

# Flexible remote attestation of pre-SNP SEV VMs using SGX enclaves

PEDRO ANTONINO <sup>1</sup>, ANTE DEREK <sup>2</sup>, and WOJCIECH ALEKSANDER WOŁOSZYN <sup>1,3,4</sup>

<sup>1</sup>The Blockhouse Technology Limited, Oxford, UK

<sup>2</sup>Faculty of Electrical Engineering and Computing, University of Zagreb, Zagreb, Croatia

<sup>3</sup>Mathematical Institute, University of Oxford, Oxford, UK

<sup>4</sup>St Hilda's College, Oxford, UK

Corresponding author: Pedro Antonino (e-mail: pedro@tbl.com).

This research has been partly supported by the European Regional Development Fund under the grant KK.01.1.1.01.0009 (DATACROSS).

**ABSTRACT** We propose a protocol that explores a synergy between two TEE implementations: it brings SGX-like remote attestation to SEV VMs. We use the notion of a *trusted guest owner*, implemented as an SGX enclave, to deploy, attest, and provision an SEV VM. This machine can, in turn, rely on the trusted owner to generate SGX-like attestation proofs on its behalf. Our protocol combines the application portability of SEV with the flexible remote attestation of SGX. We formalise our protocol and prove that it achieves the intended guarantees using the Tamarin prover. Moreover, we develop an implementation for our trusted guest owner together with example SEV machines, and put those together to demonstrate how our protocol can be used in practice; we use this implementation to evaluate our protocol in the context of creating *accountable machine-learning models*. We also discuss how our protocol can be extended to provide a simple remote attestation mechanism for a heterogeneous infrastructure of trusted components.

**INDEX TERMS** remote attestation, trusted execution environments, SGX, SEV, security

## I. INTRODUCTION

Primitives to implement a Trusted Execution Environment (TEE) [34] are becoming a common feature of modern processors. Such an environment typically allows a program to execute confidentially whereby not even the operator can tell what instructions and data are being used, we refer generically to such a protected execution as an *isolated computation*. Intel's Software Guard Extensions (SGX) [17], [24], AMD's Secure Encrypted Virtualization (SEV) [5], [28], and ARM's TrustZone [47] are examples of TEE implementations available. They are designed to address different application scenarios, but they all share similar core capabilities.

Intel's SGX and AMD's SEV provide competing TEE architectures that isolate computations at different levels of granularity. While SGX was designed to isolate (part of) an operating system process (an *enclave* in SGX terminology), SEV isolates an entire virtual machine (VM). Given these design choices, SGX does not offer the same level of application portability that SEV does. An application has to be redesigned to be made SGX-aware, whereas SEV allows it to be seamlessly executed within a confidential machine. This portability comes at the price of a having a typically larger *trusted computing base*. While SGX allows developers to finely tune which functions and data are part of the enclave,

SEV VM would usually contain an entire operating system (OS) together with the relevant applications to be executed. The larger the trusted computing base, the more prone to bugs and vulnerabilities it is.

*Remote attestation* is the process that establishes trust on an isolated computation. It consists of a protocol that produces evidence that a given computation has been properly isolated and, typically, provides a way to establish a secure channel with the isolated computation. While SGX provides a very flexible mechanism to attest enclaves, SEV (pre-SNP<sup>1</sup>) relies on a very restrictive scheme for that. While SGX's attestation is *undirected*, namely, any third-party can establish trust on a given enclave, SEV proposes a mechanism by which only a designated party, called the *guest owner*, can meaningfully attest (and provision) its SEV VM. To illustrate such a *directed* attestation procedure, we present in Figure 1 a scenario where a relying party is trying to interact with a SEV VM machine that has been deployed using the traditional guest owner approach. To engage with this VM, the relying party has to trust the guest owner in order to attest the SEV VM; the green outline denotes this trust boundary. So, the relying party

<sup>1</sup>We call *SEV pre-SNP* the SEV implementations predating SEV SNP (Secure Nested Paging) [54], i.e., the original SEV implementation [28] and SEV-ES (Encrypted State) [27].

has to trust the VM measurements and provisioned secrets that have been generated and transmitted by the guest owner, so that it can authenticate the SEV VM it wants to interact with. Of course, given that the guest owner knows this secret it can impersonate the VM. Given this possibility, this sort of directed attestation seems unsuited for scenarios where a (non-guest-owner) relying party is trying to attest a SEV VM to perform some confidential computing. For this type of attestation, the only reasonable usage scenario is arguably to have the guest owner as the single user of the SEV VM; a very restrictive scenario.

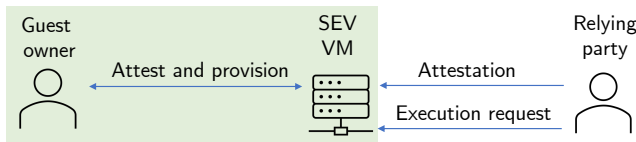


FIGURE 1. SEV attestation scenario with traditional guest owner.

We propose, formalise, verify, implement and evaluate a new protocol that provides *SGX-like remote attestation to an SEV VM*. Broadly speaking, it relies on a special enclave that we design, the *trusted guest owner*, that is responsible for deploying, attesting, and provisioning the SEV VM it owns. Moreover, while operating, this VM can request the generation of attestation reports, on its behalf, to the trusted guest owner — in the similar way to how an enclave can create an attestation report in the SGX architecture. Our innovative combination of TEE implementations brings together the best of both worlds, namely, the application portability of SEV and the flexible (undirected) attestation of SGX. However, our protocol requires two separate platforms: an SGX-capable machine to run the trusted guest owner and an SEV-capable one for the confidential VM. Therefore, the flexibility comes at a price of a larger trusted computing base. We illustrate our protocol and contrast with the traditional SEV VM deployment with a scenario where a relying party is trying to interact with a SEV VM machine that has been deployed using our trusted guest owner; see Figure 2. To engage with this VM, the relying party can attest the trusted guest owner and confirm that its code is correctly launching, attesting and provisioning the expected SEV VM instance. In this alternative scenario, the relying party has to trust another TEE implementation — the trusted owner implemented as an SGX enclave. However, instead of blindly trusting a generic entity, the guest owner, that could impersonate the desired SEV VM, it can attest the *trusted* guest owner and be sure, assuming the SGX implementation is trusted, that this enclave will correctly execute its code and behave in a trustworthy way.

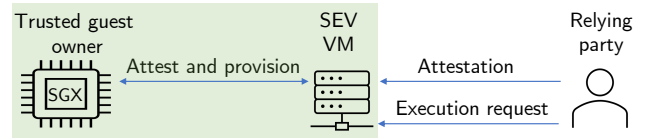


FIGURE 2. SEV attestation scenario with our trusted guest owner.

A composition of systems does not necessarily yield a scheme that inherit the security properties of the components — for instance, composing secure protocols does not automatically yield a secure scheme. Finding a protocol design that ensures the desired attestation properties was therefore challenging, and that is also why we formally analyse our protocol. We use the Tamarin prover [35] to model our protocol and to verify that it indeed achieves the desired goal of authenticity and integrity of attestation proofs. Additionally, we verify security properties of SGX and SEV attestation as used in our protocol — the authenticity of the SGX attestation proofs and secrecy of SEV provisioned secrets, respectively. All results hold in a general setting with unbounded number of participants and sessions, assuming a Dolev-Yao attacker [19] and a fine-grained threat model that, for example, allows the attacker to run enclaves of its choice alongside the trusted guest owner and compromise some TEE platforms.

In this paper, we assume that the trusted guest owner and guest VM cannot be compromised; we are interested in proving the security of our protocol under this assumption. Note that even with secure TEE implementations, cryptographic protocols could misuse them and introduce protocol-level attacks — we show that this is not the case for our protocol. Furthermore, we believe that making TEE implementations secure and more robust is a problem orthogonal to the one we solve here. We are interested in showing that the protocol is secure under this assumption. There are a number of papers that look into side-channel attacks and how to mitigate and prevent them [9], [12], [13], [31], [32], [40]–[42], [48], [56]–[60], [63], for instance, and at how TEEs can be deployed in a cluster, for instance, to improve robustness [33].

To demonstrate the protocol, we implement the protocol participants — namely the trusted guest owner, the SEV guest VM attestation library and several sample SEV guest VMs. Furthermore, we evaluate our protocol by harnessing it to implement a notion of accountability for machine learning models — i.e. creating a cryptographic report that ties a model to the technique and data used to generate it. Our evaluation demonstrates that our protocol incurs a negligible overhead while delivering on its security promises.

Some recent TEE implementations such as SEV SNP (Secure Nested Paging) [54] and Intel’s TDX (Trust Domain eXtensions) [25] were designed to provide a combination of remote attestation flexibility and application portability that is similar to what our protocol achieves with the proposed pairing of SGX and SEV. However, these technologies are still not widely available and the underlying attestation mechanisms and primitives have not yet been fully scrutinised

by the research community. Since Q1 2023, there a limited number of Intel CPU models supporting TDX available on the market [16]. However, at the time of writing (May 2023) the general availability of TDX remains planned for future Intel Xeon family releases and no major cloud provider offers TDX capable CPUs. Hardware support for SEV SNP was launched two years ago (Q2 2021), but software support is somewhat lagging and SNP patches were being merged to Linux kernel in Q3 2022. While some cloud providers do offer SEV SNP enabled hardware, we found that no major provider exposes the exible attestation interface to the end user. Microsoft Azure, for example, only allows their pre-approved VMs to be launched as SEV SNP guests [37], [38], and exposes attestation only through Azure-issued JWT (JSON Web Token) tokens [36]. Our protocol, on the other hand, is based upon TEE implementations that are reasonably mature: SGX and SEV pre-SNP have been launched in 2015 and 2017. So, they have been available, used, and investigated for quite a few years, whereas SEV SNP and TDX have not. Even when these new technologies mature, our protocol will still be relevant for platforms, legacy or not, that do not support SEV SNP or TDX but support SEV pre-SNP.

Our protocol sheds light in a new line of research, that is, finding synergies between TEE implementations. In our case, we create a protocol that brings together a pairing of an SGX enclave and an SEV VM in a way that it offers better features than both elements individually. Moreover, it can be extended to handle a related problem, namely, how to attest a homogeneous infrastructure of trusted components. Our protocol can be seen as a degenerate case of this problem where the trusted guest owner deploys a simple trusted infrastructure consists of a single SEV VM. However, our ideas could be carried over to the context of a general trusted deployment that could deploy, attest and provision a complex composition of trusted components. We elaborate on this idea in Section III-C but we leave a full investigation and implementation of it as future work.

We sum up our contributions in the following:

We propose a protocol that brings SGX-like remote attestation to SEV VMs, creating a synergy that combines the application portability of SEV with the exible remote attestation of SGX.

We formalise our protocol and verify it achieves the desired guarantees/goals using the Tamarin prover.

We created implementations for our trusted owner and several protocol-compatible SEV VMs.

We carried out an evaluation that demonstrates how our protocol can be used to implement a notion of accountability for machine learning models. It also shows that it delivers its guarantees with negligible overhead.

The proposal of our protocol sheds light in a new line of research consisting of exploring synergies between different TEE implementations.

We discuss how our protocol can be extended to provide a simple way to remotely attest an infrastructure involving heterogeneous trusted components.

Outline. In Section II, we introduce relevant background. Section III introduces our protocol, together with minimalist and abstract versions of SEV and SGX attestation protocols, and presents the formalisation of our protocol and discuss the properties that we were able to verify using Tamarin, and demonstrate an application of our protocol together with an evaluation of how it fares in practice. Section IV discusses some of the works related to ours, whereas in Section V, we present our concluding remarks.

In this section, we introduce the background elements that are necessary for understanding the rest of our paper.

### A. SGX

Intel's SGX (Software Guard eXtensions) [17] allows an untrusted host process to create a protected virtual-memory range where integrity-protected and confidential code and data are hosted; this protected area is called an enclave. SGX extends Intel's traditional instruction set with privileged instructions to create, initialise, and dispose of this protected memory range and also to non-privileged instructions to execute enclave code [22]. A number of hardware and software components take part in enforcing the integrity and confidentiality of an enclave's execution and in attesting these properties. These elements together with the enclave code itself form the trusted computing base (TCB) of that enclave, which is depicted in Figure 3; green elements are trusted, the others are not. At the lowest level, we have the trusted SGX hardware, comprising CPU package and Memory Encryption Engine [20], and low-level code; they ensure the integrity, confidentiality and freshness of the enclave's protected memory area. Privileged code is trusted privileged instructions cannot be executed in enclave mode. Hence, an enclave has to delegate to untrusted code, in the form of the OS/hypervisor, the execution of system calls, for instance. An enclave does not automatically trust other enclaves; they are isolated from one another. There are, however, some special architectural enclaves which are trusted. They play a fundamental part in the attestation process, namely, in the protocol by which an enclave provides to a counterpart evidence that it is indeed a valid isolated computation executing on an authentic platform. This process attests, in fact, the entire TCB: it provides the digest (or measurement) of the code loaded into the enclave, and information about the version of the architectural enclaves used and the SGX hardware and low-level code. We elaborate on this process/protocol later. Applications in user-space are also not trusted by the enclave. We refer generically to the untrusted components around an enclave in an SGX platform as the SGX host

<sup>2</sup>We make the protocol implementation, the sample systems used for evaluation, as well as the formal model and proofs publicly available [2] under a permissive open source license.

SGX enclave trusted computing base in green.

an SGX computation, making it potentially more vulnerable to bugs and design flaws. However, the fact that an entire OS (and its privileged instructions) is part of the trusted world makes this architecture more attractive in terms of application portability. An application that was not designed specifically to target an SEV VM can seamlessly (i.e. without modification) execute inside one. The same cannot be said of SGX: typically, applications have to be significantly redesigned to fit their enclave model.

### C. TAMARIN PROVER

Tamarin prover [35] is a tool for modeling security protocols and reasoning about their properties in the symbolic model of cryptography. Protocols are specified using multiset rewriting rules, while the security properties are specified either as guarded first-order logic formulas over execution traces or as observational equivalences. Proofs can be carried manually using the interactive mode or in an automated fashion where the procedure can be further tuned by supplying a proof oracle that prioritises available proof steps.

Tamarin prover has been successfully used to analyse, discover vulnerabilities and provide machine-verifiable proofs of various security properties for real-world protocols such as TLS v1.3 [18], smartcard payment protocols [8], 5G authentication protocols [7], and many others. In the area of trusted hardware, the tool has been used for analysis of a Direct Anonymous Attestation protocol based on the Trusted Platform Module (TPM) technology [61], [62].

In this section, we introduce a protocol that combines SGX and SEV attestation protocols in a way that it enables the hardware and firmware, especially in the form of the Secure Processor - also known as Platform Security Processor, or PSP. Note that other SEV VMs are not trusted; they are isolated from one another. Other non-SEV VMs are untrusted as well. Similarly to what we do for enclaves, we refer generically, to the untrusted elements surrounding an SEV VM in a platform as the SEV host. This section with a discussion on some interesting extensions to our protocol and its limitations. In this paper, we assume that side-channels attacks are possible and that the attacker can corrupt and extract secrets from arbitrary SGX/SEV platforms, enclaves, and VMs, except for specific platforms, enclaves, and VMs used in the protocol sessions. We claim (and formally verify) that the proposed protocol provides a level of robustness to those attacks.

### A. REMOTE ATTESTATION FOR SGX ENCLAVES

Intel has proposed two mechanisms to perform the remote attestation of an enclave: Enhanced Privacy ID (EPID) [26], and Data Center Attestation Primitives (DCAP) [53]. We present a minimalist protocol for remote attestation inspired by DCAP but that abstracts away its complexity and details, focusing on its broad trust guarantees and functionality. It

SEV VM trusted computing base in green.

### B. SEV

AMD's SEV (Secure Encrypted Virtualization) [28], [54] proposes an architecture to support confidential virtual machines (VMs), which we refer to as SEV (guest) VMs. This TEE implementation was designed so that even if the host (hypervisor included) is untrusted, it is unable to peek into the execution of an SEV guest VM. As for SGX, the AMD's typical instructions set was extended to incorporate directives to manage SEV VMs [5]. The TCB of an SEV guest machine is illustrated in Figure 4. It consists of its own code plus SEV and SEV attestation protocols in a way that it enables the hardware and firmware, especially in the form of the Secure Processor - also known as Platform Security Processor, or PSP. Note that other SEV VMs are not trusted; they are isolated from one another. Other non-SEV VMs are untrusted as well. Similarly to what we do for enclaves, we refer generically, to the untrusted elements surrounding an SEV VM in a platform as the SEV host.

The SEV architecture has evolved from (original) SEV [28], to SEV-ES (Encrypted State) [27], and recently to SEV-SNP (Secure Nested Paging) [54]. SEV-ES brings extra confidentiality guarantees when a switch from an trusted to an untrusted execution takes place, namely, the contents of the registers storing the state of the confidential VM enclaves, and VMs used in the protocol sessions. We claim (and formally verify) that the proposed protocol provides a level of robustness to those attacks. It also brings a form of remote attestation that is more flexible than SEV and SEV-ES. We discuss an abstract version of the pre-SNP attestation protocol later.

The difference in the level of granularity for the isolated computations between SGX and SEV has relevant practical consequences. In SGX, a simple (part of a) process is isolated as opposed to an entire VM in SEV. Therefore, the TCB for any SEV isolated computation tends to be much larger than that of

should be straightforward to adapt our protocol to work with the fully-edged DCAP or EPID.

Our SGX attestation protocol involves four parties: the attested enclave  $E$ , the quoting enclave of the attested platform QE, Intel's Root of Trust service Intel RoT, and a relying party RP. Broadly speaking QE is a trusted architectural enclave that runs in the same platform  $P$  and is certified by Intel RoT, and it creates proofs to attest to RP. Note that our italicised notation here denotes the name of the participants in our protocol. So QE is not an abbreviation for quoting enclave in general but a identifier denoting the attested quoting enclave that participates in our protocol. We adopt this notation consistently for the participants involved in the protocols that we describe in this paper.

#### Protocol goal.

The protocol produces an attestation proof  $\Pi$  consisting of a quote in SGX terminology and an SGX platform certificate. It authenticates  $E$ 's TCB. The platform certificate also contains the Platform Provisioning ID (PPID) uniquely identifying the platform instance. The quote also contains a piece of data  $D$  that is provided by  $E$ . Any relying party can, then, cryptographically validate this proof and be convinced that this quote was generated on a platform identified by PPID using the given TCB and the  $D$  provided when the protocol was executed.

#### Threat model and trust assumptions.

We assume that the platform in which  $E$  is deployed has not been compromised but the attacker controls the SGX host, i.e. untrusted platform elements, and the network. So, it can arbitrarily influence communications and computations executed by these elements, and create other enclave instances. The attacker has access to compromised SGX platforms to which it can deploy enclaves. A compromised platform would allow the attacker to have access to the cryptographic keys managed by the quoting enclave and, hence, to construct arbitrary quotes that validate as correct quotes from that particular platform. The enclave itself is known, and the attacker can deploy it at will on any platform of its choice. However, the entire attested TCB, including  $E$  and QE, and Intel RoT are trusted. Hence, the attacker can only interact with them in the ways prescribed by their implementation. We assume that the attacker cannot perform fork attacks or rollback attacks on our enclave. This is a reasonable assumption since the enclave state will be entirely in-memory with no persisted data.

#### Cryptographic schemes.

Our protocol relies on the following cryptographic schemes:

Intel RoT uses an asymmetric signature scheme with key-pair generation function  $agen_R()$ , signing function  $asign_R(m; k)$ , and verification function  $averi_R(m; s; k_{pb})$ ,  $m$  is a message,  $s$  is a signature,  $k$  is a private key, and  $k_{pb}$  a public one. We use the same notation with a similar meaning when denoting other asymmetric signature schemes;

Intel RoTs long term key pair  $(IntelLtk_{pb}; IntelLtk)$ , public and private elements, respectively, is generated using  $agen_R()$  and used by it to issue SGX platform certificates;

QE uses the asymmetric signature scheme with functions  $agen_{QE}()$ ,  $asign_{QE}(m; k)$  and  $averi_{QE}(m; s; k_{pb})$ ; QE key pair  $(Qek_{pb}; Qek)$ , public and private elements, respectively, is generated using  $agen_{QE}()$  and used by the quoting enclave to issue attestation quotes.

We assume throughout the paper that all cryptographic payloads are tagged with labels describing the payload structure and intent of the message. For example, the payload in the certificate  $C_{QE}$  below is  $h'sgx\_platform\_certificate'; Qek_{pb}; ppidi$ . However, we leave the type tags out of the protocol description to simplify notation. Of course, we include the tags in the formal model and in the protocol implementation.

#### Protocol.

We split the attestation protocol into the setup, quote generation, and quote verification phases. The protocol is depicted in Figure 5.

The platform setup phase establishes and ensures the existence of a chain of trust that extends from the Intel's root of trust to the attestation proof. During the setup phase, the SGX platform interacts with Intel RoT. Using a secret shared in the manufacturing process, the platform can attest itself and the quoting enclave to the root of trust. Once this attestation is successfully carried out, the root of trust certifies the quoting enclave, that is, the root of trust produces a certificate  $C_{QE} = (Qek_{pb}; ppidi; asign_R(hQek_{pb}; ppidi; IntelLtk))$ . We assume that this phase happens successfully as the SGX platform is being set up so that  $C_{QE}$  is made publicly available. The quote generation phase, if successfully executed, produces a quote which is a tuple  $(msr; plat; data; sig)$  where  $msr$  is the measurement of the enclave being attested,  $plat$  is a data structure containing information about the SGX platform,  $data$  is a vector of "free" data generated by the enclave being attested, and  $sig = asign_{QE}(msr; plat; data; Qek)$  is the signature of the quoting enclave on these other quote elements the notation  $\{e_1; \dots; e_n\}$  provides the ordered concatenations of elements through  $toe_n$ . It is a statement that an enclave with measurement  $msr$  was running in a authentic SGX platform with characteristics given by  $plat$  and it provided  $data$  when taking part in the attestation protocol. The quote is only produced if  $E$  provides a local attestation report. When the enclave with measurement  $msr$  invokes the SGX instruction EREPORT passing  $data$  as an argument, it creates such a report with which the quoting enclave can verify the integrity of  $data$  and its provenance from  $enclave_{msr}$ .

Given the expected enclave measurement  $msr_{exp}$ , the expected data  $data_{exp}$ , a quote  $Q = (msr; plat; data; sig_{QE})$ , a certificate  $C_{QE} = (Qek_{pb}; ppidi; sig_R)$ , RP can execute quote verification process consisting of: (i) verifying the signature  $sig_R$  using  $averi_R(hQek_{pb}; ppidi; sig_R; IntelLtk_{pb})$ , (ii) ver-

DCAP protocol sequence diagram.

ifying  $sig_{QE}$  using  $verify_{QE}(hmsr; plat; data; sig_{QE}; Qek_{pb})$ ,  
 (iii) checking that  $hmsr$  and  $data$  corresponds to the expected  
 enclave measurements  $emsr_{exp}$  and  $data_{exp}$ . Optionally, in some  
 usage scenarios the relying party may also verify that the  
 and  $plat$  match expected values or satisfy some other crite-  
 ria. We use the function  $VerifyQuote(emsr_{exp}; data_{exp}; Q; C)$  to  
 capture the validations (i-iii) of the quote verification phase.

Our simplified protocol abstracts away the details and complex-  
 ity of DCAP while focusing on its essential behaviour. The fully-  
 edged DCAP protocol relies on another architectural enclave (the  
 Provisioning Certification Enclave) in the setup phase, and the  
 certification of the quoting enclave is given by a certificate  
 chain, whereas our protocol abstract that chain by a single certi-  
 cate. We do not detail what is in the plat structure as the goal  
 of this paper is not to discuss the practical intricacies of an  
 SGX platform.

Despite its simplicity, our protocol still provides achieves the  
 protocol's goal given the threat model and trust assumptions  
 defined, as demonstrated by our formal analysis. Note that a  
 quote is not directed at a specific verifier: any relying party  
 possessing Intel's root of trust key can verify the quote and  
 SGX platform certificate.

### B. REMOTE ATTESTATION FOR SEV MACHINES

Compared to SGX, SEV's attestation primitives are not as  
 flexible giving rise to an attestation protocol that is arguably  
 more restrictive and intricate. The attestation protocol takes  
 place as the SEV guest VM is being created, and includes a  
 provisioning step. In this paper, we are concerned with the  
 attestation protocol and infrastructure of SEV pre-SNP. As  
 for SGX, we propose an abstracted protocol that focus on the  
 relevant functionality implemented by the fully- edged SEV  
 protocol.

The protocol involves the following parties: the AMD's  
 secure processor of the attested platform (SP), AMD's root  
 of trust service (AMD RoT), and the guest VM owner (GO),  
 and its attested guest VM (SVM). AMD RoT is in charge of

certifying the platform's SP, while GO interacts with SP to  
 attest, provision, and create SVM.

Protocol goal.

The protocol produces GO-directed attestation proof, mea-  
 surement in SEV terminology<sup>3</sup> and a SEV platform certi-  
 cate and provisions SVM with GO-generated secret. Once  
 the protocol is completed, GO is convinced of the authenticity  
 of SVMs TCB, and that S could only have been provisioned  
 to SVM.

Threat model and trust assumptions.

The same threat model and trust assumptions used for the  
 SGX protocol are used in the analysis of the SEV protocol,  
 with the exception that, here, we consider the SEV TCB  
 and platform and AMD RoT service as trusted elements as  
 opposed to the SGX and Intel counterparts, of course. Here,  
 a compromised platform would allow the attacker to obtain  
 any information that SP knows, including the cryptographic  
 keys it manages. We do not allow SEV VM migration. We  
 do not consider memory-remapping, rollback, or fork attacks;  
 we assume integrity-checking mechanisms can be put in place  
 to prevent those. Moreover, our trust in the attested SEV TCB  
 is intended to prevent all architectural attacks including  
 the ones affecting attestation primitives [11], [64]. This as-  
 sumption allows us to analyse the security properties of  
 the protocol itself, as opposed to weaknesses linked to the  
 bad design/implementation of the underlying primitives.

Cryptographic schemes.

The protocol involves the following cryptographic schemes:

<sup>3</sup>An SEV measurement is different from an SGX measurement. The latter  
 refers to the digest of the enclave's code, whereas the former is a digest  
 calculated from the VM firmware code but it also includes some platform  
 and launch-policy information as well as a nonce binding the measurement to  
 a particular VM launch session.

AMD RoT uses an asymmetric signature scheme defined by functions  $agen_{AR}()$ ,  $asign_{AR}(m; k)$ , and  $averi_{AR}(m; s; k_{pb})$ ;

AMD RoT key pair  $(AmdLtk_{pb}; AmdLtk)$ , public and private elements, respectively, is generated using  $agen_{AR}$  and used by the root of trust to issue SEV platform certificates;

SP and GO rely on the asymmetric secret-negotiation scheme with key-generation function  $sgen()$  and secret computation function  $snc(K_{pb}; K)$ , where  $K_{pb}$  and  $K$  are public and private key elements of the scheme. Diffie-Hellman key-sharing scheme is an instantiation of such a scheme.

GO generates the key pair  $(GoSp_{pb}; GoSr)$  using  $sgen()$ .

SP generates a key pair  $(PspSp_{pb}; PspSr)$  using  $sgen()$ . SP and GO rely on a key-derivation function  $oder(Sd)$ , where  $Sd$  is a derivation seed.

SP and GO rely on the symmetric encryption scheme defined by key-generation function  $sgen_e()$ , encryption functions  $sen(m; k)$ , and decryption functions  $sdec(m; k)$ , where  $m$  is a message and  $k$  is a scheme's key. This scheme is used for encrypting key-wrapping interactions and transported messages between them.

SP and GO rely on the message authentication code (MAC) scheme defined by key-generation function  $sgen_l()$ , signing function  $ssigr(m; k)$ , and verification function  $sver(c; k)$ , where  $m$  is a message,  $c$  is an authentication code, and  $k$  is a scheme's key. This scheme is used for integrity-protecting key-wrapping interactions and transported messages between them.

Here and in our protocol description we are relying on a single symmetric encryption scheme and a single MAC one for the sake of simplicity. However, one could use multiple schemes, one for each different application, without affecting the protocols' guarantees.

Protocol.

We divide the protocol execution into three phases: SEV platform setup, secure-channel establishment, VM validation & provisioning, all of which we detail next. The protocol is depicted in Figure 6.

The platform setup phase for the SEV protocol is very similar to the one that we presented for SGX. It involves only SP and the AMD's root of trust service. It establishes a similar chain of trust, providing similar guarantees, and it also relies on a fused pre-shared secret for platform authentication. So, when successfully executed, this phase produces the SEV platform certificate  $C_{sp} = (PspSp_{pb}; asign_{AR}(PspSp_{pb}; AmdLtk))$ . We assume that this phase is successfully completed at the time the platform is set up and that this certificate is made publicly available. Notice that, unlike SGX platform certificates, SEV certificates (by AMD's design) do not contain a platform identifier. In our protocol, we will use SP's public key  $PspSp_{pb}$  to uniquely identify a particular SEV platform.

During the secure-channel establishment phase, SP and GO interact to set up a communication channel. GO obtains the PSP certificate  $C_{PSP} = (PspSp_{pb}; sig)$  for the platform and verifies it using  $averi_{AR}(PspSp_{pb}; sig; AmdLtk_{pb})$ . At this point, GO generates the (shared) secret  $s = snc(PspSp_{pb}; GoSr)$ , which is used in turn to generate keys  $Kek$  and  $Kik$  via the key derivation functions  $oder$ . These two key-wrapping keys (as per SEV terminology) are then used to transmit the pair of freshly generated transport keys  $Tek = sgen_e()$  and  $Tik = sgen_l()$  generated by GO. It creates the deploy package message  $(GoSp_{pb}, blob_D, mac_D, vm_D)$  to be transmitted to SP where  $vm_D$  is SVMs firmware code,  $blob_D = sen_e(hTek; Tiki; Kek)$  is the encrypted-keys blob, and  $mac_D = ssigr(blob_D; Kik)$  its authentication code. Note that SVMs code is transmitted in the clear without any integrity protection.

Upon receiving the message  $(GoSp_{pb}; blob; mac; vm_D)$ , SP can derive the same secret  $s$  using  $snc(GoSp_{pb}; PspSr)$ , and use it to derive keys  $Kek$  and  $Kik$  by the same key derivation process as GO. These keys can be, in turn, used to decrypt the received blob and recover the transport keys, i.e.  $hTek; Tiki = sdec(blob; Kek)$ , and authenticate and integrity check them with  $sver(blob; mac; Kik)$ . Therefore, at the end of this phase SP and GO have set up a secure communication channel by sharing  $Tik$  and  $Tek$ .

The VM attestation & provisioning phase proceeds as follows. SP prepares SVM with code  $vm_C$  for launch and calculates the corresponding code digest  $dig_C$ . Then, it creates the measurement  $msr = (plat_{sev}; launch_{sev}; dig; nonce)$  where  $nonce$  is a freshly generated random value. Structures  $plat_{sev}$  and  $launch_{sev}$  abstract information related to SVMs. SP constructs the measurement package message  $(msr, mac_{T1})$ , where  $mac_{T1} = ssigr(msr; Tik)$ , which is transmitted to GO.

Upon receiving message  $(msr; mac)$ , GO validates the measurement by checking  $sver(msr; sig; Tik)$  and that the measurement elements are as expected; it includes checking  $diges(msr) = dig_{exp}$ , where  $diges(m)$  gives the code digest element of the measurement  $m$ , and  $dig_{exp}$  is the digest independently computed by GO using  $vm_C$ .

If this measurement validation succeeds, GO proceeds to provision SVM. It generates secret  $s$ , and creates the encrypted blob  $blob_P = sen_e(s; Tek)$ , and the corresponding authentication code  $mac_P = ssigr(blob_P; msr; Tik)$ . Note that  $mac_P$  takes into account the SVMs measurement  $msr$ . This secret package message  $(blob_P; mac_P)$  is then sent to SP. Upon receiving message  $(blob; mac)$ , SP recovers the secret  $s$  by decrypting the encrypted blob  $s = sdec(blob; Tek)$ , and it checks  $sver(blob; msr; mac; Tik)$  to verify the secret blob's authenticity and integrity, and that it is provisioning the machine with the correct  $msr$ . If this verification does not succeed, this provisioning step is aborted. Otherwise, SP places the secret  $s$  in an encrypted SVMs memory. Once this step is completed, SVM is allowed to start its execution.

Our protocol focuses on the essential functionality required

SEV remote attestation protocol sequence diagram.

to prove that it achieves the desired goal given the threat model and trust assumptions defined. So, we simplify and abstract away elements as long as the intended guarantee can be delivered. For instance, the fully-edged SEV protocol relies on a certificate chain which we "atten" to a single platform certificate. Moreover, we abstract platform and launch details by relying on opaque structures. Our model could rely on predicates over these opaque structures to identify "desirable" platform and launch settings. There are many implementation details related to identifying memory ranges in the messages exchanged with

Unlike the SGX protocol, the SEV attestation (and provisioning) is directed at the guest owner, and it does not contain any SEV-VM-provided data. Hence, a relying party cannot independently and convincingly establish an authenticated channel with an SEV VM – the guest owner alone has this capability.

### C. OUR PROTOCOL

Our protocol is built upon the notion of a trusted guest owner – an entity that deploys and provisions an SEV guest VM and is trusted to provide attestation reports on the deployed SEV VM's behalf.

Our protocol involves the parties in both SGX and SEV attestation protocols. However, the enclave in the SGX attestation coincides with the guest owner of the SEV attestation. So, the parties are: the trusted guest owner, the SEV guest VM, the quoting enclave, the AMD's secure processor, Intel's root of trust service, Intel RoT, and AMD's root of trust service, AMD RoT, and the relying party.

Protocol goal. The protocol produces an attestation proof consisting of a quote, and both SGX and SEV platform certificates. It authenticates both SVM and TO's TCBs. The SGX platform certificate contains the Platform Provisioning ID (PPID) uniquely identifying the SGX platform instance that was running, while the quote itself contains a digest of the SVM's public key uniquely identifies the SEV platform instance where the SP and SVM were running. Finally, the quote has the digest of a piece of data that is provided by SVM. Any relying party can, then, cryptographically validate this proof and be convinced that this quote was generated using the SGX platform identified by PPID and the SEV platform identified by PspSp<sub>b</sub> with the corresponding SGX and SEV TCBs, and that SVM provided D when the protocol was executed.

#### Threat model and trust assumptions.

We combine both models and assumptions of the two SGX and SEV attestation sub-protocols we use; the assumptions on TO are the same as the ones made about the attested enclave in the SGX attestation protocol. Moreover, SVM is trusted not to expose the provisioned secret, which is, in our protocol, a secret key shared between TO and SVM – we call such a machine compliant.

#### Cryptographic schemes.

We rely on the cryptographic schemes that are required by both SGX and SEV attestation protocols, which we do not restate here for the sake of brevity, plus the cryptographic hash function hash<sub>TO</sub> used by TO in emitting reports for SVM.



Flexible SEV attestation protocol sequence diagram outline.

Protocol

We split our protocol into phases: setup, secure channel establishment, VM attestation & provisioning, VM report generation, and verification by relying party. The protocol is depicted in Figure 7; we omit the setup phase from the diagram for conciseness.

The setup phase successfully carries out the setup phases of both SGX and SEV attestation protocols for the attested platforms, and it precedes the other phases of our protocol. As a result, it produces SE and QE certificates ( $PspSp_b; asig_{AR}(PspSp_b; AmdLtk)$ ) and ( $Qek_p_b; ppidi; asig_{R}(H(Qek_p_b; ppidi); IntelLtk)$ ), respectively.

TO plays a central role in the remaining phases of our protocol. Its code is presented in Algorithm 1. The global variables, stored in protected memory, define the enclave's state; they are listed after the keyword `vars`. The AMD root of trust public key is the only enclave constant, and is listed after keyword `consts`. The functions describe the trusted behaviour it can engage on. The input arguments for such a function is transmitted from unprotected to protected memory before its execution starts, output ones move in the opposite direction at the end of its execution, and its execution is confidential and integrity-protected. Note that, for a given instance of our trusted owner enclave, the implementation of our trusted functions ensures that `DeployVm` and `ProvisionVm` can only be meaningfully (without returning `None`) executed once and in this order. Function `GenerateReportForVm` can be meaningfully executed multiple times but only after the other two have meaningfully executed. We do not address the possibility

of replayed calls to function `GenerateReportForVm`. For the sort of usage we envision that possibility does not seem too problematic, but we could address that in future versions of our protocol.

The secure-channel establishment and the VM attestation & provision phases correspond to the homonyms of the SEV attestation protocol, presented in Section III-B, with TO playing the part of GO.

The function `DeployVm` implements the guest owner's behaviour in this phase. Given a PSP certificate and an SEV VM code digest as input, this function carries out all the necessary certificate verification, secret negotiation, key derivations and generations on its way to create and return TO's secret-negotiation public key  $gSp$ , the encrypted blob  $blob$  and authentication code  $mgf$  for the generated transport keys.

These keys are stored in enclave global variables `Tek` and `Tik`. This function also checks the expected code digest of the SEV VM being deployed, which is stored in the global enclave variables `VmDig`. Note that this function is only concerned with the digest of the VM code – the code itself can be stored and communicated by untrusted components. The elements returned by this function together with the VM code itself are combined to create the deployment package message. This message is relied on by SP, who carries out the rest of this phase as described in Section III-B.

The VM attestation & provision phase starts with SP constructing the measurement package message as per Section III-B. The function `ProvisionVM`, which implements the behaviour of the guest owner in this phase, takes as input

**Algorithm 1** TO's code. We use the schemes as defined in the text, and the well-known Option type. The enclave global variables and constants start with an uppercase letter whereas the local ones start with a lowercase one. Their types are not explicitly mentioned but they can be inferred from their usage. The constants hold the values of the corresponding public keys, and the global variables are initialised with None. As for the types of our functions, we use  $PUB_x$  to denote the public-key type of scheme identified by  $SIG_x$  is a signature type,  $CYP_{Pa}$  cyphertext type,  $DIG_{sev}$  the SEV code digest type,  $MSR_{sev}$  the SEV measurement type,  $REP_{sgx}$  the SGX local attestation report type, and  $DAT$  the VM report data type.

```

vars PspId, Tik, Tek, VmDig, Msr, Cik  None
consts AmdLtkPb

function DeployVm((PspSpb, sig) : PUB_Sn  SIG_AR, dig : DIG_sev) : Option(PUB_Sn  CYP_kek  SIG_kik)
  if VmDig = None ^ averAR(PspSpb, sig, AmdLtkPb) then
    PspId Some(PspSpb)
    VmDig Some(dig)
    (goSpb, goSn) snger()
    sd sns(PspSpb, goSn)
    kek, kik sdev(hsd, 'sev_kek'), sdev(hsd, 'sev_kik')
    Tek, Tik Some(sger()), Some(sger())
    blobD send(hTek, Tiki, kek)
    maD ssign(blobD, kik)
    return Some(goSpb, blobD, maD)
  end if
  return None
end function

function ProvisionVm(msr : MSR_sev, mac : SIG_Tik) : Option(CYP_TE  SIG_Tik)
  if VmDig ∈ None ^ Cik = None ^ sver(msr, mac, Tik) ^
  digest(msr) = VmDighen
    Msr Some(msr)
    Cik Some(sger())
    blobP send(Cik, Tek)
    maD ssign(hmsr, blobPi, Tik)
    Tek, Tik None, None
    return Some(blobP, maD)
  end if
  return None
end function

function GenerateReportForVm(vmdata : DAT, mac : SIG_Tik) : Option(REP_sgx)
  if Cik ∈ None ^ sverCI(vmdata, mac, Cik) then
    rpdata hashTO(hPspId, Msr, vmdata)
    return Some(EREPORT(rpdata))
  end if
  return None
end function

```

the measurement and authentication code in that message. The function carries out the verification of the input measurement, generates a MAC-scheme key stored in  $Cik$ , and produces the secret encrypted blob and authentication code. The blob and code are used to create the secret package message which is sent to RP, which carries out the secret package verification and provisioning, bringing this phase to an end, as per Section III-B. The sharing of the  $Cik$  key via this provisioning step establishes an authenticated (but not confidential) channel between TO and SVM.

The VM quote generation and verification phases involve the execution of the SGX attestation protocol, presented in Section III-A. These phases of the protocol take place after the initial three have successfully completed and SVM has started.

The VM quote generation starts with SVM creating a report request  $(vmdata, mac)$ , where  $vmdata$  is a piece of data generated by it, and  $mac = ssign_{CI}(vmdata, Cik)$ . This report request is then communicated to TO by invoking `GenerateReportForVm` with  $vmdata$  and  $mac$  as inputs. Upon successful verification of  $mac$ , this function creates an SGX report addressed to TO containing: TO's enclave measurement  $msr_{TO}$ , and a digest of the public key  $PspSpb$  identifying the attested SEV platform,  $vmdata$  and of SVM's measurement  $MSr$  represented as  $data$ . This report is transmitted to QE which generate the corresponding quote  $(rpdata, asig_{QE}(hmsr, data, Qek))$ .

RP verifies the VM quote using the function `VerifyQuote` in Section III-A. Let  $Q$  be the VM quote received,  $msr_{TO}$  the enclave measurement for TO,  $C_{QE}$  the quoting enclave certificate  $vmdata$  the VM piece of data,  $vmmsr$  the VM measurement, and  $pspid$  the attested SEV platform id. RP calculates the expected report data  $rpdata_{exp} = hash_{TO}(hpspid, vmmsr, vmdata)$ , and checks `VerifyQuote(msrTO, rpdataexp, Q, CQE)`. This validation convinces RP that the protocol's goal has been achieved, namely, that the  $vmdata$  was generated by an SEV VM with measurement  $vmmsr$ .

#### D. FORMAL SPECIFICATION AND VERIFICATION

To validate our proposal, we give a formal model of the executable attestation protocol, and use the Tamarin prover to provide machine-verifiable proofs that it has the desired security properties. Hence, the protocol meets its stated goals in a setting with an unbounded number of sessions assuming a Dolev-Yao attacker and a threat model described in Section III-C. We make the formal model as well as the proofs and the proof oracle needed to replicate the results publicly available at [2].

##### Protocol model

We model the protocol by specifying all participants using multiset rewriting rules as in [35]. Each rule is of the form  $id: [l] [a] [r]$ , where  $l$ ;  $a$ ;  $r$  are sets of facts. Facts in  $l$  are rule premises, facts in  $r$  are conclusions, and those in  $a$  are reaction facts of the rule. As an example, the rule corresponding to the `DeployVm` function of the trusted owner is given in Figure 8. First of all, the "let" binding only acts as syntactic sugar making the specification more readable. In the rule premises, TO ensures that it is running on an initialised SGX platform (by checking the existence of a persistent fact generated by another rule); it makes sure the PSP's certificate is already verified (by another rule); it receives the code of the guest VM SVM to deploy (abstracted as a public value); it creates and stores the necessary information in its session state. The request is created by generating the shared secret, deriving

```

rule TO_Enclave_Deploy_VM:
let
  go_sn_pk = 'g' ^ ~go_sn
  sd = psp_sn_pk ^ ~go_sn
  kek = h(<'sev_kek', sd>)
  kik = h(<'sev_kik', sd>)
  msg_content = <'transport_keys', ~tek, ~tik>
  blob = senc(msg_content, kek)
  mac = h(<msg_content, kik>)
  deploy_package = <go_sn_pk, blob, mac, $vm_dig>
in
[
  !SGX_Platform_Initialised(~ppid)
  , Platform_PK_Verified(psp_sn_pk)
  , In($vm_dig)
  , Fr(~go_sn)
  , Fr(~tek)
  , Fr(~tik)
]-[
  TO_Enclave_Deploy_VM()
  , TO_Enclave_Secrets(psp_sn_pk, sd, kek,
    kik, ~tek, ~tik)
]->[
  Out(deploy_package)
  , TO_Enclave_VM_Deployed(psp_sn_pk, ~ppid,
    ~tek, ~tik, $vm_dig)
]

```

One of the rewrite rules modeling the TO

keys and encrypting/MAC-ing appropriate data. Action facts are later used to specify security properties. In addition to protocol participants from Figure 3 (VM, SP, TO, QE, RP), we explicitly model Intel and AMD roots of trusts services.

The functional part of the formal model consists of 21 rules given in Table 1. The rules are almost in one-to-one correspondence with the description of protocol steps given in Section III-C. The exception are the attacker rules that we introduced to faithfully capture the threat model and allow the corruption of parts of the system.

Attacker model

The Dolev-Yao attacker rules are automatically embedded in the model by the Tamarin tool, but we need to add additional attacker actions to be faithful to the desired threat model. In particular, we add rules that disclose quoting enclaves and PSPs long term private keys to the attacker, corresponding to corruptions of arbitrary SGX and SEV platforms; these rules do not apply to non-compromised platforms. We also add rules to corrupt both roots of trust as a means to sanity check our model.

We list and discuss the attacker rules related to SEV here, the rules related to SGX are similar. The `Compromise_AMD_RoT` allows the adversary to compromise the AMD RoT and extract the `AmdLtk` private key. This rule was added purely for sanity checking purposes and, indeed, the main results and well as the lemmas related to SEV are falsified unless we assume the adversary did not use this rule.

```

rule Compromise_AMD_RoT:
[
  !AMD_RoT_Ltk(~amd_rot_ltk)
]

```

Rule name	Protocol party
Intel_RoT_Initialize	Intel RoT
Intel_RoT_Certify	Intel RoT
SGX_QE_Initialize	QE
SGX_QE_Generate_Quote	QE
AMD_RoT_Initialize	AMD RoT
AMD_RoT_Certify	AMD RoT
SEV_PSP_Initialize	SP
SEV_PSP_Initialize_Guest	SP
SEV_PSP_Launch_Guest	SP
TO_Enclave_Verify_Platform_Cert	TO
TO_Enclave_Deploy_VM	TO
TO_Enclave_Provision_VM	TO
TO_Enclave_Generate_Report_For_VM	TO
Guest_VM_Request_Report	SVM
RP_Verify_Quote	RP
Compromise_Intel_RoT	adversary
Compromise_SGX_QE	adversary
Adversary_Request_Quote	adversary
Compromise_AMD_RoT	adversary
Compromise_SEV_PSP	adversary
Adversary_Extract_SEV_Secret	adversary

All the rules in the formal model.

```

]-[
  Compromise_AMD_RoT()
]->[
  Out(~amd_rot_ltk)
]

```

The `Compromise_SEV_PSP` allows the adversary to compromise one specific SP and extract the `PspSn` private key of that platform. This rule models platform compromise (e.g., by side-channel attacks). We show that the main results hold even if the adversary can compromise arbitrary platforms, as long as the specific SP used in the protocol execution is not compromised.

```

rule Compromise_SEV_PSP:
[
  !PSP_Ltk(~cpu_id, ~psp_sn)
  , !PSP_Pk(~cpu_id, psp_pk)
]-[
  Compromise_SEV_PSP(psp_pk)
]->[
  Out(~psp_sn)
]

```

One of the modeling challenges was formalising the relationship between a measurement and the behaviour of the measured code. Using SGX as an example, we need to be able to combine the fact that the quoting enclave produced a quote with measurements `msrE` and data `dataE` with the fact that measurements `msrE` corresponds to specific enclave code `E` with certain behaviour when executed on trusted hardware (e.g., `E` only provides attestation reports in which `dataE` is in a specific format). To address this challenge in general, the framework has to support higher-order reasoning about the building blocks of protocol specification e.g., we need to use those building blocks both as programs that can be executed and as data that can be hashed or send over the network (perhaps to be executed on the other end). To the

best of our knowledge, no protocol verification framework currently allows reasoning about such constructions.

As our scope in this paper is limited to modeling and verifying the proposed protocol, we overcome this challenge by using a simple over-approximation of the attacker's capabilities. In the SGX setting, we assign a fixed measurement to the enclave TO is running. Furthermore, we allow the attacker to obtain valid quotes with arbitrary data for any measurement except for  $const_{TO}$ . Hence, we hardcode the relationship and the measurement of its enclave in our model, and assume enclaves corresponding to all other measurements are under the control of the attacker. We take a similar approach with SEV we hardcode the launch digest  $const_{SVM}$  of our guest VM and allow the attacker to extract secrets provisioned by the PSP from any SEV VM whose launch digest is different from  $const_{SVM}$ . We list the rule and give more details for SEV here.

The Adversary\_Extract\_SEV\_Secret allows the adversary to extract a provision secret from a VM running on arbitrary SP. This rule models the fact that adversary can launch and control arbitrary VMs on an arbitrary SP. The only thing we disallow (via the Neq restriction) is that the adversary extracts the secret from our specific SVM whose digest a constant  $const_{SVM}$  (a string burrito\_guest\_vm in the Tamarin model).

```
rule Adversary_Extract_SEV_Secret:
  [
    ISEV_PSP_Guest_Running(~cpu_id, psp_sn_pk,
      $vm_dig, ~guest_secret)
  ]-[
    Neq($vm_dig, 'burrito_guest_vm')
    , Adversary_Extract_SEV_Secret($vm_dig,
      ~guest_secret)
  ]->[
    Out(~guest_secret)
  ]
]
```

Security properties and proofs

The main security property we are interested in verifying is the authenticity and integrity of the resulting VM quotes. As helper lemmas, but also as results of their own merit, we verify the security properties of both SGX attestation and SEV secure guest deployment as used in our system. The most important verified properties are informally described next, and they are followed by the corresponding Tamarin lemmas.

**SGX quote authenticity** If RP verifies an SGX quote with the measurement  $const_{TO}$ , with a certificate identifying the ppid SGX platform, and quote data  $rd$ , then TO has executed GenerateReportForVm function on an SGX platform identified by ppid and rd data is equal to  $hash_{TO}(hPspId, Msr, vmdata)$  for some PspId, Msr and vmdata. The claim holds unless the attacker has compromised the Intel root of trust QE, the quoting enclave running on platform ppid.

**Secrecy of SEV guest secrets** TO executes ProvisionVm with the  $const_{SVM}$  parameter and a specific PspId value, then the secret being provisioned Cik is never known to the attacker. The claim holds unless the attacker has

compromised AMD RoT or SP, the specific PSP whose public key is PspId.  
**quote authenticity** If RP verifies an SGX quote with the measurement  $const_{TO}$ , with a certificate identifying the ppid SGX platform, and quote data that is equal to  $hash_{TO}(hPspId, Msr, vmdata)$  for some PspId and vmdata, and the digest in measurement  $const_{SVM}$  being  $const_{SVM}$ , then SEV VM has executed GenerateReportForVm while running on an SEV platform identified by PspId with the data in the request equal to vmdata. The claim holds unless one of the following is true: the attacker has compromised the Intel root of trust; the attacker has compromised QE, i.e., the specific QE corresponding to platform ppid; the attacker has compromised the AMD root of trust; the attacker has compromised SP, i.e., the specific PSP whose public key is PspId.

We present formal statements of the main results as well as the most important auxiliary lemmas in Tamarin notation. This notation is somewhat different compared to the informal statements above so we give clarifications when needed.

In the SGX quote authenticity lemma below, the informal statements "RP verifies an SGX quote" and "TO has executed GenerateReportForVm function" are modelled as Tamarin action facts (RP\_Verify\_Quote and TO\_Enclave\_Generate\_Report\_For\_VM respectively). These action facts hold at timestamps when the corresponding rules are executed. The variables  $ppid$  and  $rd$  correspond to ppid and rd data in the informal statement, while  $k$ ,  $d$  and  $v$  correspond to the report hash payload PspId, Msr and vmdata. Note that these are untyped in the lemma statement below and are, hence, quantified over all possible messages. Variables  $\#i$  and  $\#j$  are typed as timestamps. The constant SGX measurement of  $const_{TO}$  is simply a string burrito\_enclave\_sgx\_measurement.

```
lemma lm_sgx_quote_authenticity:
  "All ppid #i rd.
  RP_Verify_Quote(<'sgx_quote',
    'burrito_enclave_sgx_measurement', ppid,
    rd>) @ i
  ==>
  (
    (Ex v d k #j. rd = h(<'report_data', k, d,
      v>) &
      TO_Enclave_Generate_Report_For_VM(ppid,
        k, d, v) @ j )
    | (Ex #j. Compromise_Intel_RoT() @ j )
    | (Ex #j. Compromise_SGX_QE(ppid) @ j )
  )
"
```

In the Secrecy of SEV guest secrets lemma below, the constant launch digest of the SVM simply the string burrito\_guest\_vm. The action fact KU models the attacker knowledge, while  $s$  is the secret being provisioned to the SVM.

```
lemma lm_sev_guest_secret_secrecy:
  "All k s #i.
  TO_Enclave_Provision_VM(k, s,
    'burrito_guest_vm'
```

```

) @ i
==>
(
  (not Ex #j. KU(s) @ j)
  | (Ex #j. Compromise_AMD_RoT() @ j )
  | (Ex #j. Compromise_SEV_PSP(k) @ j )
)
"

```

In the VM quote authenticity lemma below, notation is similar same as in the previous two lemmas. Note that we do not include the platform and the policy metadata and launch\_sev to the SEV measurement as they do not play a security-related role on the level of abstraction used on our model. Instead, the SEV measurement is just a pair consisting of a nonce (modelled by variable  $n$ ) and the launch digest of the SVM

```

lemma Im_burrito_quote_integrity_strong:
  "All ppid d k m #i.
  RP_Verify_Quote(<'sgx_quote',
    'burrito_enclave_sgx_measurement', ppid,
    h(<'report_data', k, <m,
      'burrito_guest_vm'>, d)>)
  ) @ i
  ==>
  (
    (
      Ex ts #j.
        d = <'burrito_report', ts>
        & Guest_VM_Request_Report(k, ts) @ j
    )
    | (Ex #j. Compromise_Intel_RoT() @ j )
    | (Ex #j. Compromise_SGX_QE(ppid) @ j )
    | (Ex #j. Compromise_AMD_RoT() @ j )
    | (Ex #j. Compromise_SEV_PSP(k) @ j )
  )
"

```

We prove all results using the Tamarin prover's automated procedure with a custom proof oracle that was necessary to achieve proof termination. In addition to the main results stated above, we prove weaker variants of the claims above where we disallow the attacker from compromising any SGX or SEV platform. We also prove a number of helper lemmas and a number of sanity-checking lemmas in order to test the model itself. Most notably, we show that all the premises for main lemmas are indeed necessary by demonstrating the existence of an attack when any of the premises is removed.

### E. IMPLEMENTATION AND EVALUATION

To demonstrate how our protocol works in practice, we have created an implementation of our trusted guest owner, which can be applied to any compliant SEV VM we have published our code [2]. Our prototype relies on (i.e., instantiates) the abstract SEV and SGX protocols we present with the fully-edged versions of the SEV pre-SNP and SGX DCAP attestation protocols.

Our trusted owner enclave implementation uses the SGX SDK [23] to capture the behaviour described in Algorithm 1. The SGX SDK provides two main abstractions for the development of enclaves: trusted functions, which are called `calls` and untrusted ones, which are called `ecalls`. The enclave functions are described by `ecalls`, which can, in turn, rely

on `ecalls` to execute untrusted privileged code. Our functions `DeployVm`, `ProvisionVm`, and `GenerateReportForVm` are all implemented as `ecalls`, and they take into account the fully-edged SEV attestation operations and data formats. So, for instance, `DeployVm` checks the SEV certificate chain to authenticate the secret negotiation key as opposed to our single certificate abstraction. Our implementation uses the code of the `SEV-Tool`[1] as a library to carry out a number of operations related to the SEV attestation protocol this standalone tool has been created to help developers operate SEV VMs and platforms.

As a proof of concept and to evaluate how our protocol fares in practice, we applied it to the generation (i.e. training) of machine learning (ML) models. We use our protocol as a way to create a notion of model accountability in the sense that VM quotes can link a specific model with the training algorithm and data set that was used to create it. This sort of quote could be used, for instance, in the context of regulated ML, where one could be interested in analysing if a model was created in an unbiased/fair way.

The SEV VM that we create runs a single service, called `tf_service` at startup and shuts itself down after the service execution has finished. This service executes (via a Docker container) a TensorFlow [4] script that creates a ML model, export into `model.tar.gz`, and we capture the standard output of this script into `lestdout`. After creating these files, it produces a VM quote containing a hash of these two files as the VM quote report data. Thus, a relying party can verify that a given model was generated with a given data set and script. Note that the data could even be kept private up until the point it needs to be divulged to a regulator/auditor to ensure the appropriate generation of the associated model.

Name	Deploy	Provision	GenReport	VmLife	Over. (%)
advanced.py	0.118	0.088	0.139	198.268	0.174
bidirectional.py	0.121	0.0911	0.132	532.250	0.065
knowledge.py	0.122	0.103	0.132	1140.456	0.031
beginner.py	0.128	0.087	0.123	92.217	0.367
text.py	0.118	0.089	0.134	98.184	0.347
text_trans.py	0.129	0.081	0.130	1648.881	0.021
cnn.py	0.122	0.089	0.135	339.739	0.101
keras.py	0.121	0.094	0.134	117.956	0.295
preprocessing.py	0.120	0.101	0.136	98.965	0.360
classification.py	0.114	0.097	0.143	889.031	0.039
imbalanced.py	0.118	0.086	0.149	310.548	0.113
word2vec.py	0.116	0.093	0.131	117.545	0.289

Accountable ML evaluation results.

Our VM is based upon the Alpine Linux distribution. It relies on a modified SEV-ready kernel, an initial ramdisk that includes a root filesystem (containing `tf_service` and its dependencies), and a `xed` kernel command line these are the elements necessary to boot a Linux VM. The hashes of these three pieces of information are recorded in the initial VM firmware and are, hence, part of the VM measurement that can be verified by the relying party. The root filesystem is setup in main memory as opposed to disk.

<sup>4</sup><https://www.alpinelinux.org/>

We point out that our machine does not rely on the typical attestation scenario that is suggested by AMD, i.e. using a secure channel for the phases regarding attestation and establishment of a secure channel would depend on the specifics of each particular guest-owner-encrypted disk for which the key is provisioned TEE implementation supported. However, all the TEEs could use the SEV attestation protocol. Of course, once a VM has been setup using our protocol (and an initial root system information. The trusted deployer could both generate quotes in main memory like we do), it could include a routine to create an encrypted disk whose key would remain protected in main memory. So, our protocol and example VM could still accommodate disk encryption seamlessly.

Our evaluation takes into account 12 TensorFlow scripts. For each of them, we create corresponding VMs as explained on this paper's protocol and carry out deployment, provisioning, and report generation using our trusted owner, as per our protocol. The results of executing these VMs is presented in Table 2. We use a AMD SGX machine using an EPYC 7402P 24-Core processor to run the VMs and an Intel machine with a Intel(R) Xeon(R) E-2288G CPU @ 3.70GHz processor. In this evaluation, we measure the times taken to perform each of the trusted owner functions they include network latency as we use a remote trusted owner. The overhead is calculated as the  $(Deploy+Provision+GenReport)*100/VmLife$ ; it gives the percentage of time taken by the trusted owner operations with respect to the entire VM execution (VmLife).

As expected, the timings for executing trusted owner operations are fairly constant and independent of the VM lifetime (and execution complexity). Note that trusted owner operations are of fixed type and size so those are independent of the type of the VM being run. Moreover, the overhead imposed by our protocol is minimal: in all cases it came under 0.5% of the VM execution time. Therefore, unsurprisingly, our protocol delivers its guarantees without incurring in significant VM-execution overheads.

## F. DISCUSSION

Our protocol can be extended to accommodate a more generic and ambitious application. Instead of a single SEV VM, we could use the same principles to create a trusted deployer that sets up and attests an entire trusted (and possibly heterogeneous) infrastructure. Instead of having to attest the components of that infrastructure individually, possible using different protocols with varied levels of flexibility depending on the heterogeneity of the trusted components, the extended version of our protocol would allow a trusted deployer, with a flexible attestation mechanism and the capacity to deploy all the other components, to generate a single attestation report on the infrastructure's behalf. Broadly speaking, our protocol uses an SGX enclave to attest a SEV VM and establish a channel (via provisioning) that allows the SEV VM machine to communicate with the SGX enclave and generate SGX-like attestation reports (i.e. quotes). As most TEE implementations provide some form of attestation and provisioning and these primitives typically involve cryptographic operations that can be implemented by an SGX enclave, our trusted deployer could use the same attest-and-establish-secure-channel pattern implemented in this paper to generate quotes for other TEE implementations. Of course, the actual implementation

exibility for platforms that support SEV pre-SNP but do not support SEV SNP or TDX. analysis of the cryptographic protocol itself by assuming that the underlying primitives are trusted. These papers provide, then, guidelines to harden attestation primitives so that our assumptions are validated and our protocol can deliver on its other types of information. Our protocol creates an authenticated channel between trusted owner and SEV VM by sharing a shared MAC key. We could extend our protocol to create an authenticated and confidential channel between them by passing additionally a shared encryption key. The SEV VM and trusted owner could also have their APIs extended to exchange other pieces of verifiable information. For instance, they could both offer a remote function to provide a verifiable hardware-generated random string of bits. They could combine this string with a locally generated one to create a random "stronger" source of randomness.

Our protocol (and implementation) has also some limitations. A flaw in either of the TEE implementations that we rely upon can thwart the guarantees/goals of our protocol as we assume both SGX and SEV TCBs to be trusted. That limitation is inherent to any combination of TEE implementations that makes this assumption. Moreover, in terms of our scheme as our focus is on the interplay of SGX and SEV implementation, the SEV version that we use does not offer integrity protection; only SEV SNP gives integrity guarantees. We could implement our protocol using any SEV-like TEE implementation, with or without integrity protection, provided that the required attestation primitives are available.

In this section, we examine papers that focus on hardware-based TEEs and remote attestation protocols involving the hypervisor and its (non-SEV) VMs, with a formal proof of a number of applications and extensions to the SGX attestation protocols have been proposed. From incorporating attestation information into the TLS protocol [29], to proposing exible attestation verification infrastructures [15], to proposing exible mutual attestation protocols [14]. Kueh et al. [30] propose a protocol that involves similar parties but is very different in many ways to ours. They use SGX attestation to perform an integrity check on the system of (non-SEV) VMs at startup. Another line of research consists of identifying vulnerabilities and attacks specifically targeting attestation primitives [11], [55], [64]. Swami has shown that some of the privacy guarantees are thwarted by Intel's EPID design [55]. Buhren et al. [11] has shown how the PSP firmware can be updated to a version that allows the extraction of cryptographic keys managed by the PSP. We et al. [64] have shown how the memory-permutation insensitivity of the SEV launch measurement can be exploited in a way that allows the VM to execute arbitrary code and yet its original launch measurement remains unchanged. We regard these works as complementary to ours. The findings about SGX's speaking, these works implement virtual TPMs (vTPMs) in EPID can improve its privacy guarantees, and as a consequence, the benefits it could bring if it was used as part of our protocol. The other two SEV attacks are prevented by our protocol assumptions requiring the attested SEV TCB to be trusted and platform to not be compromised; we focus on the

real. have proposed a new scheme to remotely attest a hypervisor and its (non-SEV) VMs, with a formal proof of their authorized linked attestation protocol [6]. Their protocol design, trust assumptions, threat model, and protocol goals are completely different from ours.

We have found only another work that combines different TEE architectures. Zha et al. [65] propose a framework, called vSGX, by which one can emulate the behaviour of an SGX enclave inside an SEV VM. The main purpose of that work is to allow unmodified SGX enclave binaries to run on SEV hardware. Thus, they do not combine TEE implementations like we do, but they implement the execution model specific to a TEE architecture on top of another. The scheme that they propose for remote attestation relies on provider privacy guarantees are thwarted by Intel's EPID design [55]. to provision vSGX enclaves with "fused secrets." Note that Buhren et al. [11] has shown how the PSP firmware can secretly providing this "fused secret" requires the directed be updated to a version that allows the extraction of the rigid SEV remote attestation. That framework could move away from such directed and provider-centric attestation to a more exible one by employing our protocol to carry the remote attestation of their virtual enclaves.

Some works have investigated how to seamlessly provide standardised TPM features to SEV VMs [44], [46]. Broadly speaking, these works implement virtual TPMs (vTPMs) in the isolated memory space of the confidential VM. They are mainly concerned in solving the problem of heterogeneous interfaces and offering new features such as providing runtime measurement in addition to launch measurement and offering new features such as providing a unified TPM interface. Their work, however, does

not address the limitations of the SEV pre-SNP remote attestation like we do here; we are interested in making remote attestation, specifically, executable. In this sense, our work could be used to improve on their vTPM implementations for SEV pre-SNP. Their work, like ours, could be applied to create a unified interface offering security features to a heterogeneous infrastructure of TEEs. Some efforts are being put into the creation of libraries to simplify the use of TEEs [3].

We propose a cryptographic protocol that explores a synergy between SGX and SEV: it brings together the executability of SGX's remote attestation to the application portability of SEV neither of these two TEE implementations offer this combination of features independently. Our protocol relies on the notion of a trusted guest owner implemented in an SGX enclave, that is in charge of deploying, attesting, and provisioning an SEV VM. The latter can rely on the former to generate attestation reports on its behalf. Moreover, we formally demonstrate that our protocol enforces security properties related to the authenticity of quotes and confidentiality of provisioned secrets using Tamarin. Furthermore, we demonstrate with an application to machine-learning-models accountability how it can be used in practice while incurring negligible overheads.

We plan to further explore the extensions to our protocol that are required to apply it to the remote attestation of an infrastructure of heterogeneous trusted components.

We thank the anonymous reviewers for the valuable feedback they provided which helped improve this paper.

- [1] SEV-Tool repository, 2022. Available at: <https://github.com/AMDESE/sev-tool>.
- [2] Paper repository, 2023. Available at: <https://github.com/blockhousetech/sgx-sev-burrito>.
- [3] The Certifier Framework for Confidential Computing Whitepaper, 2023. Available at: <https://github.com/vmware-research/certifier-framework-for-confidential-computing/blob/752e8335395d527678baf29f8cab379569d885bc/Doc/RepositoryWhitepaper.pdf>.
- [4] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015. Software available from tensorflow.org.
- [5] Advanced Micro Devices, Inc. *AMD64 Architecture Programmer's Manual Volume 2: System Programming*, 2021. Available at: <https://www.amd.com/system/les/TechDocs/24593.pdf>.
- [6] Ghada Arfaoui, Pierre-Alain Fouque, Thibaut Jacques, Pascal Lafourcade, Adina Nedelcu, Cristina Onete, and Léo Robert. A cryptographic view of deep-attestation, or how to do provably-secure layer-linking. In Giuseppe Ateniese and Daniele Venturi, editors, *Applied Cryptography and Network Security*, pages 399–418, Cham, 2022. Springer International Publishing.
- [7] David Basin, Jannik Dreier, Lucca Hirschi, Saša Radomirovic, Ralf Sasse, and Vincent Stettler. A formal analysis of 5g authentication. *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security CCS '18*, page 1383–1396, New York, NY, USA, 2018. Association for Computing Machinery.
- [8] David A. Basin, Ralf Sasse, and Jorge Toro-Pozo. The EMV standard: Break, x, verify. In *42nd IEEE Symposium on Security and Privacy, SP 2021*, San Francisco, CA, USA, 24–27 May 2021, pages 1766–1781. IEEE, 2021.
- [9] Andrea Biondo, Mauro Conti, Lucas Davi, Tommaso Frassetto, and Ahmad-Reza Sadeghi. The guard's dilemma: Efficient code-reuse attacks against intel sgx. In *Proceedings of the 27th USENIX Conference on Security Symposium SEC'18*, page 1213–1227, USA, 2018. USENIX Association.
- [10] Ernie Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. In *Proceedings of the 11th ACM Conference on Computer and Communications Security CCS '04*, page 132–145, New York, NY, USA, 2004. Association for Computing Machinery.
- [11] Robert Buhren, Christian Werling, and Jean-Pierre Seifert. Insecure until proven updated: Analyzing AMD sev's remote attestation. *CoRR*, abs/1908.11680, 2019.
- [12] Claudio Canella, Daniel Genkin, Lukas Giner, Daniel Gruss, Moritz Lipp, Marina Minkin, Daniel Moghimi, Frank Piessens, Michael Schwarz, Berk Sunar, Jo Van Bulck, and Yuval Yarom. Fallout: Leaking data on meltdown-resistant cpus. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security CCS '19*, page 769–784, New York, NY, USA, 2019. Association for Computing Machinery.
- [13] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H. Lai. Sgxpectre: Stealing intel secrets from sgx enclaves via speculative execution. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 142–157, 2019.
- [14] Guoxing Chen and Yinqian Zhang. MAGE: Mutual attestation for a group of enclaves without trusted third parties. *31st USENIX Security Symposium (USENIX Security 22)*, Boston, MA, August 2022. USENIX Association.
- [15] Guoxing Chen, Yinqian Zhang, and Ten-Hwang Lai. Opera: Open remote attestation for intel's secure enclaves. *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security CCS '19*, page 2317–2331, New York, NY, USA, 2019. Association for Computing Machinery.
- [16] Cr Cohen, Josh Eads, James Forshaw, and Felix Wilhelm. Intel trust domain extensions (TDX) security review. Technical report, Google Inc., April 2023.
- [17] Victor Costan and Srinivas Devadas. Intel sgx explained. *ACM Cryptol. ePrint Arch.*, 2016(86):1–118, 2016.
- [18] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. A comprehensive symbolic analysis of tls 1.3. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security CCS '17*, page 1773–1788, New York, NY, USA, 2017. Association for Computing Machinery.
- [19] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 39(2):198–208, 1983.
- [20] S. Gueron. Memory encryption for general-purpose processors. *IEEE Security Privacy* 14(6):54–62, 2016.
- [21] Christian Göttel, Rafael Pires, Isabella Rocha, Sébastien Vaucher, Pascal Felber, Marcelo Pasin, and Valerio Schiavoni. Security, performance and energy trade-offs of hardware-assisted memory protection mechanisms. In *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*, pages 133–142, 2018.
- [22] Intel Corporation. Intel(R) 64 and IA-32 Architectures Software Developer's Manual Volume 3D: System Programming Guide, Part 20. Available at: <https://software.intel.com/content/dam/develop/external/us/en/documents-tps/332831-sdm-vol-3d.pdf>.
- [23] Intel Corporation. Intel(R) Software Guard Extensions Developer Guide 2020. Available at: [https://download.01.org/intel-sgx/sgx-linux/2.12/docs/Intel\\_SGX\\_Developer\\_Guide.pdf](https://download.01.org/intel-sgx/sgx-linux/2.12/docs/Intel_SGX_Developer_Guide.pdf).
- [24] Intel Corporation. Intel(R) Software Guard Extensions Developer Reference for Linux® OS2020. Available at: [https://download.01.org/intel-sgx/sgx-linux/2.12/docs/Intel\\_SGX\\_Developer\\_Reference\\_Linux\\_2.12\\_Open\\_Source.pdf](https://download.01.org/intel-sgx/sgx-linux/2.12/docs/Intel_SGX_Developer_Reference_Linux_2.12_Open_Source.pdf).
- [25] Intel Corporation. Intel(R) Trust Domain Extension 2022. Available at: <https://cdrdv2.intel.com/v1/dl/getContent/690419>.



- [26] Simon Johnson, Vincent Scarlata, Carlos V. Rozas, Ernie Brickell, Francis X. McKeen, and Ernie Brickell. Intel® software guard extensions: Epid provisioning and attestation services. 2016.
- [27] David Kaplan. Protecting VM Register State with SEV-ES. Advanced Micro Devices, Inc, 2017. Available at: <https://www.amd.com/system/les/TechDocs/Protecting%20VM%20Register%20State%20with%20SEV-ES.pdf>.
- [28] David Kaplan, Jeremy Powell, and Tom Woller. AMD Memory Encryption. Advanced Micro Devices, Inc, 2021. Available at: [https://developer.amd.com/wordpress/media/2013/12/AMD\\_Memory\\_Encryption\\_Whitepaper\\_v9-Public.pdf](https://developer.amd.com/wordpress/media/2013/12/AMD_Memory_Encryption_Whitepaper_v9-Public.pdf).
- [29] Thomas Knauth, Michael Steiner, Somnath Chakrabarti, Li Lei, Cedric Xing, and Mona Vij. Integrating remote attestation with transport layer security. *CoRR*, abs/1801.05863, 2018.
- [30] Michał Kucab, Piotr Boryło, and Piotr Chodura. Remote attestation and integrity measurements with intel sgx for virtual machines. *Computers & Security* 106:102300, 2021.
- [31] Mengyuan Li, Yinqian Zhang, and Zhiqiang Lin. Crossline: Breaking "security-by-crash" based memory isolation in amd sev. *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security CCS '21*, page 2937 2950, New York, NY, USA, 2021. Association for Computing Machinery.
- [32] Mengyuan Li, Yinqian Zhang, Zhiqiang Lin, and Yan Solihin. Exploiting unprotected I/O operations in AMD's secure encrypted virtualization. In *28th USENIX Security Symposium (USENIX Security 2019)*, pages 1257 1272, Santa Clara, CA, August 2019. USENIX Association.
- [33] Zecheng Li, Bin Xiao, Songtao Guo, and Yuanyuan Yang. Securing deployed smart contracts and devices with distributed tee clusters. *IEEE Transactions on Parallel and Distributed Systems* 33(3):828 842, 2023.
- [34] P. Maene, J. Götzfried, R. de Clercq, T. Müller, F. Freiling, and I. Verbauwhede. Hardware-based trusted computing architectures for isolation and attestation. *IEEE Transactions on Computers* 67(3):361 374, 2018.
- [35] Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. The tamarin prover for the symbolic analysis of security protocols. In *Natasha Sharygina and Helmut Veith, editors, Computer Aided Verification*, pages 696 701, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
- [36] Microsoft Corporation. What is guest attestation for confidential VMs?. 2022. Available at: <https://learn.microsoft.com/en-us/azure/confidential-computing/guest-attestation-confidential-vms>.
- [37] Microsoft Corporation. DCasv5 and ECasv5 series confidential VMs 2023. Available at: <https://learn.microsoft.com/en-us/azure/confidential-computing/confidential-vm-overview>.
- [38] Microsoft Corporation. Microsoft Azure Attestation 2023. Available at: <https://learn.microsoft.com/en-us/azure/attestation/overview>.
- [39] Saeid Mofrad, Fengwei Zhang, Shiyong Lu, and Weidong Shi. A comparison study of intel sgx and amd memory encryption technology. *HASP '18*, New York, NY, USA, 2018. Association for Computing Machinery.
- [40] Mathias Morbitzer, Manuel Huber, and Julian Horsch. Extracting Secrets from Encrypted Virtual Machines. *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security CCS '19*, page 221 230, New York, NY, USA, 2019. Association for Computing Machinery.
- [41] Mathias Morbitzer, Manuel Huber, Julian Horsch, and Sascha Wessel. Severed: Subverting amd's virtual machine encryption. *Proceedings of the 11th European Workshop on Systems Security EuroSec '18*, New York, NY, USA, 2018. Association for Computing Machinery.
- [42] Mathias Morbitzer, Sergej Proskurin, Martin Radev, Marko Dorfhuber, and Erick Quintanar Salas. Severity: Code injection attacks against encrypted virtual machines. In *2021 IEEE Security and Privacy Workshops (SPW)*, pages 444 455, 2021.
- [43] James Ménétreay, Christian Göttel, Marcelo Pasin, Pascal Felber, and Valerio Schiavoni. An exploratory study of attestation mechanisms for trusted execution environments. 2022.
- [44] Vikram Narayanan, Claudio Carvalho, Angelo Ruocco, Gheorghe Almási, James Bottomley, Mengmei Ye, Tobin Feldman-Fitzthum, Daniele Buono, Hubertus Franke, and Anton Burtsev. Remote attestation of sev-snp confidential vms using e-vmpls, 2023.
- [45] Arto Niemi, Sampo Sovio, and Jan-Erik Ekberg. Towards interoperable enclave attestation: Learnings from decades of academic work. In *31st Conference of Open Innovations Association (FRUCT)*, pages 189 200, 2022.
- [46] Joana Pecholt and Sascha Wessel. Cocotpm: Trusted platform modules for virtual machines in confidential computing environments. *Proceedings of the 38th Annual Computer Security Applications Conference ACSAC '22*, page 989 998, New York, NY, USA, 2022. Association for Computing Machinery.
- [47] Sandro Pinto and Nuno Santos. Demystifying arm trustzone: A comprehensive survey. *ACM Comput. Surv.* 51(6), January 2019.
- [48] Martin Radev and Mathias Morbitzer. Exploiting interfaces of secure encrypted virtual machines. *Reversing and Offensive-Oriented Trends Symposium ROOTS'20*, page 1 12, New York, NY, USA, 2020. Association for Computing Machinery.
- [49] Muhammad Sardar, Thomas Fossati, and Simon Frost. Sok: Attestation in confidential computing, 01 2023.
- [50] Muhammad Usama Sardar, Rasha Faqeh, and Christof Fetzer. Formal foundations for intel sgx data center attestation primitives. *Formal Methods and Software Engineering: 22nd International Conference on Formal Engineering Methods, ICFEM 2020, Singapore, Singapore, March 1 3, 2021, Proceedings*, page 268 283, Berlin, Heidelberg, 2020. Springer-Verlag.
- [51] Muhammad Usama Sardar, Saidgani Musaev, and Christof Fetzer. Demystifying attestation in intel trust domain extensions via formal verification. *IEEE Access* 9:83067 83079, 2021.
- [52] Muhammad Usama Sardar, Do Le Quoc, and Christof Fetzer. Towards formalization of enhanced privacy id (epid)-based remote attestation in intel sgx. In *2020 23rd Euromicro Conference on Digital System Design (DSD)*, pages 604 607, 2020.
- [53] Vincent Scarlata, Simon Johnson, James Beaney, and Piotr Mijewski. Supporting third party attestation for intel sgx with intel® data center attestation primitives. 2018.
- [54] AMD SEV-SNP. Strengthening vm isolation with integrity protection and more. 2020.
- [55] Yogesh Swami. Sgx remote attestation is not sufficient. *Cryptology ePrint Archive, Paper 2017/736*, 2017. <https://eprint.iacr.org/2017/736>.
- [56] J. Van Bulck, D. Moghimi, M. Schwarz, M. Lippi, M. Minkin, D. Genkin, Y. Yarom, B. Sunar, D. Gruss, and F. Piessens. Lvi: Hijacking transient execution through microarchitectural load value injection. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 54 72, 2020.
- [57] Jo Van Bulck, Marina Minkin, O r Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the trusted sgx kingdom with transient out-of-order execution. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 991 1008, 2018.
- [58] Jo Van Bulck, Frank Piessens, and Raoul Strackx. Nemesis: Studying microarchitectural timing leaks in rudimentary cpu interrupt logic. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security CCS '18*, page 178 195, New York, NY, USA, 2018. Association for Computing Machinery.
- [59] Stephan van Schaik, Andrew Kwong, Daniel Genkin, and Yuval Yarom. SGAXe: How SGX fails in practice. <https://sgxeattack.com/>, 2020.
- [60] Jan Werner, Joshua Mason, Manos Antonakakis, Michalis Polychronakis, and Fabian Monrose. The severest of them all: Inference attacks against secure virtual enclaves. *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security Asia CCS '19*, page 73 85, New York, NY, USA, 2019. Association for Computing Machinery.
- [61] Stephan Wesemeyer, Christopher J.P. Newton, Helen Treharne, Ligu Chen, Ralf Sasse, and Jordan Whitehead. Formal analysis and implementation of a tpm 2.0-based direct anonymous attestation scheme. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security ASIA CCS '20*, page 784 798, New York, NY, USA, 2020. Association for Computing Machinery.
- [62] J. Whitehead, L. Chen, R. Sasse, S. Schneider, H. Treharne, and S. Wesemeyer. A symbolic analysis of ecc-based direct anonymous attestation. In *2019 IEEE European Symposium on Security and Privacy (EuroSec)*, pages 127 141, 2019.
- [63] Luca Wilke, Jan Wichelmann, Mathias Morbitzer, and Thomas Eisenbarth. Security: No security without integrity : Breaking integrity-free memory encryption with minimal assumptions. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1483 1496, 2020.
- [64] Luca Wilke, Jan Wichelmann, Florian Sieck, and Thomas Eisenbarth. Undeserved trust: Exploiting permutation-agnostic remote attestation. In *2021 IEEE Security and Privacy Workshops (SPW)*, pages 456 466, 2021.
- [65] S. Zhao, M. Li, Y. Zhang, and Z. Lin. vsgx: Virtualizing sgx enclaves on amd sev. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 687 702, Los Alamitos, CA, USA, may 2022. IEEE Computer Society.

