{mr}(s_b), m), \mathsf{egetrpkey}(s_b)) \rightarrow m$$

egetreport is used by any entity (*e.g.*, the adversary) that obtains the report to access the report data since EREPORT protects only the integrity of the report data, but not the confidentiality. egetmrfrom is used to figure out the identity, *i.e.*, the measurement of the enclave who generates the report. everif is used by an enclave to verify that whether the report is meant to be sent to her.

Now we can begin to model our framework:

- Intel's quoting enclave is modeled as a process that takes a report, verifies it via everif. An EPID signature will be generated if the verification passes.
- IssueE is modeled as a process that receives EPID joining requests from another enclave and provisions private member keys to that enclave if its measurement matches AttestE's measurement.
- AttestE is modeled as a process that achieves two tasks: first is to run the EPID joining protocol with IssueE to obtain a

private EPID member key, and second is to take a report from an attesting enclave, verify it and sign it accordingly.
- An example IsvE and an example ISV client are modeled as two separate processes that derive an ECDH shared key through a remote attestation using the OPERA quote generated by the AttestE. The example ISV client then sends a secret encrypted using the derived shared key to the example IsvE.

These 5 processes are trusted. The measurements of these enclaves will be made public by sending them to a public channel. With these 5 processes, we modeled the OPERA provisioning protocol, which involves Intel's quoting enclave, IssueE and AttestE, and the OPERA attestation protocol, which involves AttestE, IsvE and the ISV client. The OPERA provisioning protocol is run first to provision AttestE a member private key. Then IsvE and the ISV client run a remote attestation with the help of AttestE. After the remote attestation, a shared key will be derived by both IsvE and the ISV client. The ISV client then sends to the IsvE a secret that is encrypted using the shared key.

The security property to be proved here is the secrecy of the secret the ISV client tries to provision to IsvE. It is represented as a query whether the adversary could derive the secret given that the adversary could access any publicly available information such as enclaves' measurements, manipulate messages exchanged between trusted processes, run her own enclaves, *etc*. ProVerif evaluated the modeled processes and protocols and concluded that the adversary could not derive the secret, indicating that the secret can be securely provisioned to a trusted IsvE. The related code is listed in Appendix A.

4.2 Privacy Analysis

We consider the privacy of OPERA-SVs, IsvEs, and SGX platforms:

4.2.1 OPERA-SV *Privacy.* The OPERA enclaves (*i.e.*, IssueEs and AttestEs) both use IAS to establish their trustworthiness, an OPERA-SV must register itself to Intel. Hence, during runtime, its own identity, *i.e.*, the SPID, will be revealed to Intel. Moreover, the measurements of the OPERA enclaves will also be revealed to Intel since they are included in IAS quotes.

4.2.2 IsvE *Privacy.* IAS is not involved in the attestation process of the IsvE. Moreover, OPERA Servers and IssueEs are not involved in the attestation other than providing the gvCerts. Therefore, as long as the AttestE code is verified such that it does not collect ISV's information, the IsvE' identity could be hidden from both Intel and OPERA. However, SGX platforms will learn the identity of the IsvEs since as the enclave metadata must be known by the host to launch an enclave. This is the minimal privacy leakage the ISV has to accept to run enclaves on a third-party SGX platform.

4.2.3 SGX platform Privacy. Note that the gvCert is completely generated in the OPERA Server. AttestE provides only its own maintained timestamp, which should be indicating the current time interval. Hence, in the OPERA quote sent back to the ISV, there is no extra information specific to the underlying SGX platform. We further assume that other identity information of SGX platform, *e.g.* IP address, can be concealed via network proxies. As such the SGX platform remains anonymous to the ISV.

5 IMPLEMENTATION AND EVALUATION

5.1 Implementation

Our implemented OPERA prototype consists of three components: the software running on OPERA Servers, including the IssueE, the software running on SGX platforms, including the AttestE, and the OPERA-SV frontend. In our implementation, we adopted Intel EPID SDK (version 6.0.0) that comes along with Intel SGX SDK (version 2.5.101.50123), which is released under Apache License version 2.0. Intel EPID SDK provides implementations of the EPID join protocol, signing and verification procedures. But Intel does not open source the implementation of the EPID issuer. Therefore, we implemented our own EPID issuer for the IssueE.

5.1.1 OPERA *Server Software and* IssueE. OPERA Server software is responsible for launching IssueE and handling the communication between the IssueE and its external world. Since the IssueE is stateful (including the revocation lists), it might be a target of replay attacks [27]. Hence, we leveraged the SGX platform trusted service, particularly, the monotonic counter, to prevent replay attacks. For IssueE setup, we implemented three ECall functions:

- create_issuer() creates a new issuer context, *i.e.*, creating an issuing key and its corresponding group public key, and initializing two revocation lists. An issuer context represents an EPID group. A *replay protected policy* is created to track the status of the revocation lists, by registering a monotonic counter with SGX trusted platform service. The monotonic counter's ID and its current value will be sealed together with the data it protects, *i.e.*, the issuing private key, the group public key, and the hash values of the revocation lists. The complete revocation lists are maintained by the OPERA Server software and verified by the IssueE when needed.
- load_issuer() loads an existing issuer context sealed by itself previously. The replay protected policy is verified: First, the IssueE extracts the monotonic counter's ID and queries SGX trusted platform service to get the value of that counter. If the counter value is the same as the one sealed previously, the sealed issuer context is trusted and restored. Then, the IssueE checks the hash values of the revocation lists against those extracted from the sealed data to preserve the integrity of the lists.
- delete_issuer() is used to delete the issuer context, *i.e.*, revoke the EPID group. The corresponding monotonic counter will also be deleted.

Note that when deriving the seal key, the IssueE will use its enclave measurement MRENCLAVE as input, so that the sealed data could only be unsealed and accessed by the IssueE, preventing other enclaves signed by the same OPERA-SV from accessing the issuing key.

After the IssueE either creates or loads an issuer context, IssueE validation process needs to be performed. We implemented two ECall functions for this task:

- request_gvcert() takes the current timestamp as input, generates a nonce, and outputs a report with the hash of the group verification certificate as the report data, along with the pseManifest. The OPERA Server software asks Intel's quoting enclave to sign the report and sends the resulting quote and pseManifest to IAS via an OPERA-SV frontend for verification.

- produce_gvcert() takes the returned IAS verification report, the corresponding report signature, and the report signing certificate chain as input, checks its validity, and outputs a gvCert accordingly. The root certificate, *i.e.*, the Attestation Report Signing CA Certificate, is hard-coded into the enclave.

We also implemented two ECall functions to support the EPID provisioning process between the IssueE and the AttestE as described in Sec. 3.2: gen_nonce() generates and outputs a nonce, *i.e.*, *Msg 2*, upon receiving a join request, *i.e.*, *Msg 1*. certify_member() takes as input *Msg 3*, and output *Msg 4*. We implemented two ECall functions for revocation: revoke_priv() and revoke_sig() are used for revoking compromised EPID member private keys and signatures linked with abnormal behaviors. When either revocation list is updated, the monotonic counter will be increased. The updated issuer context and monotonic counter value will be re-sealed to reflect this change.

5.1.2 SGX Platform Software and AttestE*.* Applications running on the SGX platform are responsible for launching AttestE and handling the communication between the AttestE and its external world. To support the EPID provisioning process between the IssueE and the AttestE as described in Sec. 3.2, we implemented two ECall functions in AttestE: join_request() takes as input the nonce received from *Msg 2*, and generates *Msg 3*. create_member() takes as input the *Msg 4* and finalize the creation of a new member private key. Another two ECall functions, seal_member() and unseal_member(), were implemented for saving and restoring existing member private keys. Note that the seal key is derived using the AttestE's enclave measurement MRENCLAVE, in order to prevent other enclaves signed by the same OPERA-SV from accessing the sealed secrets.

Two ECall functions were developed for AttestE validation (*i.e.*, updating its internal timestamp):

- ts_update_request() generates a nonce, and outputs a report and pseManifest accordingly. The SGX platform software asks Intel's quoting enclave to sign the report and sends the resulting quote and pseManifest to IAS via an OPERA-SV frontend for verification.
- ts_update_response() verifies the returned IAS verification report, report signature and report signing certificate chain, and update its timestamp accordingly.

For OPERA attestation, we implemented an ECall function, opera_gen_quote(), to generate OPERA quote for remote attestation. Particularly, it takes as input a report provided by an attested enclave, and signs a data blob including the attested enclave's report, its internal timestamp and pseManifest status using the member private key. The resulting OPERA quote, along with the gvCert and revocation lists will be returned for proving the identities of the attested enclaves and integrity of the report data.

5.1.3 OPERA-SV *Frontend Software.* The OPERA-SV frontend is implemented using cURL to communicate with IAS to verify IAS quotes. We implemented one function for this task: get_ias_report() receives IAS quotes and optionally pseManifest from either IssueEs or AttestEs, forwards them to IAS, receives the IAS verification reports to be returned to the requesting OPERA enclaves. Communication between OPERA-SV frontend server and other servers is implemented using Internet sockets.

Table 3: The latency evaluation

EPID SDK	Latency (ms)
EpidSignBasic	5.09
EpidVerifyBasicSig	13.77
Intel's Remote Attestation	**Latency (ms)**
sgx_get_quote	8.46
send quote to IAS and receive report	195.25
Open Remote Attestation	**Latency (ms)**
get OPERA quote via IPC socket	5.17
verify OPERA quote	13.81

5.1.4 IsvE *development tools.* We implemented tools for IsvE developers to adopt OPERA. Specifically, we implemented a function generate_report() which can be called within the IsvE to generate a report of attesting message to be signed later. We implemented a function opera_get_quote() to be used by IsvE host program to communicate with AttestE for generating OPERA quote. On the ISV client side, we implemented a function opera_verify_quote() for OPERA quote verification.

5.1.5 Example IsvE *and ISV client.* We also implemented an example IsvE to demonstrate how to use OPERA. The IsvE has one ECall function, gen_report() which generates a report. An OPERA quote is generated by its host program and sent along with the report to the SGX platform software using inter-process communication mechanisms (*e.g.*, IPC socket).

With the returned OPERA quote and the group verification certificate, we implemented two example ISV clients (one written in C++ and the other in Python) which can verify the validity of the OPERA quote and determine the identity the IsvE and the integrity of its report data.

5.2 Evaluation

In this section, we evaluate the performance of OPERA. All experiments were conducted on a Dell Latitude 5480 laptop equipped with an Intel Core i7-7820HQ processor and 8GB memory. The CPU has 4 physical cores and 8 logical cores. Wi-Fi is used for communication between different servers and access to IAS. Evaluation was conducted with empty revocation lists.

Latency. We first measured the latencies of different operations. The results shown in Table 3 are average measurements over 10,000 runs.

To set up a baseline for our evaluation, we first measured the execution time of the two basic EPID operations that were implemented in Intel SGX SDK, *i.e.*, EpidSignBasic() and EpidVerifyBasicSig(). The former is used to sign an enclave quote and generate an EPID signature, and the latter is used to verify an EPID signature. Both these operations are computationally intensive. By running the two operations 10,000 times and calculating the average, we found the execution time of the two operations were

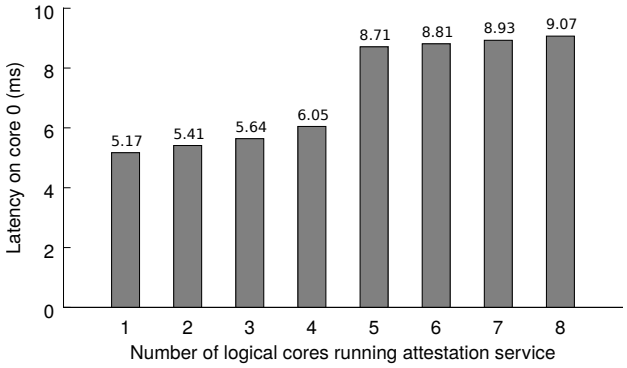


Figure 8: Latency of OPERA attestation with multiple AttestEs

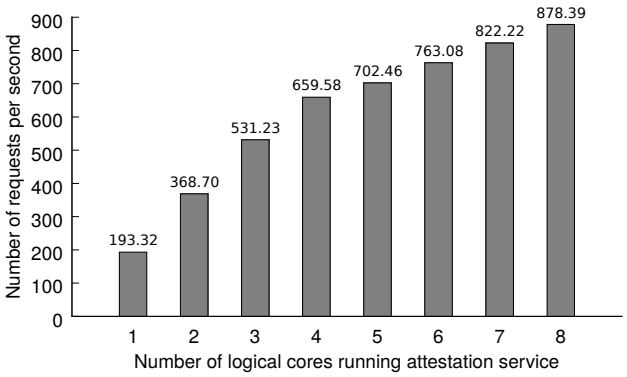


Figure 9: Throughput of OPERA attestation

Table 4: OPERA overhead

	IssueE	AttestE
One-time setup	279.05ms	397.01ms
Per re-validation	219.17ms	376.10ms

5.09ms and 13.77ms, respectively. We then measured the latency of Intel's Remote Attestation operations as a control experiment. sgx_get_quote() is an API provided by Intel SGX SDK for generating a quote using Intel's quoting enclave. Its execution time is around 8.46ms (average over 10,000 runs). We also measured the time taken to send the signed quote to IAS and receive the verification report. The average latency (including the network latency) was 195.25ms.

We then measured the latency of our implemented OPERA. First of all, the latency of obtaining an OPERA quote, which measures the time it takes to send a report generated by an IsvE to AttestE and receive an OPERA quote through an IPC socket, is 5.17ms (over 10,000 runs). The latency is very close to that of EpidSignBasic() (*i.e.*, 5.09ms). It means the majority of the time is consumed by the EpidSignBasic() operations, and the overhead of inter-process communication is by contrast extremely small.

We also evaluated the latency of attestation with OPERA when using multiple AttestEs on the same machine. Particularly, we increase the number of AttestEs from 1 to 8, and assigned these AttestEs to run on logical CPU core 0, core 1, ..., core 7 in order. Each logical core i ($i \in [0, 1, 2, 3]$) is on the same physical core as core $i + 4$. In Fig. 8, we show the attestation latency of the AttestE running on core 0 and how the latency is affected by running more AttestEs on other CPU cores concurrently. It can be seen that resource contention due to Hyper-Threading is much more significant than contention from other physical cores. Nevertheless, OPERA can be parallelized with multiple AttestEs on the same machine (up to the number of logical cores) to improve its scalability.

Next, the latency of verifying the received OPERA quote, which involves verification of a chain of certificates, is around 13.81ms. Compared with the execution time of EpidVerifyBasicSig(), which is 13.77ms, we can see that the group signature verification is dominating the verification latency.

Compared with the attestation with IAS, OPERA attestation is significantly faster (5.17 + 13.81ms vs. 8.46 + 195.25ms). To make the comparison fair, it is worthy noting that Intel's remote attestation involves not only an extra round-trip network communication, but also reconstructing an EPID context after unsealing the EPID private key and a verification of the EPID signature at IAS. However, these overheads are unnecessary for frequent attestation requests, thus can be avoided by OPERA.

Throughput. We measured the number of attestation requests per second that could be processed by OPERA on the same SGX platform. Intuitively, the more logical cores running AttestEs that provide attestation services concurrently, the higher the throughput. The results are shown in Fig. 9. The throughput ranges from 193.32 requests per second to 878.39 requests per second. In comparison, sgx_get_quote() generates 118.18 IAS quotes per second[1]. The throughput of OPERA can be much higher than that of IAS.

Overhead. We evaluated the overhead of our implemented OPERA. The overhead consists of two parts: one time setup overhead and periodic re-validation overhead. The results shown in Table 4 are average measurements over 1,000 runs. For IssueE, we measured an averaged setup time of 279.05ms, consisting of generating a new EPID issuing key and the corresponding group public key, and registering a monotonic counter for the replay protected policy. To measure the validation time, *i.e.*, generating a refreshed gvCert, we run a frontend on the same machine as the IssueE to access IAS. The averaged time for IssueE validation is 219.17ms.

To measure the overhead of setting up AttestE, we run an IssueE and a frontend on another laptop, a Lenovo Thinkpad X1 Carbon (4th Gen) laptop with an Intel Core i5-6200U processor and 8GB memory. The setup time of an AttestE was 397.01ms, including running the provisioning protocol with the IssueE and one round of Intel's remote attestation via the frontend. The periodic re-validation of AttestE includes retrieving a refreshed gvCert and one round of Intel's remote attestation. We measured an averaged validation time of 376.10ms.

[1]This result is measured with only one quoting enclave. As Intel's Architectural Enclave Service Manager (AESM) implementation launches only one quoting enclave, which is ensured to be singled threaded (the number of TCS of the quoting enclave is set to 1), hence its throughput would not increase with more logical cores.

While the setup overhead is incurred only once and thus not critical, the periodic re-validation cost depends on the re-validation frequency. For example, when the re-validation is needed per hour, the overhead could be insignificant ($\approx 0.01\%$).

Other Evaluations. IssueE contains 1705 lines of code (including the EPID issuer implemented by ourselves) and AttestE contains 780 lines of code. The binary sizes of an IssueE and an AttestE are 1.08MB and 1.09MB, respectively. Multiple IssueEs and AttestEs could be launched on the same machine (even with 128MB EPC).

6 DISCUSSION

6.1 Application Scenarios

OPERA is a generic attestation framework that fits both centralized and decentralized attestation model.

OPERA can support centralized attestation as in the case of confidential cloud computing [2, 31] and privacy-preserving blockchains [13, 50], the cloud provider or the blockchain operator may serve as the OPERA-SV and launch IssueEs and AttestEs for the hosted applications. Users of such cloud services may use OPERA so they do not even need to use IAS. Scalability of attestation and privacy of the execution can be provided by OPERA. As SGX requires enclaves to be launched with a valid MRSIGNER, the hosted applications may request the cloud provider or blockchain operator to sign the applications and run with the providers' MRSIGNER. Alternatively, if the SGX platforms support Flexible Launch Control (FLC) feature [19], the providers may implement customized launch enclaves to allow the hosted applications to run without registering with Intel.

OPERA can also be implemented to support decentralized attestation, in which enclave programs on the same SGX platform attest one another. When an ISV launches her enclave on a SGX platform and requests remote attestation through OPERA. If there is no AttestE already launched on that SGX platform, the ISV could launch an AttestE herself to provide attestation service to her own enclaves and other enclaves to be launched later on the SGX platform. The AttestE will search for an IssueE to join. If no IssueE is found, the ISV may launch an IssueE somewhere to support her own AttestE and others' AttestEs. Since the source code of OPERA enclaves are publicly available, and the measurements are calculated unrelated to who signs and/or launches the enclaves. Therefore, the IssueE setup by one ISV would allow AttestEs from other ISVs to join.

6.2 Alternative Design Options

Supporting multiple IssueEs. In our design, if the OPERA Server that runs IssueE goes offline, all associated AttestEs can only provide attestation service till the current group public key certificate expires (*e.g.*, after a day or a month). In our current design, OPERA may launch more than one AttestEs that are associated with different IssueEs on each SGX platform. In this way, an SGX platform may always find a working AttestE to conduct attestation. An alternative design is to launch IssueEs that share the same issuing key on multiple OPERA Servers to eliminate single-point-of-failure. To do so, these IssueEs must authenticate each other through IAS attestation process to verify their identities and synchronize the issuing keys and revocation lists among all these IssueEs.

Running OPERA-SV frontends in enclaves. OPERA may also opt to run OPERA-SV frontends in enclaves. There are two reasons for this alternative design choice: First, as the IAS verification reports are encrypted by IAS with the OPERA-SV's public key, the OPERA-SV frontend must be involved to decrypt the quotes. As periodic validation of IssueEs and AttestEs are required, the OPERA-SV frontend must be always online. Second, during the periodic validation, these OPERA-SV frontends may collect information regarding the setup and validation of both IssueEs and AttestEs. While privacy of IssueEs and AttestEs are less important in general (as they are public enclaves), however, under certain circumstances, such as decentralized attestation schemes, users of OPERA may place less trust on the OPERA-SV than Intel.

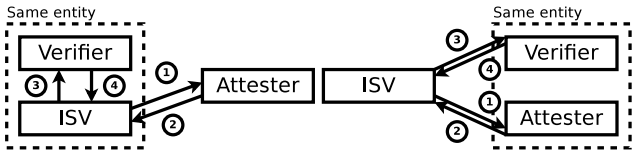
To address this, OPERA-SV can ship the decryption functionality in enclaves (dubbed FrontE) and run the FrontE on the same OPERA Server as IssueE. The FrontE could attest itself (using IAS) to other OPERA enclaves to derive shared keys for secure communications between these enclaves. The EPID private key can be provisioned into the FrontE after attestation. Hence, the OPERA Servers do not need to be trusted with OPERA-SV's private keys. Alternatively, FrontE may first generate a public/private key pair inside the enclave and then expose the certificate to the outside. In this way, even the OPERA-SV could not participate in the validation process of IssueEs and AttestEs. Thus, privacy of these enclaves can be preserved.

OPERA on AMD's Secure Encrypted Virtualization (SEV). OPERA is primarily designed for Intel' SGX platforms. However, we do not see major challenges of adopting OPERA on other trusted execution environments, such as AMD's SEV [23]. However, as the time of writing, we could not obtain the source code to perform remote attestation for SEV-enabled virtual machines. Neither is AMD's attestation service available to us. We leave extension of OPERA to other platforms to the future work.

6.3 Dependence on Intel

OPERA still relies on Intel (IPS and IAS) to establish the root of trust on the underlying SGX platforms. As analyzed by by Francillon *et al.* [16], a remote attestation protocol requires exclusive accesses to a long-term key for generating attestation signatures. For Intel SGX, as its manufacturer, Intel is the only entity that could embed such long-term secrets into SGX processors, and retain exclusive accesses to these secrets. Hence, OPERA, and any other alternative attestation services, must rely on Intel to bootstrap the trust on the SGX platform.

Intel may decide to ban IssueE and AttestE. However, in its current design, Intel could only ban OPERA-SVs but not individual enclaves. This is because enclaves' launch control is enforced in the launch enclave using a white list of MRSIGNERs (corresponding to ISVs), and using IAS's service requires the ISV's registered SPID and certificate. Therefore, Intel could only ban the OPERA-SVs that run IssueE and AttestE. With the formation of an open attestation framework, OPERA will invite a large number of ISVs to participate in the attestation service, and thus preventing Intel from banning all participating OPERA-SVs. Banning individual enclaves instead of ISVs would introduce considerable changes to the current design of SGX.



(a) ISV performs verification. (b) Verifier = Attester.

Figure 10: Third-party attestation models. ①ISV sends an attestation request the the Attester (SGX platform); ②Attester sends the attestation report to ISV; ③ISV requests Verifier to verify the attestation report; ④Verifier responds to ISV.

Nevertheless, given the benefit of security, privacy, and usability, instead of suppressing OPERA, Intel will benefit by supporting it. OPERA may attract more ISVs with particular security and/or privacy requirements, *e.g.*, confidential cloud computing and privacy-preserving blockchains, to adopt SGX for development. We are discussing with Intel to gain support from the tech-giant to OPERA.

6.4 Intel's Third-Party Attestation Scheme

Concurrent to our work, Intel announced Data Center Attestation Primitives (DCAP) to support third-party attestation in on-premise network, such as data centers [32]. In such a scheme, third-party quoting enclaves can be used instead of Intel's quoting enclave. A pair of attestation keys used by the third-party quoting enclave are generated from inside, using algorithms such as Elliptic Curve Digital Signature Algorithm (ECDSA). The public key is exported to Intel's Provisioning Certification Enclave (PCE) and signed using the private Provisioning Certification Key (PCK). PCK is a pair of asymmetric keys derived from the root provisioning secret burnt into the SGX platform's e-fuses.

To establish a chain of trust, Intel needs to publish PCK certificates for users to verify the public key of PCKs, which can be used to verify the ECDSA public key certificate generated by the PCE. Therefore, a user with knowledge of Intel's public key, PCK certificates, and ECDSA public key certificates, can verify the ECDSA attestation result all by herself (*i.e.*, the quote signed by the ECDSA private key). Similar to OPERA, the revocation list of PCKs still needs to be periodically retrieved from Intel (and thus the third-party attestation service still relies on Intel).

However, what is not made clear by Intel is where the verification should be performed [32]. Depending on whether the PCK certificates could be disclosed to public, two verification models can exist (Fig. 10): First, when the certificates can be made publicly available (Fig. 10a), the attestation results could be verified locally by the users without trusting any third party. However, as a PCK certificate can uniquely identify an SGX platform [32], such a design leaks the identity of attester, *i.e.*, the SGX platform that runs the enclave application, which can be a privacy concern—this is also the motivation of adopting EPID in SGX original attestation scheme. In the case of confidential cloud computing, users of public clouds may use such information to perform co-location attacks [30, 45].

Second, when the certificates are only accessible by the service provider (Fig. 10b), to use the third-party attestation service, users have to fully trust the third-party attestation service. As argued by Swami [39], collusion between the attester and the verifier could completely invalidate the attestation results. Specifically, if the attestation service provider is the same as the operator of the data center—which is typically the case—enclave programs can be launched on non-SGX machines but still pass the remote attestation.

In comparison, OPERA addresses the disadvantages of both designs. On one hand, OPERA adopts EPID schemes to protect SGX platforms' privacy. On the other hand, OPERA leverages Intel SGX to exclude the third party from the trusted domain. Therefore, we envision OPERA could be used in cases where Internet-based attestation service is required, such as privacy-preserving permissionless blockchain [13, 50], confidential public clouds [2, 31], *etc.*

Build OPERA on top of DCAP. It is also possible to integrate OPERA and DCAP together. The idea is to replace the functionality of IAS with DCAP. Note that IAS is used to (1) prove AttestE's identity to IssueE and (2) prove IssueE's identity to the ISV. For the first purpose, instead of sending an IAS quote, AttestE sends DCAP quote and the corresponding PCK certificate and ECDSA public key certificate to IssueE. IssueE could then verify AttestE's identity and provision private EPID member key. Note that the PCK certificate and ECDSA public key certificate of the SGX platform will not be recorded by IssueE nor forwarded to IsvE so that SGX platform privacy can be reserved. For the second purpose, instead of appending IAS verification report in the gvCert, IssueE appends the PCK certificate and ECDSA public key certificate of the OPERA Server in gvCert. In this way, IssueE's identity can be verified by the ISV, and only the OPERA Server's identity is disclosed.

7 RELATED WORK

Intel SGX adopts the EPID scheme which is a group signature with additional security goals such as anonymity [9], and member revocation [10]. Some of its security guarantees, however, have been questioned by Swami [39]. In this paper, we further explore problems in SGX attestation and propose an open attestation platform to address the privacy and performance issues.

To adapt SGX in clouds, VC3 proposes to use a Cloud Quoting Enclave to generate an additional attestation signature for the attested enclaves, in order to prove that an SGX platform is not only authentic but indeed in the possession of the cloud provider [33]. However, this Cloud Attestation scheme does not replace Intel's SGX attestation, but rather merely complements it. VC3 could extend OPERA (by offering a provider specific signature) to replace its combined use of SGX Attestation and Cloud Attestation.

SafeKeeper [25] and ReplicaTEE [37] addressed a related problem to ours, i.e., enabling unregistered users to use IAS. They achieved this by forwarding quotes to IAS via a registered proxy. However, this design does not address the privacy issues considered in our paper, as IAS could still collect information from the forwarded quotes. Concurrent to our work, Alder *et al.* [1] proposed a transitive attestation protocol in the context of Function-as-a-Service clouds, enabling the users to verify the worker enclaves by attesting only a centralized key distributing enclave. Despite this technical similarity, however, their protocol targets a specific application, while ours aims to provide a generic attestation service.

Knauth *et al.* [24] from Intel proposed to use Intel's remote attestation to generate X.509 certificates for TLS communication. They used IAS verification reports to prove the validity of the certificates, which is similar to how `IssueE` creates `gvCert`. Moreover, Intel's Data Center Attestation Primitives also enables third parties to conduct attestation services [32]. However, as discussed in Sec. 6.4, their scheme has limitations and does not replace `OPERA`.

Also related to our work is due to Jackson [21], who proposed several alternative attestation schemes to address similar attestation problems. However, the schemes were still rudimentary; the security of the proposed schemes were not formally analyzed, and no proof-of-concept implementation was provided.

Prior to our work, `ProVerif` [8] has been used to verify security properties for various protocols [5, 7]. In the context of SGX, `ProVerif` has also be used for verification of confidentiality properties of enclave programs [36].

8 CONCLUSION

This paper presents `OPERA`, an Open Platform for Enclave Remote Attestation for SGX applications to perform distributed, frequent, and privacy-preserving remote attestation. The advantages of `OPERA`, in comparison to Intel's attestation service, are its openness, improved privacy and performance. We analyzed and proved the security and privacy guarantees of `OPERA`, implemented a prototype, and evaluated its performance.

ACKNOWLEDGMENTS

The work was supported in part by the NSF grants 1718084 and 1750809, and research gifts from Intel and Dfinity.

REFERENCES

[1] Fritz Alder, N. Asokan, Arseny Kurnikov, Andrew Paverd, and Michael Steiner. 2018. S-FaaS: Trustworthy and Accountable Function-as-a-Service using Intel SGX. *CoRR* abs/1810.06080 (2018). arXiv:1810.06080 http://arxiv.org/abs/1810.06080

[2] Alicloud. 2018. ECS Bare Metal Instance. https://www.alibabacloud.com/product/ebm.

[3] Ittai Anati, Shay Gueron, Simon P Johnson, and Vincent R Scarlata. 2013. Innovative Technology for CPU Based Attestation and Sealing. In *2nd International Workshop on Hardware and Architectural Support for Security and Privacy*. ACM.

[4] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'Keeffe, Mark L. Stillwell, David Goltzsche, Dave Eyers, Rüdiger Kapitza, Peter Pietzuch, and Christof Fetzer. 2016. SCONE: Secure Linux Containers with Intel SGX. In *12th USENIX Symposium on Operating Systems Design and Implementation*. USENIX Association.

[5] Michael Backes, Matteo Maffei, and Dominique Unruh. [n.d.]. Zero-Knowledge in the Applied Pi-calculus and Automated Verification of the Direct Anonymous Attestation Protocol. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy*. IEEE Computer Society, Washington, DC, USA.

[6] Andrew Baumann, Marcus Peinado, and Galen Hunt. 2014. Shielding Applications from an Untrusted Cloud with Haven. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. USENIX Association, Broomfield, CO.

[7] K. Bhargavan, B. Blanchet, and N. Kobeissi. 2017. Verified Models and Reference Implementations for the TLS 1.3 Standard Candidate. In *2017 IEEE Symposium on Security and Privacy (SP)*.

[8] Bruno Blanchet. 2016. Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif. *Foundations and Trends in Privacy and Security* 1, 1–2 (Oct. 2016), 1–135.

[9] Ernie Brickell, Jan Camenisch, and Liqun Chen. 2004. Direct Anonymous Attestation. In *Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS '04)*. ACM, New York, NY, USA.

[10] E. Brickell and J. Li. 2010. Enhanced Privacy ID from Bilinear Pairing for Hardware Authentication and Attestation. In *IEEE Second International Conference on Social Computing*. 768–775.

[11] Chia che Tsai, Donald E. Porter, and Mona Vij. 2017. Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. USENIX Association, Santa Clara, CA.

[12] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H. Lai. 2019. Stealing Intel Secrets from SGX Enclaves via Speculative Execution. In *Proceedings of the 2019 IEEE European Symposium on Security and Privacy*.

[13] Raymond Cheng, Fan Zhang, Jernej Kos, Warren He, Nicholas Hynes, Noah M. Johnson, Ari Juels, Andrew Miller, and Dawn Song. 2019. Ekiden: A Platform for Confidentiality-Preserving, Trustworthy, and Performant Smart Contract Execution. In *Proceedings of the 2019 IEEE European Symposium on Security and Privacy*.

[14] D. Dolev and A. C. Yao. 1981. On the Security of Public Key Protocols. In *Proceedings of the 22nd Annual Symposium on Foundations of Computer Science (SFCS '81)*. IEEE Computer Society, Washington, DC, USA.

[15] Fortanix. 2018. Runtime Encryption with Intel SGX. https://fortanix.com/.

[16] Aurélien Francillon, Quan Nguyen, Kasper B. Rasmussen, and Gene Tsudik. 2014. A Minimalist Approach to Remote Attestation. In *Proceedings of the Conference on Design, Automation & Test in Europe (DATE '14)*. European Design and Automation Association, 3001 Leuven, Belgium, Belgium.

[17] Matthew Hoekstra, Reshma Lal, Pradeep Pappachan, Vinay Phegade, and Juan Del Cuvillo. 2013. Using Innovative Instructions to Create Trustworthy Software Solutions. In *2nd International Workshop on Hardware and Architectural Support for Security and Privacy*. ACM.

[18] Tyler Hunt, Zhiting Zhu, Yuanzhong Xu, Simon Peter, and Emmett Witchel. 2016. Ryoan: A Distributed Sandbox for Untrusted Computation on Secret Data. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association.

[19] Intel. 2017. Intel 64 and IA-32 Architectures Software Developer's Manual, Combined Volumes:1,2A,2B,2C,3A,3B,3C and 3D. https://software.intel.com/sites/default/files/managed/39/c5/325462-sdm-vol-1-2abcd-3abcd.pdf.

[20] Intel. 2018. Intel Software Guard Extensions (Intel SGX) Services. https://api.portal.trustedservices.intel.com/.

[21] Alon Jackson. 2017. Trust is in the Keys of the Beholder : Extending SGX Autonomy and Anonymity. *M.Sc. dissertation. Interdisciplinary Center, Herzliya.* (2017).

[22] Simon Johnson, Vinnie Scarlata, Carlos Rozas, Ernie Brickell, and Frank Mckeen. 2016. *Intel Software Guard Extensions: EPID Provisioning and Attestation Services*. Technical Report. Intel, Tech. Rep. https://software.intel.com/sites/default/files/managed/57/0e/ww10-2016-sgx-provisioning-and-attestation-final.pdf.

[23] David Kaplan, Jeremy Powell, and Tom Woller. 2016. AMD memory encryption. White paper. https://developer.amd.com/wordpress/media/2013/12/AMD_Memory_Encryption_Whitepaper_v7-Public.pdf.

[24] Thomas Knauth, Michael Steiner, Somnath Chakrabarti, Li Lei, Cedric Xing, and Mona Vij. 2018. Integrating Remote Attestation with Transport Layer Security. *CoRR* abs/1801.05863 (2018). arXiv:1801.05863 http://arxiv.org/abs/1801.05863

[25] Arseny Kurnikov, Klaudia Krawiecka, Andrew Paverd, Mohammad Mannan, and N. Asokan. 2018. Using SafeKeeper to Protect Web Passwords. In *Companion Proceedings of the The Web Conference 2018 (WWW '18)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland.

[26] Dmitrii Kuvaiskii, Oleksii Oleksenko, Sergei Arnautov, Bohdan Trach, Pramod Bhatotia, Pascal Felber, and Christof Fetzer. 2017. SGXBOUNDS: Memory Safety for Shielded Execution. In *12th European Conference on Computer Systems*. ACM.

[27] Sinisa Matetic, Mansoor Ahmed, Kari Kostiainen, Aritra Dhar, David Sommer, Arthur Gervais, Ari Juels, and Srdjan Capkun. 2017. ROTE: Rollback Protection for Trusted Execution. In *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC.

[28] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday Savagaonkar. 2013. Innovative Instructions and Software Model for Isolated Execution. In *2nd International Workshop on Hardware and Architectural Support for Security and Privacy*. ACM.

[29] Olga Ohrimenko, Felix Schuster, Cedric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. 2016. Oblivious Multi-Party Machine Learning on Trusted Processors. In *25th USENIX Security Symposium*. USENIX Association.

[30] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. 2009. Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM conference on Computer and communications security*. ACM.

[31] Mark Russinovich. 2017. Introducing Azure confidential computing. https://azure.microsoft.com/en-us/blog/introducing-azure-confidential-computing/.

[32] Vinnie Scarlata, Simon Johnson, James Beaney, and Piotr Zmijewski. 2018. Supporting Third Party Attestation for Intel SGX with Intel Data Center Attestation Primitives. (2018). https://software.intel.com/sites/default/files/managed/f1/b8/intel-sgx-support-for-third-party-attestation.pdf

[33] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich. 2015. VC3: Trustworthy Data Analytics in the Cloud Using SGX.

In *36th IEEE Symposium on Security and Privacy*.
[34] Jaebaek Seo, Byoungyoung Lee, Seongmin Kim, Ming-Wei Shih, Insik Shin, Dongsu Han, and Taesoo Kim. 2017. SGX-Shield: Enabling Address Space Layout Randomization for SGX Programs. In *The Network and Distributed System Security Symposium*.
[35] Shweta Shinde, Dat Le Tien, Shruti Tople, and Prateek Saxena. 2017. Panoply: Low-TCB Linux Applications With SGX Enclaves. In *The Network and Distributed System Security Symposium*.
[36] Rohit Sinha, Sriram Rajamani, Sanjit Seshia, and Kapil Vaswani. 2015. Moat: Verifying Confidentiality of Enclave Programs. In *22nd ACM Conference on Computer and Communications Security*. ACM.
[37] Claudio Soriente, Ghassan Karame, Wenting Li, and Sergey Fedorov. 2019. ReplicaTEE: Enabling Seamless Replication of SGX Enclaves in the Cloud. In *Proceedings of the 2019 IEEE European Symposium on Security and Privacy*.
[38] Raoul Strackx and Frank Piessens. 2016. Ariadne: A Minimal Approach to State Continuity. In *25th USENIX Security Symposium*. USENIX Association.
[39] Yogesh Swami. 2017. SGX Remote Attestation is not Sufficient. Cryptology ePrint Archive, Report 2017/736. https://eprint.iacr.org/2017/736.
[40] Sandeep Tamrakar, Jian Liu, Andrew Paverd, Jan-Erik Ekberg, Benny Pinkas, and N. Asokan. 2017. The Circle Game: Scalable Private Membership Test Using Trusted Hardware. In *ACM on Asia Conference on Computer and Communications Security*. ACM.
[41] Florian Tramer, Fan Zhang, Huang Lin, Jean-Pierre Hubaux, Ari Juels, and Elaine Shi. 2017. Sealed-Glass Proofs: Using Transparent Enclaves to Prove and Sell Knowledge. In *Proceedings of the 2017 IEEE European Symposium on Security and Privacy*.
[42] D. Tychalas, N. G. Tsoutsos, and M. Maniatakos. 2017. SGXCrypter: IP protection for portable executables using Intel's SGX technology. In *22nd Asia and South Pacific Design Automation Conference*.
[43] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. 2018. Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, Baltimore, MD.
[44] Stephan van Schaik, Alyssa Milburn, Sebastian ÃŰsterlund, Pietro Frigo, Giorgi Maisuradze, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. 2019. RIDL: Rogue In-flight Data Load. In *40th IEEE Symposium on Security and Privacy*.
[45] Venkatanathan Varadarajan, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. 2015. A Placement Vulnerability Study in Multi-Tenant Public Clouds. In *USENIX Security Symposium*.
[46] Samuel Weiser and Mario Werner. 2017. SGXIO: Generic Trusted I/O Path for Intel SGX. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy (CODASPY '17)*. ACM, New York, NY, USA.
[47] Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. 2016. Town Crier: An Authenticated Data Feed for Smart Contracts. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*. ACM, New York, NY, USA.
[48] Fan Zhang, Ittay Eyal, Robert Escriva, Ari Juels, and Robbert Van Renesse. 2017. REM: Resource-Efficient Mining for Blockchains. In *26th USENIX Security Symposium*. USENIX Association.
[49] Wenting Zheng, Ankur Dave, Jethro G. Beekman, Raluca Ada Popa, Joseph E. Gonzalez, and Ion Stoica. 2017. Opaque: An Oblivious and Encrypted Distributed Analytics Platform. In *14th USENIX Symposium on Networked Systems Design and Implementation*. USENIX Association.
[50] Guy Zyskind, Oz Nathan, and Alex Pentland. 2015. Enigma: Decentralized Computation Platform with Guaranteed Privacy. *CoRR* abs/1506.03471 (2015). arXiv:1506.03471 http://arxiv.org/abs/1506.03471

A PROVERIF CODE

The ProVerif code for the security proof is listed below.

```
(* EPID scheme *)

type epidikey.
type epidvkey.

fun epidvk(epidikey): epidvkey.

type epidmkey.
type epidpkow.
type epidcert.

fun epidpk(epidmkey, bitstring): epidpkow.
fun epidct(epidpkow, epidikey, bitstring): epidcert.
fun epidsign(bitstring, epidmkey, epidcert): bitstring.
reduc forall m: bitstring, k: epidmkey, c: epidcert;
  getepidmsg(epidsign(m, k, c)) = m.
reduc forall m: bitstring, mk: epidmkey, ik: epidikey, nonce: bitstring;
 checkepidsign(
  epidsign(m, mk, epidct(epidpk(mk, nonce), ik, nonce)),
  epidvk(ik)
 ) = m.

fun vk2bs(epidvkey): bitstring.
reduc forall vk: epidvkey; bs2vk(vk2bs(vk)) = vk.

fun pk2bs(epidpkow): bitstring.
reduc forall pk: epidpkow; bs2pk(pk2bs(pk)) = pk.

fun ct2bs(epidcert): bitstring.
reduc forall ct: epidcert; bs2ct(ct2bs(ct)) = ct.

(* Symmetric encryption *)
type sekey.

fun senc(bitstring, sekey): bitstring.
reduc forall m: bitstring, k: sekey;
  sdec(senc(m, k), k) = m.

(* Diffie-Hellman *)
type G.
type exponent.
const g: G [data].
fun exp(G, exponent): G.
equation forall x: exponent, y: exponent;
  exp(exp(g, x), y) = exp(exp(g, y), x).

fun G2sek(G): sekey.
fun G2bs(G): bitstring.
reduc forall gxy: G; bs2G(G2bs(gxy)) = gxy.

(* SGX related *)
type sgxsp.
type mrenc.
type rpkey.

fun mr(sgxsp): mrenc.

fun egetrpkey(sgxsp): rpkey.
fun ereport(sgxsp, mrenc, bitstring): bitstring.
reduc forall sFrom: sgxsp, rTo: mrenc, m: bitstring;
  egetreport(ereport(sFrom, rTo, m)) = m.
reduc forall sFrom: sgxsp, rTo: mrenc, m: bitstring;
  egetmrfrom(ereport(sFrom, rTo, m)) = mr(sFrom).
reduc forall sFrom: sgxsp, sTo: sgxsp, m: bitstring;
  everif(ereport(sFrom, mr(sTo), m), egetrpkey(sTo)) = m.

(* free names *)
free ae_ie: channel.
free ie_qe: channel.
free ae_qe: channel.
free la: channel.
free ra: channel.
free c: channel.

free epidikIntel: epidikey [private].

free sgxIntelQE: sgxsp [private].
free sgxIssueE: sgxsp [private].
free sgxAttestE: sgxsp [private].
free sgxIsvE: sgxsp [private].

free s: bitstring [private].

(* query *)
query attacker(s).

(* protocol *)

let IntelQE(sp: sgxsp, mk: epidmkey, ct: epidcert, la: channel) =
  in(la, erpt: bitstring);
  let rpt = everif(erpt, egetrpkey(sp)) in
  let mrf = egetmrfrom(erpt) in
  out(la, epidsign((mrf, rpt), mk, ct)).

let IssueE(sp: sgxsp, epidik: epidikey, aem: mrenc,
```

```
  qem: mrenc, qec: channel, ae: channel) =

  (* msg 1 *)
  in(ae, req: bitstring);

  (* get IAS quote of cert *)
  let vk = epidvk(epidik) in
  out(qec, ereport(sp, qem, vk2bs(vk)));
  in(qec, cert: bitstring);

  new nonce: bitstring;

  (* msg 2 *)
  out(ae, (nonce, cert));

  (* msg 3 *)
  in(ae, quotepk: bitstring);
  let (mrf: mrenc, pk: bitstring)
= checkepidsign(quotepk, epidvk(epidikIntel)) in

  (* security check: verify the identity of AttestE *)
  if mrf = aem then

  (* msg 4 *)
  out(ae, epidct(bs2pk(pk), epidik, nonce)).

let AttestE(sp: sgxsp, qem: mrenc, qec: channel,
  ie: channel, la: channel) =

  (* msg 1 *)
  new req: bitstring;
  out(ie, req);

  (* msg 2 *)
  in(ie, (nonce: bitstring, cert: bitstring));
  new mk: epidmkey;
  let pk = epidpk(mk, nonce) in

  (* get IAS quote *)
  out(qec, ereport(sp, qem, pk2bs(pk)));
  in(qec, quotepk: bitstring);

  (* msg 3 *)
  out(ie, quotepk);

  (* msg 4 *)
  in(ie, ct: bitstring);

  (* remote attesation *)
  in(la, erpt: bitstring);
  let rpt = everif(erpt, egetrpkey(sp)) in
  let mrf = egetmrfrom(erpt) in
  out(la, (epidsign((mrf, rpt), mk, bs2ct(ct)), cert)).


let IsvE(sp: sgxsp, aem: mrenc, la: channel, ra: channel) =

  (* rcv isv challenge *)
  in(ra, (gx: G, nonce: bitstring));
  new y: exponent;
  let dhsk = G2sek(exp(gx, y)) in
  let enonce = senc(nonce, dhsk) in

  (* get OPERA quote *)
  out(la, ereport(sp, aem, (exp(g, y), enonce)));
  in(la, (quote: bitstring, cert: bitstring));

  (* send response *)
  out(ra, (quote, cert));

  (* rcv isv secret *)
  in(ra, emsg: bitstring).

let IsvClient(isvm: mrenc, iem: mrenc, ra: channel) =

  (* send challenge *)
  new x: exponent;
  new nonce: bitstring;
  out(ra, (exp(g, x), nonce));

  (* rcv response *)
  in(ra, (quote: bitstring, cert: bitstring));

  let (mrfie: mrenc, asvk: bitstring)
= checkepidsign(cert, epidvk(epidikIntel)) in

  (* security check: verify the identity of IssueE *)
  if mrfie = iem then
  let (mrf: mrenc, (gy: G, enonce: bitstring))
= checkepidsign(quote, bs2vk(asvk)) in

  (* security check: verify the identity of the attested enclave *)
  if mrf = isvm then
  let dhsk = G2sek(exp(gy, x)) in
  if sdec(enonce, dhsk) = nonce then

  (* send secret *)
  out(ra, senc(s, dhsk)).

process
  let epidvkIntel = epidvk(epidikIntel) in out(c, epidvkIntel);
  new epidmkQE1: epidmkey;
  new epidmkQE2: epidmkey;
  new nonceQE1: bitstring;
  new nonceQE2: bitstring;

  let mrQE = mr(sgxIntelQE) in out(c, mrQE);
  let mrIE = mr(sgxIssueE) in out(c, mrIE);
  let mrAE = mr(sgxAttestE) in out(c, mrAE);
  let mrISVE = mr(sgxIsvE) in out(c, mrISVE);

  new epidikAS: epidikey;
  (
    IntelQE(sgxIntelQE, epidmkQE1,
      epidct(epidpk(epidmkQE1, nonceQE1),
      epidikIntel, nonceQE1), ie_qe) |
    IssueE(sgxIssueE, epidikAS, mrAE, mrQE, ie_qe, ae_ie) |
    IntelQE(sgxIntelQE, epidmkQE2,
      epidct(epidpk(epidmkQE2, nonceQE2),
      epidikIntel, nonceQE2), ae_qe) |
    AttestE(sgxAttestE, mrQE, ae_qe, ae_ie, la) |
    IsvE(sgxIsvE, mrAE, la, ra) |
    IsvClient(mrISVE, mrIE, ra)
  )
```