Towards a Privacy-preserving Attestation for Virtualized Networks

Ghada Arfaoui[1], Thibaut Jacques[1,2], Marc Lacoste[1], Cristina Onete[2], and Léo Robert[3]

[1] Orange
[2] XLIM, University of Limoges
[3] MIS, Université de Picardie Jules Verne

Abstract. TPM remote attestation allows to verify the integrity of the boot sequence of a remote device. *Deep Attestation* extends that concept to virtualized platforms by allowing to attest virtual components, the hypervisor, and the link between them. In multi-tenant environments, deep attestation solution offer security and/or efficiency, but no privacy.
In this paper, we propose a privacy preserving TPM-based deep attestation solution in multi-tenant environments, which provably guarantees: **(i) Inter-tenant privacy**: a tenant is cannot know whether other VMs outside its own are hosted on the same machine; **(ii) Configuration hiding**: the hypervisor's configuration, used during attestation, remains hidden from the tenants; and **(iii) Layer linking**: tenants can link hypervisors with the VMs, thus obtaining a guarantee that the VMs are running on specific hardware. We also implement our scheme and show that it is efficient despite the use of complex cryptographic tools.

Keywords: Deep Attestation · Multi-tenant · 5G · Privacy

1 Introduction

The use of virtualization has revolutionized mobile networks (5G and beyond). Virtual Network Functions (VNFs) can be easily added, removed, or migrated to form slices (also called tenants), propose new services on demand, and meet the heterogeneous, stringent requirements of verticals (*e.g.*, e-health, banking, etc.). The flexibility of virtualization, however, induces an inevitable loss of control and a need to regularly confirm that the resulting network can be trusted.

A solution recommended by the European Telecommunications Standards Institute (ETSI) is *remote attestation* [11]. It enables a prover to convince an authorized verifier that it conforms to some specifications (and thus that it has specific properties). In this paper, we focus on the type of attestation that verifies the integrity state of a component. Recently, at ACNS 2022, [3] proposed a solution to attest VNFs and their underlying infrastructure, offering both security and scalability *for single tenant use*, *i.e.*, one entity operates all VNFs *and* the underlying infrastructure. However, new mobile network generations are typically multi-tenant environments (*i.e.*, the operator provides slices / tenants to different verticals). In that setting, the solution of [3] becomes inefficient. Moreover, privacy concerns arise (*e.g.*, the operator does not want to reveal its network nodes configuration), which are not addressed.

1.1 Our contribution

We consider the typical multi-tenant architecture shown in Figure 1, which is equipped with a hardware Trusted Platform Module (TPM) and spawns a virtual TPM (vTPM) for each VM it manages. VMs can be operated by tenants, and one tenant can have multiple VMs. Every tenant has a dedicated verifier to perform attestation. Our work makes a triple contribution.

A new protocol. We propose a primitive called privacy-preserving multi-tenant attestation (PP-MTA), which provides attestation, but also layer-binding and privacy: *Inter-tenant privacy* (no tenant can learn if other tenants share the same platforms as its own VMs) and *Configuration hiding* (the hypervisor attestation convinces a tenant that the hypervisor is well-configured without revealing the configuration). These strong properties are achieved with no modification to the TPM, and rely on ZK-SNARKs, vector-commitment schemes, and secure-channel establishment.

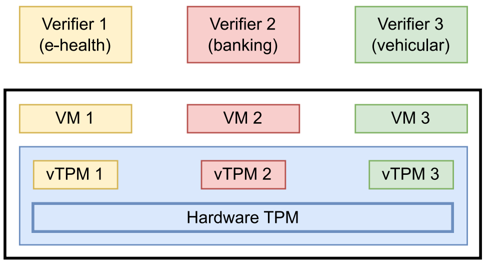


Fig. 1. Multi-tenant architecture where each VNFs belongs to a different tenant.

Formal analysis. We formally model and prove the security and privacy of our protocol. We *extend* the layer-binding properties defined in [3] to a multi-tenant environment, and *add* definitions for *inter-tenant privacy* and *configuration-hiding*. We formally quantify the privacy of our protocol. The security of our scheme relies on standard ACCE-secure channels, secure vector commitments, and zero-knowledge succinct non-interactive arguments of knowledge (ZK-SNARKs), but also two new properties: partner-hiding authenticated key-exchange (AKE) and collision-resistant vector commitments.

Implementation. We provide an implementation of our protocol, with several benchmarks. Despite relying on ZK-SNARKs, known for poor performance, our scheme remains fast enough for real-world use.

1.2 Background and Related work

Our work builds on TPM-based remote-attestation, for which we recall some basics below. We also recall the syntax of vector commitment and ZK-SNARK. Finally we review related work on attestation with a focus on [3].

Vector commitment. Introduced in [7], vector commitment schemes allow a user to commit to a list of values (rather than to a single message). Vector commitments can be opened partially, by index (opening information exists separately per committed value). After a setup phase $\mathsf{VC}.\mathtt{Setup}(1^\lambda, q) \rightarrow \mathsf{ppar}$, one can commit to a sequence of values $\mathsf{VC}.\mathtt{Com}(v) \rightarrow (c, aux)$. Given an opening $\mathsf{VC}.\mathtt{Open}(m, i, aux) \rightarrow \pi_i$, one can verify that a value is contained in the commitment $\mathsf{VC}.\mathtt{Ver}(m, c, i, \pi_i) \rightarrow b \in \{0, 1\}$.

ZK-SNARK. Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge (ZK-SNARKs) [5] are generic NIZK proof systems. Given an NP statement, one can prove knowledge of a valid witness without revealing it. A common reference string (CRS) is generated during setup: $\mathsf{ZKP}.\mathtt{Setup}(R) \rightarrow \mathsf{CRS}$. Given the CRS, a statement x_{ZK}, and a witness w_{ZK} such that $(x_{ZK}, w_{ZK}) \in R$, one can compute $\mathsf{ZKP}.\mathtt{Prove}(\mathsf{CRS}, R, x_{ZK}, w_{ZK}) \rightarrow \pi_{ZK}$ and verify the proof $\mathsf{ZKP}.\mathtt{SkVer}(\mathsf{CRS}, R, x_{ZK}, \pi_{ZK}) \rightarrow b \in \{0, 1\}$.

TPM Remote Attestation. This type of attestation, which allows the verification of the integrity state of a target, has two main phases: (1) the TPM measures all the code involved in the boot process and securely stores the measurements in the *Platform Configuration Registers (PCR)*; (2) upon request, the TPM signs with an attestation key (AK) the target configuration (PCR values) and sends the result. The receiver of the attestation checks the validity of the signature and of the PCR-values. Thus, classical attestation requires verifiers to know a full set of valid PCR-values (*i.e.*, valid configurations) – which, in some environments, is a privacy threat.

Property-based attestation. Property-based attestation (PBA) [22] aims to improve privacy in TPM remote attestation. Indeed, [22] showed that binary attestation can enable easier configuration leakage. PBA provides a privacy-friendly attestation mechanism by verifying that the target satisfies some high-level properties (rather than sending PCR measurements). A property can be achieved by multiple configurations. While some PBAs [20, 8] use a Trusted Third Party (TTP), its existence is not always guaranteed. Alternatively, some PBAs [9, 12] leverage zero-knowledge proofs of membership of a configuration among a set of valid configurations – at the expense of modifying TPM specifications. PBA achieves a different kind of privacy than Direct Anonymous Attestation

(DAA) [6]. While DAA prevents linking attestations to a specific TPM (or multiple attestation to the same TPM), PBA aims to protect the *content* of the attestation itself. In our work, we focus on classical attestation in a multi-tenant environment, for which DAA is unnecessary.

Deep Attestation. Attesting a virtual component (*e.g.,* a VM) implies attesting the underlying virtualization infrastructure, a process called *deep attestation* (DA). In *single-channel* DA [11, 27, 24, 19], the integrity of a VM is verified simultaneously with attesting its hypervisor, and the linking of the two components *layer binding.* This approach can ensure trust, scales badly for a large number of VMs. Conversely, *multiple-channel* DA [11, 27, 10] drops layer-binding to provide independent (and efficient) VM and hypervisor attestations.

Comparison to [3]. The DA scheme by [3] provides both layer-binding DA that scales as well as multiple-channel attestation, and requires no modification of the TPM. Hypervisor attestations also incidentally attest the public keys of the VMs they manage. As a result, the verifier can link those attestations with those of the VMs. Unfortunately, this solution does not scale in multi-tenant environments (e.g., 5G and beyond) for which *inter-tenant privacy* is required. Moreover, as the scheme builds on standard DA, the verifier learns, from the attestation quote, the current PCR-configurations of the physical machine. This is a privacy risk for the platform owner (e.g., the operator).

Attestation in the cloud. Although other works have explored attestation for trust in cloud-like environments [29, 4, 25, 30, 23, 21, 18], none focused on DA. Such solutions require full trust in the infrastructure provider. DA enables stronger security and privacy, making less trust assumptions on the provider.

2 Technical Overview

In this section we give an overview of our solution for DA in multi-tenant environments.

Use case and problem statement. We consider a use-case similar to that of Keylime [17, 24]. This solution found in cloud infrastructures provides system integrity monitoring based on TPM attestations. Keylime allows deploying an agent on each target VM: the agent sends, every few seconds, an attestation to a cloud verifier. Keylime is efficient even in very large cloud-infrastructures [14]. Unfortunately, it is limited to VM attestations (rather than to linked hypervisor/VM attestations). Moreover, it provides neither tenant privacy nor hypervisor configuration privacy.

Ideally, we would like to provide linkable attestation in multi-tenant settings, such that the resulting solution is practical (scalable and efficient), secure, provides strong privacy for both the tenants and for the provider of the physical infrastructure and requires no modification of the TPM.

Solution outline. A naïve application of [3] to multi-tenant scenarios provides layer-linking but not privacy. The result also scales poorly.

Hypervisor-configuration hiding requires that attestation quotes, signed by a physical root of trust (TPM), only prove the validity of the PCR-measurement, without revealing it. The naïve approach of the TPM computing a zero-knowledge proof would require TPM modifications. This is precisely what we want to avoid. In our solution, this is achieved by the use of ZK-SNARKs.

Moreover, the protocol in [3] is designed for environments where a single entity owns the infrastructure and all the VMs. Naïvely reusing that solution in multi-tenant environments creates a significant bottleneck when multiple tenants request attestations simultaneously; this could be resolved by *batching* [23] hypervisor attestations for multiple tenants. Batching is challenging to achieve simultaneously with inter-tenant privacy and layer-linking. As layer-linking requires VM attestations to be linked to their managing hypervisor's attestation, the latter has to include some binding information to the VMs it is managing. Yet, in multi-tenant environments such VMs might belong to distinct users; each user must only verify the binding of its own VMs to the hypervisor. To bridge that gap, we use vector-commitment schemes to store (in a hidden form) linking information to VMs hosted on the hypervisor. This allows each tenant to open some specific positions in that commitment, learning nothing about other positions.

Our solution. We introduce several new elements to the layer-linking DA solution of [3]. A trivial, but necessary modification is at setup: unlike [3], we need to account for the *ownership*, by a tenant, of a VM. In our infrastructure, tenants will have to use long-term credentials in order to register new VMs.

As VM attestations are independently requested by VM-owners, we can simply use typical multiple-channel attestation for this step. Our key contribution, however, is a novel hypervisor-attestation method which is scalable (as it allows for simultaneous attestation requests to be batched), linkable to VM attestations, and guaranteeing inter-tenant privacy and hypervisor-configuration hiding.

The hypervisor may receive one or more attestation requests from one or more tenants. Requests are buffered if the TPM is busy. As soon as the TPM is free, the hypervisor makes an attestation request for the aggregated queries, using a special nonce computed as follows.

For each tenant, the hypervisor retrieves linking information and concatenates it with that tenant's nonces (per each VM). Then it assigns a random position in a vector commitment to each tenant, places the concatenated linking-values for the tenant at that position, and commits to the resulting vector[4]. This is the nonce used in the attestation request.

The unmodified TPM computes and signs a regular attestation quote (revealing the PCR values). The hypervisor receives the signed quote and computes a ZK-SNARK confirming its validity without revealing the configuration. Then, the hypervisor computes, for each tenant, an opening to the vector for its attributed positions. Each tenant can verify the ZK-SNARK and use the opening information to retrieve its linking information, thus ensuring layer-linking.

3 Model

Our security model applies to the virtualization architecture shown in Figure 1. Tenants associated with unique identities $\mathcal{T}$ can register a number of virtual machines VM on a hypervisor $\mathcal{H}$.

Each hypervisor $\mathcal{H}$ has a physical root of trust represented by a TPM TPM. We assume each physical machine (with a unique hypervisor $\mathcal{H}$) upper-bounds the number of tenants $N_{\mathcal{T}}$ it can host, and the number of VMs N_{VM} each tenant can have on $\mathcal{H}$. Such bounds do exist in practice, usually driven by physical constraints. For the sake of legibility, we assume *universal* bounds (for all hypervisors), rather than *local*, hypervisor-specific ones.

We call the list of tuples of PCR measurements and accepted values during hypervisor attestation the *configuration* of the hypervisor. We assume the existence of a set (of more than one element) $\mathcal{CONF}$ of possible configurations for each hypervisor. In the quote, the current configuration is represented as the hash of the list of PCRs.

Security/privacy notions. We formally define the privacy properties required for our protocol and the adversary model in Sections 3.2 and 3.3. For attestation security we require an extension of the linking property formalized by [3], adapted to the multi-tenant setting. In a nutshell, this notion requires that no malicious party (even a malicious hypervisor) be able to fool a tenant into falsely believing that a VM is hosted by the hypervisor when in fact it is not. The full formalization of this property is in section 3.4.

3.1 Primitive Syntax

We formally define a new primitive called privacy-preserving multi-tenant attestation (PP-MTA). It consists of 9 PPT algorithms: PP-MTA= ($\mathtt{Setup}$, $\mathtt{HSetup}$, $\mathtt{TKGen}$, $\mathtt{VMReg}$, $\mathtt{HAttest}$, $\mathtt{VMAttest}$, $\mathtt{VfHAttest}$, $\mathtt{VfVMAttest}$, $\mathtt{Link}$) with:

$\underline{\mathtt{Setup}(1^{\lambda}) \to \{\mathsf{ppar}, \mathsf{spar}\}}$: On input of a security parameter λ, this algorithm outputs public parameters ppar (including the bounds $N_{\mathcal{T}}$, N_{VM}, and valid configuration-set $\mathcal{CONF}$), and private parameters spar (which may be instantiated to $\perp$ if not useful). The public parameters are input implicitly for every subsequent algorithm.

[4] If not all tenants simultaneously request attestations, or if the platform contains less tenants than its capacity, remaining vector positions are filled with dummy values.

$\underline{\texttt{HSetup}(\mathsf{ppar}) \to \{\mathcal{H}.\mathsf{pk}, \mathcal{H}.\mathsf{sk}, \mathsf{AK.pk}, \mathsf{AK.sk}, \mathcal{H}.\mathsf{Conf}, \mathcal{H}.\mathsf{state}\}}$: This algorithm sets up the (honest) hypervisor $\mathcal{H}$, by associating it with a public key $\mathcal{H}.\mathsf{pk}$, a private key $\mathcal{H}.\mathsf{sk}$, public- and private-attestation credentials $(\mathsf{AK.pk}, \mathsf{AK.sk})$, and a configuration $\mathcal{H}.\mathsf{Conf} \in \mathcal{CONF}$. The hypervisor "inherits" the universal bounds $N_{\mathcal{T}}$ and N_{VM} from ppar. The hypervisor maintains state $\mathcal{H}.\mathsf{state}$ related to hosted tenants and their VMs.

$\underline{\texttt{TKGen}(\mathsf{ppar}) \to \{\mathcal{T}.\mathsf{pk}, \mathcal{T}.\mathsf{sk}\}}$: This algorithm generates public and private keys for a single tenant $\mathcal{T}$. All parties have access to all the public keys. Only the tenant has access to its private key.

$\underline{\texttt{VMReg}(\mathcal{H}, \mathcal{T}.\mathsf{sk}, \mathsf{VMdesc}) \to \{(VM, \mathsf{VAK.pk}), \mathsf{VAK.sk}, \mathcal{H}.\mathsf{state}\} \cup \bot}$: This algorithm performs the registration, by tenant $\mathcal{T}$, of a VM of description VMdesc on the machine with hypervisor $\mathcal{H}$. If the tenant's request exceeds either the hypervisor's capacity to host new tenants $N_{\mathcal{T}}$, or its capacity for VMs for this tenant N_{VM}, then the algorithm returns $\bot$. Otherwise, the hypervisor creates the required VM, for which it returns a handle VM, as well as a tuple of public/private parameters, corresponding to the *attestation* keypair for that VM, as stored by the vTPM: $(\mathsf{VAK.pk}, \mathsf{VAK.sk})$. The algorithm also requires mutual authentication of the tenant and the hypervisor, enabling $\mathcal{H}$ to update its state $\mathcal{H}.\mathsf{state}$. If the authentication fails, the algorithm returns $\bot$. Otherwise it returns to the tenant the handle VM and the public keys and $\mathsf{VAK.pk}$.

$\underline{\texttt{HAttest}\langle \mathcal{T}(\mathcal{T}.\mathsf{sk}, \mathsf{nonce}_{\mathcal{T}}), \mathcal{H}(\mathcal{H}.\mathsf{sk}, \mathsf{AK.sk}, \mathcal{H}.\mathsf{state}, \mathcal{H}.\mathsf{Conf})\rangle \to \{\mathsf{ATT}_{\mathcal{H},\mathcal{T}}\}}$: The hypervisor attestation protocol is an interactive algorithm between a tenant $\mathcal{T}$ which takes as input its private key and a fresh nonce $\mathsf{nonce}_{\mathcal{T}}$ and the hypervisor with input its long-term credentials $\mathcal{H}.\mathsf{sk}$, $\mathsf{AK.sk}$, its current state $\mathcal{H}.\mathsf{state}$, its configuration $\mathcal{H}.\mathsf{Conf}$. It outputs an attestation $\mathsf{ATT}_{\mathcal{H},\mathcal{T}}$.

$\underline{\texttt{VfHAttest}(\mathsf{ATT}_{\mathcal{H},\mathcal{T}}, \mathsf{nonce}_{\mathcal{T}}, \mathsf{link}_{\mathcal{T}}) \to \{0,1\}}$: Given as input a hypervisor attestation $\mathsf{ATT}_{\mathcal{H},\mathcal{T}}$, a nonce $\mathsf{nonce}_{\mathcal{T}}$ and linking information $\mathsf{link}_{\mathcal{T}}$, this verification algorithm outputs 1 if the attestation is valid and 0 otherwise.

$\underline{\texttt{VMAttest}\langle VM(\mathsf{VAK.sk}), \mathcal{T}(\mathcal{T}.\mathsf{sk}, \mathsf{nonce})\rangle \to \{\mathsf{ATT}_{VM}\} \cup \bot}$: The interactive VM attestation protocol takes place between a tenant (using its key $\mathcal{T}.\mathsf{sk}$ and a fresh nonce nonce) and a VM that the tenant owns (associated with its private key $\mathsf{VAK.sk}$). The output could be $\bot$ (typically if the tenant does not own VM) or a VM attestation ATT_{VM}.

$\underline{\texttt{VfVMAttest}(\mathsf{ATT}_{VM}, \mathsf{nonce}, \mathsf{link}) \to \{0,1\}}$: Given as input a VM attestation ATT_{VM}, a nonce nonce and linking information link, the VM attestation-verification algorithm outputs 1 if the attestation is valid and 0 otherwise.

$\underline{\texttt{Link}(\mathsf{ATT}_{\mathcal{H},\mathcal{T}}, \mathsf{nonce}_{\mathcal{T}}, \mathsf{link}_{\mathcal{T}}, \mathsf{ATT}_{VM}, \mathsf{nonce}, \mathsf{link}) \to \{0,1\}}$: Given as input a tuple consisting of a hypervisor attestation quote $\mathsf{ATT}_{\mathcal{H},\mathcal{T}}$ and hypervisor attestation linking information $\mathsf{link}_{\mathcal{T}}$, and a tuple consisting of a VM attestation quote ATT_{VM} and VM attestation linking information link, the linking algorithm outputs 1 if the two attestation are linked and 0 if they are not.

3.2 Inter-tenant Privacy

Intuitively, inter-tenant privacy ensures that a malicious, but legitimate tenant cannot tell whether or not VMs from other tenants are running on the same hypervisor as its own VMs. In the real-world, tenants might be well-aware that they are sharing resources with other tenants. However, by considering a much stronger privacy notion, we ensure that this (and potentially other) information is not leaked by the attestation. This makes our protocol usable in all situations, not just those in which some leakage is acceptable. The definition we provide (and prove for our protocol) only guarantees that leakage is avoided at the protocol-level: the adversary may have alternative means of knowing about VMs co-hosted on the same hardware.

Formally, inter-tenant privacy is defined as a game (depicted in Figure 2) between a challenger $\mathcal{G}$ and an adversary $\mathcal{A}$. The challenger runs the setup algorithm $\texttt{Setup}(1^\lambda)$. It then sets up a hypervisor $\mathcal{H}$ by running $\texttt{HSetup}(\mathsf{ppar})$. $\mathcal{G}$ initiates $\mathcal{L}_H := \emptyset$ and $\mathcal{L}_C := \emptyset$. The challenger draws a random bit $b \overset{r}{\leftarrow} \{0,1\}$. The adversary, given ppar and the length of the security parameter (in unary) 1^λ, as well as the handle $\mathcal{H}$, can then use the following oracles:

$\underline{\mathsf{oHonTReg}_b(\{\mathsf{VMDesc}_i\}_{i=1}^{\ell})}$: this oracle depends on bit b. Given as input a set of VM descriptions VMDesc_i, this oracle internally runs the key-generation algorithm $\texttt{TKGen}(\mathsf{ppar})$, receiving either $\bot$ (too many tenants) or a handle $\mathcal{T}$ and keys $\mathcal{T}.\mathsf{pk}, \mathcal{T}.\mathsf{sk}$. The oracle adds $\mathcal{T}$ to $\mathcal{L}_H$ and increments a variable $n_{\mathcal{T}}$ (that stores the number of tenants on that hypervisor) by 1. Assuming that $\mathsf{oHonTReg}$ did not output $\bot$: if $b = 1$, the oracle runs $\texttt{VMReg}(\mathcal{H}, \mathcal{T}.\mathsf{sk}, \mathsf{VMDesc}_i)$ for each

VM in the input set, obtaining handles VM, keys $\mathsf{VAK.pk}, \mathsf{VAK.sk}$, and an updated hypervisor state $\mathcal{H}.\mathsf{state}$, containing tuples of the form $(\mathcal{T},\ VM_i,\ \mathsf{VAK.sk}_i,\ \mathsf{VAK.pk}_i,\ \mathsf{REAL})$ for each VM. If $b = 0$, then the VMs are not truly created: instead, the oracle generates random values $\mathsf{VAK.pk}_i$ for each $i = 1, \ldots, \ell$, and handles VM_i, updating the hypervisor state with tuples of the form $(\mathcal{T},\ VM_i, \mathsf{VAK.pk}_i, \mathsf{FAKE})$. Finally, the oracle outputs the following values to the adversary: $\mathcal{T}, \{\ VM_i\}_{i=1}^{\ell}$ as well as keys: $\mathcal{T}.\mathsf{pk}, \{\mathsf{VAK.pk}_i\}_{i=1}^{\ell}$. If $\ell > N_{VM}$, the output of $\mathtt{VMReg}$ will be $\perp$, forwarded to the adversary instead of the VM information. The adversary can, in parallel, use $\mathtt{TKGen}$ algorithm to register malicious tenants: these will be added by the challenger to $\mathcal{L}_C$.

$\underline{\mathsf{oVMReg}(\mathcal{T}, \mathsf{VMDesc})}$: given as input a (registered) tenant $\mathcal{T} \in \mathcal{L}_H$ and a VM with description VMDesc, this oracle internally runs $\mathtt{VMReg}(\mathcal{H}, \mathcal{T}.\mathsf{sk}, \mathsf{VMDesc})$. If the bound N_{VM} has still not been reached for tenant $\mathcal{T}$ then the algorithm outputs $(\ VM, \mathsf{VAK.pk})$ as in the previous oracle. The hypervisor state $\mathcal{H}.\mathsf{state}$ is updated. A malicious tenant can always register a new VM by running the $\mathtt{VMReg}$ algorithm directly.

$\underline{\mathsf{oHAttest}(\mathcal{T})}$: given as input a registered tenant $\mathcal{T} \in \mathcal{L}_H$, this oracle simulates running $\mathtt{HAttest}$ between $\mathcal{T}$ and the hypervisor $\mathcal{H}$. The adversary gains a transcript τ_{HAtt} of the protocol run (or $\perp$ if *e.g.*, $\mathcal{H}$ does not exist or if $\mathcal{T}$ has no VMs registered on $\mathcal{H}$). If the VMs created for this tenant were fake (the bit b picked by the challenger is 0), the hypervisor attestation is done over the current configuration and the VMs currently existing on the machine.

Since the adversary is a collusion of valid tenants, it does not need oracle access to VM attestations: it can simply run the correct algorithms.

The Inter-tenant Privacy Game $G_{\mathsf{TPriv}}(\lambda)$:

```
Game $\underline{G_{\mathsf{TPriv}}(\lambda)}$
$\{\mathsf{ppar}, \mathsf{spar}\} \leftarrow \mathtt{Setup}(1^\lambda)$
$\{\mathcal{H}.\mathsf{pk}, \mathcal{H}.\mathsf{sk}, \mathsf{HAK.pk}, \mathsf{HAK.sk}, \mathcal{H}.\mathsf{Conf}\}$,
$\mathcal{H}.\mathsf{state} \leftarrow \mathtt{HSetup}(\mathsf{ppar})$
$b \xleftarrow{r} \{0,1\}$
$d \leftarrow \mathcal{A}^{\mathsf{oHonTReg}_b(\cdot),\mathsf{oVMReg}(\cdot,\cdot),\mathsf{oHAttest}(\cdot)}(1^\lambda)$
$\mathcal{A}$ wins iff.: $d = b$
```

Fig. 2. The inter-tenant privacy game.

Definition 1 (Inter-tenant privacy). *A PP-MTA scheme* PP-MTA= *(*$\mathtt{Setup}$*,* $\mathtt{HSetup}$*,* $\mathtt{TKGen}$*,* $\mathtt{VMReg}$*,* $\mathtt{HAttest}$*,* $\mathtt{VMAttest}$*,* $\mathtt{VfHAttest}$*,* $\mathtt{VfVMAttest}$*,* $\mathtt{Link}$*) is* $(N_{\mathcal{T}}, N_{VM}, \epsilon)$-inter-tenant private *if, and only if, for every probabilistic polynomial adversary* $\mathcal{A}$*, the following holds:*

$$\mathsf{Adv}^{\mathsf{TPriv}}_{\mathsf{PP\text{-}MTA}}(\mathcal{A}) := \left| \Pr[\mathcal{A} \textit{ wins } G_{\mathsf{TPriv}}(\lambda)] - \frac{1}{2} \right| \leq \epsilon.$$

The value $\mathsf{Adv}^{\mathsf{TPriv}}_{\mathsf{PP\text{-}MTA}}(\mathcal{A})$ *is called the* advantage of $\mathcal{A}$ against the inter-tenant privacy of PP-MTA. *Asymptotically, we call a PP-MTA scheme* inter-tenant private *if* ϵ *is a negligible function of the security parameter* λ*.*

3.3 Hypervisor-Configuration Hiding

Intuitively, hypervisor-configuration ensures that an adversary (which can be a group of colluding, legitimate tenants) cannot learn the precise configuration of the hypervisor: just that this configuration is one of potentially many valid configurations. Technically, the property is formalized using a $\mathsf{oChooseConfig}$ oracle, which allows the adversary to choose two configurations, one of which will be used in fact for attestation. The adversary's task is to distinguish between those configurations.

In the hypervisor configuration-hiding game (figure 3), the adversary gets access to the following oracle:

– $\underline{\mathsf{oChooseConfig}_b(\mathcal{H}.\mathsf{Conf}_0, \mathcal{H}.\mathsf{Conf}_1) \to \{\mathsf{OK}\} \cup \perp}$: This oracle can only be called once. Given as input two hypervisor configurations $\mathcal{H}.\mathsf{Conf}_0$ and $\mathcal{H}.\mathsf{Conf}_1$, this oracle checks that $\mathcal{H}.\mathsf{Conf}_0 \in \mathcal{CONF}$ and $\mathcal{H}.\mathsf{Conf}_1 \in \mathcal{CONF}$. It ensure ensure that $\mathcal{H}$ has not yet been set up (*e.g.*, through `HSetup`). If either verification fails, the oracle outputs $\perp$. If verification succeeds, the oracle calls `HSetup`, forcing the picked hypervisor configuration $\mathcal{H}.\mathcal{H}.\mathsf{Conf}$ to be $\mathcal{H}.\mathsf{Conf}_b$.

The Hypervisor Privacy Game $G_{\mathsf{CPriv}}(\lambda)$:

```
Game $G_{\mathsf{CPriv}}(\lambda)$
$\{\mathsf{ppar}, \mathsf{spar}\} \leftarrow \mathtt{Setup}(1^\lambda)$
$b \xleftarrow{r} \{0,1\}$
$d \leftarrow \mathcal{A}^{\mathsf{oChooseConfig}_b(\cdot)}(1^\lambda)$
$\mathcal{A}$ wins iff.: $d = b$
```

Fig. 3. The configuration-privacy game.

Definition 2 (Configuration privacy). *A PP-MTA scheme* PP-MTA= *(*`Setup`*,* `HSetup`*,* `TKGen`*,* `VMReg`*,* `HAttest`*,* `VMAttest`*,* `VfHAttest`*,* `VfVMAttest`*,* `Link`*) is* ϵ-configurations-private *if, and only if, for every probabilistic polynomial adversary $\mathcal{A}$, the following holds:*

$$\mathsf{Adv}^{\mathsf{CPriv}}_{\mathsf{PP\text{-}MTA}}(\mathcal{A}) := \left| \Pr[\mathcal{A} \text{ wins } G_{\mathsf{CPriv}}(\lambda)] - \frac{1}{2} \right| \le \epsilon.$$

The value $\mathsf{Adv}^{\mathsf{CPriv}}_{\mathsf{PP\text{-}MTA}}(\mathcal{A})$ *is called the* advantage of $\mathcal{A}$ against the configuration privacy of PP-MTA. *Asymptotically, we call a PP-MTA scheme* configuration-private *if ϵ is a negligible function in the security parameter λ.*

Limitations. The security definition above is limited, formalizing that an adversary cannot distinguish between two valid configurations. Yet, clearly, the guarantee provided by the privacy property depends on the size of the configuration-set $\mathcal{CONF}$ – if it is small, then any tenant can guess the hypervisor configuration with a decent probability (equal to $\frac{1}{|\mathcal{CONF}|}$).

3.4 Linkability Security

The linking property ensures that two components, registered on two different platforms (later denoted $\mathbb{S}_1$ and $\mathbb{S}_2$), cannot be linked. For instance, a VM's attestation is linked to its hypervisor if both components are on the same platform, while other links from other platforms cannot be made and should be detected. Our model is directly inspired from [3].

We consider in our model some simplifications which ease the readability; notice that generalizations can be made:

– Malicious tenant: our adversary is (the only) tenant. So $\mathcal{A}$ has external capabilities with the possibility of registering VM and attesting them (same as a tenant would). Note that we consider only one tenant in our security game (or equivalently, the adversary represents all the tenants);
– The setup is made for only two platforms, each including only one hypervisor and a maximum of N_{VM} VM. Since we consider only one tenant (so $N_{\mathcal{T}} := 1$ for each platform), there is no need to let the adversary adaptively register VMs or platforms. However, each position are created for each tenant so only one tenant would lead to only one position. Thus, the challenger creates "dummy" tenants (which are not active) to allow more than one position in the vector commitment.
– The adversary does not have a corrupt oracle, all the VMs are already registered during the setup and accessible to $\mathcal{A}$. In particular, we suppose that hypervisors are always honest.

The linkability property is formalized through a security game, $G_{\mathsf{Link}}(\lambda, N_{\mathcal{P}})$, played between a challenger $\mathcal{G}$ and an adversary $\mathcal{A}$. The challenger runs the setup algorithm $\texttt{Setup}(1^\lambda)$, returns ppar to $\mathcal{A}$, then sets up an hypervisor $\mathcal{H}$ by running $\texttt{HSetup}(\mathsf{ppar})$ on both platforms $\mathbb{S}_1$ and $\mathbb{S}_2$. Then, $\mathcal{G}$ initiates $\mathcal{L}_{Att} := \emptyset$ which consists of a list of linkable attestations. The adversary then plays the game using the following oracles:

– $\underline{\mathsf{oHAttest}(\mathbb{S}_i) \rightarrow (\mathsf{ATT}_{\mathcal{H},\mathcal{T}})}$: this oracle simulates a run of the $\texttt{HAttest}$ algorithm between the adversary and the hypervisor $\mathcal{H}$ on platform $\mathbb{S}_i$ for $i \in \{1, 2\}$, allowing the adversary to gain a transcript τ_{HAtt} of the communication. Notice that this oracle does not depend on the challenge bit b. The output is stored in $\mathcal{L}_{Att}$.
– $\underline{\mathsf{oVMAttest}(VM) \rightarrow (\mathsf{ATT}_{VM})}$: this oracle simulates a run of the $\texttt{VMAttest}$ algorithm on VM. The output is stored in $\mathcal{L}_{Att}$.

At the end of the game, $\mathcal{A}$ outputs a party $\mathcal{P}$ such that its attestation is stored in $\mathcal{L}_{Att}$. We say that $\mathcal{A}$ wins the game if the following conditions hold:

– $\mathcal{P}$ is registered on $\mathbb{S}_i$ for $i \in \{0, 1\}$;
– It exists $\mathcal{Q} \in \mathbb{S}_{j \neq i}$ such that its attestation lies in $\mathcal{L}_{Att}$;
– $\texttt{Link}(\mathcal{P}||\mathcal{Q}) = 1$.

Thus the adversary wins the game if it is able to store two attestation's component in $\mathcal{L}_{Att}$ for two components on different platforms. The linkability property ensures that the probability of winning, for any adversary, is negligible.

Definition 3 (Linkability security). *A PP-MTA scheme* PP-MTA= *(*$\texttt{Setup}$*,* $\texttt{HSetup}$*,* $\texttt{TKGen}$*,* $\texttt{VMReg}$*,* $\texttt{HAttest}$*,* $\texttt{VMAttest}$*,* $\texttt{VfHAttest}$*,* $\texttt{VfVMAttest}$*,* $\texttt{Link}$*) is* $(q_{att}, N_{\mathcal{P}}, \epsilon)$-linkable *if, and only if, for every probabilistic polynomial adversary* $\mathcal{A}$*, the following holds:*

$$\mathsf{Adv}^{\mathsf{Link}}_{\mathsf{PP\text{-}MTA}}(\mathcal{A}) := \Pr[\mathcal{A} \textit{ wins } G_{\mathsf{Link}}(\lambda, N_{\mathcal{P}})] \leq \epsilon.$$

The value $\mathsf{Adv}^{\mathsf{Link}}_{\mathsf{PP\text{-}MTA}}(\mathcal{A})$ *is called the* advantage of $\mathcal{A}$ against the linkability security of PP-MTA*. Asymptotically, we call a PP-MTA scheme* linkable *if* ϵ *is a negligible function of the security parameter* λ*.*

4 Construction

In this section we instantiate the PP-MTA primitive using a signature scheme ($\texttt{SigKGen}, \texttt{SigSig}, \texttt{SigVer}$), a collision-resistant hash function H, a vector commitment scheme ($\mathsf{VC}.\texttt{Setup}$, $\mathsf{VC}.\texttt{Com}$, $\mathsf{VC}.\texttt{Open}$, $\mathsf{VC}.\texttt{Ver}$), a ZK-SNARK scheme ($\mathsf{ZKP}.\texttt{Setup}$, $\mathsf{ZKP}.\texttt{Prove}$, $\mathsf{ZKP}.\texttt{SkVer}$, $\mathsf{ZKP}.\texttt{SkSim}$), and a secure-channel establishment protocol : AKE=($\mathsf{AKE}.\texttt{KGen}$, $\mathsf{AKE}.\texttt{AKE}$, $\mathsf{AKE}.\texttt{Enc}$, $\mathsf{AKE}.\texttt{Dec}$).

4.1 Setup

We first instantiate the global-setup and hypervisor-setup algorithms ($\texttt{Setup}$, $\texttt{HSetup}$), then set up the tenants with long-term parameters.

Global setup. The goal is to instantiate the scheme's global public and private parameters: $\{\mathsf{ppar}, \mathsf{spar}\} \leftarrow \texttt{Setup}(1^\lambda)$. Two universal bounds are chosen: a maximal number of tenants per physical machine $N_{\mathcal{T}}$ and a maximal number of VMs per hosted tenant N_{VM}. These bounds are later crucial in avoiding trivial inter-tenant privacy attacks. A set of *plausible* configurations $\mathcal{CONF}$ is chosen for the hypervisor. We set up the vector commitment and ZK-SNARK:

$$(\mathsf{ppar}_{\mathsf{VC}}) \leftarrow \mathsf{VC}.\texttt{Setup}(1^\lambda, N_{\mathcal{T}})$$
$$(CRS, \tau) \leftarrow \mathsf{ZKP}.\texttt{Setup}(R)$$

The vector-commitment length is a constant $N_{\mathcal{T}}$. Given the following zero-knowledge proof:

$$\mathsf{ZK\text{-}SNARK}\{(\mathsf{quote}, \sigma) : \texttt{SigVer}(\mathsf{AK.pk}, \mathsf{quote}, \sigma, c) == 1 \ \wedge\ \mathsf{quote}.\mathcal{H}.\mathsf{Conf} \in \mathcal{CONF}\}$$

we fix the statement :

$$(x_{ZK}) \leftarrow \{\texttt{SigVer}(\mathsf{AK.pk}, \mathsf{quote}, \sigma, c) == 1; \wedge \mathsf{quote}.\mathcal{H}.\mathsf{Conf} \in \mathcal{CONF}\}$$

At the end of the global setup, we set $\mathsf{ppar} := (N_{\mathcal{T}},\ N_{VM},\ \mathcal{CONF},\ \mathsf{ppar}_{\mathsf{VC}},\ CRS,\ x_{ZK})$ and $\mathsf{spar} := \tau$.

Hypervisor setup. We instantiate the algorithm $\{\mathcal{H}.\mathsf{pk},\ \mathcal{H}.\mathsf{sk},\ \mathcal{H}.\mathsf{Conf}\} \leftarrow \texttt{HSetup}(\mathsf{ppar})$ as follows. To begin with the hypervisor will require two pairs of keys, one for the AKE protocol, the other, for attestation, as follows:

$$(\mathcal{H}.\mathsf{pk}, \mathcal{H}.\mathsf{sk}) \leftarrow \mathsf{AKE}.\texttt{KGen}(\mathsf{ppar})$$

$$(\mathsf{AK.pk}, \mathsf{AK.sk}) \leftarrow \texttt{SigKGen}(\mathsf{ppar})$$

The hypervisor then picks uniformly at random a configuration $\mathcal{H}.\mathsf{Conf} \overset{r}{\leftarrow} \mathcal{CONF}$ and sets $\mathcal{H}.\mathsf{state} = \emptyset$.

Tenant setup. The tenants generate long-term keys $\{\mathcal{T}.\mathsf{pk}, \mathcal{T}.\mathsf{sk}\} \leftarrow \texttt{TKGen}(\mathsf{ppar})$ which are, in fact, AKE keys:

$$(\mathcal{T}.\mathsf{pk}, \mathcal{T}.\mathsf{sk}) \leftarrow \mathsf{AKE}.\texttt{KGen}(\mathsf{ppar})$$

4.2 Registration

Registration is run by a tenant and a hypervisor, to register a VM of a given description on the given hypervisor:

$$\{(VM, \mathsf{VAK.pk}, \mathsf{VAK.sk}), \mathcal{H}.\mathsf{state}\} \cup \perp \leftarrow \texttt{VMReg}(\mathcal{H}, \mathcal{T}.\mathsf{sk}, \mathsf{VMdesc})$$

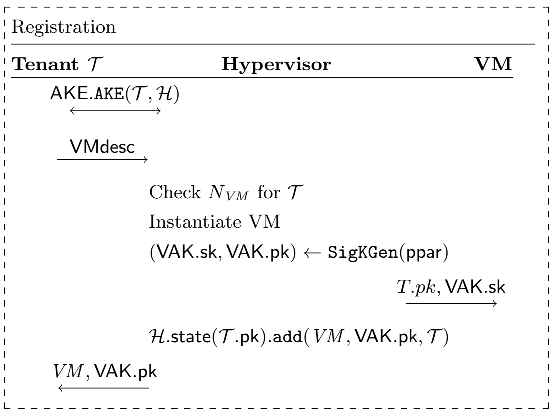

Registration
Tenant $\mathcal{T}$
Hypervisor
VM
$\mathsf{AKE}.\texttt{AKE}(\mathcal{T}, \mathcal{H})$
VMdesc
Check N_{VM} for $\mathcal{T}$
Instantiate VM
$(\mathsf{VAK.sk}, \mathsf{VAK.pk}) \leftarrow \texttt{SigKGen}(\mathsf{ppar})$
$T.pk$, VAK.sk
$\mathcal{H}.\mathsf{state}(\mathcal{T}.\mathsf{pk}).\mathsf{add}(VM, \mathsf{VAK.pk}, \mathcal{T})$
VM, VAK.pk

Fig. 4. Registration, the tenant (successfully) registers a new VM of description VMdesc.

As depicted in Figure 4, when a registration request is made, $\mathcal{H}$ and $\mathcal{T}$ run $\mathsf{AKE}.\texttt{AKE}$ (using their long-term credentials) to establish a secure channel, over which they can communicate in a confidential and authentic manner. Over this secure channel, $\mathcal{T}$ requests the registration of a VM with description VMdesc. The hypervisor verifies it can still allow tenants to register a new VM (w.r.t.N_{VM}). If this is not so, the algorithm aborts. Otherwise, the hypervisor generates attestation (signature) keys for the newly-registered VM: $(\mathsf{VAK.sk}, \mathsf{VAK.pk}) \leftarrow \texttt{SigKGen}(\mathsf{ppar})$. The keys VAK.sk and $\mathcal{T}.\mathsf{pk}$ will be stored in the vTPM corresponding to the new VM.

Once the VM is created, $\mathcal{H}$ updates its internal state $\mathcal{H}.\mathsf{state}$ with entries $(VM, \mathsf{VAK.pk}, \mathcal{T})$. Still over the secure channel, the hypervisor sends the VM handle VM and public keys VAK.pk to $\mathcal{T}$.

4.3 Hypervisor Attestation

We instantiate this algorithm as $\{\mathsf{ATT}_{\mathcal{H},\mathcal{T}}\} \leftarrow \mathtt{HAttest}\langle\mathcal{T}(\mathcal{T}.\mathsf{sk}, \mathsf{nonce}_{\mathcal{T}}), \mathcal{H}(\mathcal{H}.\mathsf{sk}, \mathsf{AK}.\mathsf{sk}, \mathcal{H}.\mathsf{state}, \mathcal{H}.\mathsf{Conf})\rangle$. The idea is to attest (in a configuration-hiding way) the hypervisor and also to embed in that attestation elements that characterise each managed VM. During hypervisor attestation, the latter retrieves the public attestation-key stored on each of the vTPMs it manages[5]. Those values are concatenated with the nonce and hashed to obtain a new nonce.

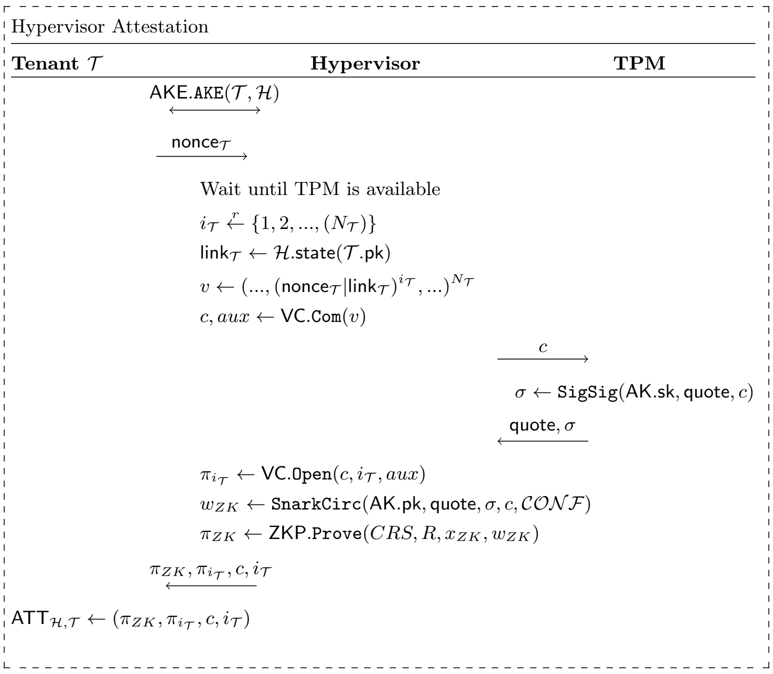

Hypervisor Attestation
Tenant $\mathcal{T}$
Hypervisor
TPM
$\mathsf{AKE}.\mathtt{AKE}(\mathcal{T},\mathcal{H})$
$\mathsf{nonce}_{\mathcal{T}}$
Wait until TPM is available
$i_{\mathcal{T}} \overset{r}{\leftarrow} \{1, 2, ..., (N_{\mathcal{T}})\}$
$\mathsf{link}_{\mathcal{T}} \leftarrow \mathcal{H}.\mathsf{state}(\mathcal{T}.\mathsf{pk})$
$v \leftarrow (..., (\mathsf{nonce}_{\mathcal{T}}|\mathsf{link}_{\mathcal{T}})^{i_{\mathcal{T}}}, ...)^{N_{\mathcal{T}}}$
$c, aux \leftarrow \mathsf{VC}.\mathtt{Com}(v)$
c
$\sigma \leftarrow \mathtt{SigSig}(\mathsf{AK}.\mathsf{sk}, \mathsf{quote}, c)$
quote, σ
$\pi_{i_{\mathcal{T}}} \leftarrow \mathsf{VC}.\mathtt{Open}(c, i_{\mathcal{T}}, aux)$
$w_{ZK} \leftarrow \mathtt{SnarkCirc}(\mathsf{AK}.\mathsf{pk}, \mathsf{quote}, \sigma, c, \mathcal{CONF})$
$\pi_{ZK} \leftarrow \mathsf{ZKP}.\mathsf{Prove}(CRS, R, x_{ZK}, w_{ZK})$
$\pi_{ZK}, \pi_{i_{\mathcal{T}}}, c, i_{\mathcal{T}}$
$\mathsf{ATT}_{\mathcal{H},\mathcal{T}} \leftarrow (\pi_{ZK}, \pi_{i_{\mathcal{T}}}, c, i_{\mathcal{T}})$

Fig. 5. Hypervisor Attestation, only the *i*-th tenant is represented but we can see the aggregation of all the nonce through commitment which allow a single TPM operation.

Authenticated key-exchange. To start, the hypervisor and tenant establish a mutually authenticated secure channel, over which all subsequent communication takes place.

Preparation of vector commitment. Multiple tenants (authenticated over a secure channel) may request attestations simultaneously, each providing a nonce to the hypervisor. The hypervisor randomly associates each tenant with an index $i \in \{1, \dots N_{\mathcal{T}}\}$. It then retrieves the $\mathsf{VAK.pk}$ of all the VMs registered by the tenant(s) requesting an attestation and concatenates, for each tenant, the nonce that tenant provided and the $\mathsf{VAK.pk}$ of all the VMs the latter owns. The list of $\mathsf{VAK.pk}$ per tenant constitutes its linking information (link in Fig. 5). If less than $N_{\mathcal{T}}$ request an attestation, empty position in the vector are filled with random values, and the commitment is always of constant size.

The vector commitment must be hiding, as tenants should learn no information about positions they will (later on) be unauthorized to open.

Our scaling approach provides scalability. Say that two or more tenants request hypervisor attestation while the TPM is busy. Without nonce-aggregation, those requests would be treated separately. Instead, aggregation allows the hypervisor to generate a single attestation that can be provided to all the tenants and *still* hide everything except the content pertinent to the tenant itself.

Hypervisor attestation. The next step is to obtain an attestation quote from the TPM. This communication is on the physical device (hidden from tenants). The hypervisor submits to the

[5] This idea appears in [3] but there the instantiation consists of simply including a set of public keys into the nonce. This achieves layer-linking but no inter-tenant privacy. In our approach, the instantiation requires vector commitments and ZK-SNARKs.

TPM the commitment c *in lieu of* an attestation nonce. The TPM computes a quote quote and a signature σ on it with the private attestation key $\mathsf{AK.sk}$ associated to the hypervisor.

Proof of attestation. The hypervisor, having received the quote and signature computes a proof of ownership of a valid attestation. Note that the attestation quote (and corresponding signature) reveal the configuration of the hypervisor, which we want to hide from the tenants. Thus, the hypervisor proves that it has a valid attestation from the TPM for a configuration within the set $\mathcal{CONF}$, with respect to nonce c, *i.e.*, it needs to compute:
$\mathsf{ZK\text{-}SNARK}\{(\mathsf{quote},\sigma)\colon \mathtt{SigVer}(\mathsf{AK.pk},\mathsf{quote},\sigma,c) == 1 \wedge \mathsf{quote}.\mathcal{H}.\mathsf{Conf} \in \mathcal{CONF}\}$

We can compile this computation into an arithmetic circuit. Then, a ZK-SNARK will allow the hypervisor to prove it has run this algorithm for some public set $\mathcal{CONF}$, the nonce c, with respect to $\mathsf{AK.pk}$, and that the algorithm output 1, all this without revealing the quote quote nor the signature σ.

Algorithm 1 The snark circuit

```
procedure SnarkCirc($\mathsf{AK.pk}, \mathsf{quote}, \sigma, c, \mathcal{CONF}$)
    if SigVer($\mathsf{AK.pk}, \mathsf{quote}, \sigma, c$) == 1 and $\mathsf{quote}.\mathcal{H}.\mathsf{Conf} \in \mathcal{CONF}$ then
        return 1
    else
        return 0
    end if
end procedure
```

Opening. Finally, the hypervisor needs to provide to each tenant its partial vector-commitment opening (*i.e.*, each tenant can only open the position corresponding to the index the hypervisor associated with that tenant at the beginning of its attestation). The hypervisor sets, for each tenant $\mathsf{ATT}_{\mathcal{H},\mathcal{T}}$ to contain: the proof of attestation π_{ZK}, the vector commitment c; the position i on which the tenant is placed; and the opening information π_t for that position.

4.4 VM Attestation

This algorithm $\{\mathsf{ATT}_{VM}\} \leftarrow \mathtt{VMAttest}(VM(\mathsf{VAK.sk}, \mathsf{VAK.pk}), \mathcal{T}(\mathcal{T}.\mathsf{sk}, \mathsf{nonce}))$ (fig. 6) generates a quote that only the tenant owning the VM can actually see and verify. The tenant and VM run an AKE protocol to establish a secure channel, over which $\mathcal{T}$ requests an attestation and forwards a nonce. The VM retrieves the linking information ($\mathsf{VAK.pk}$) and concatenates it with the nonce to obtain a value later hashed to lkaux. Then, the VM requests a signed quote for lkaux and forwards the response to the tenant over the secure channel.

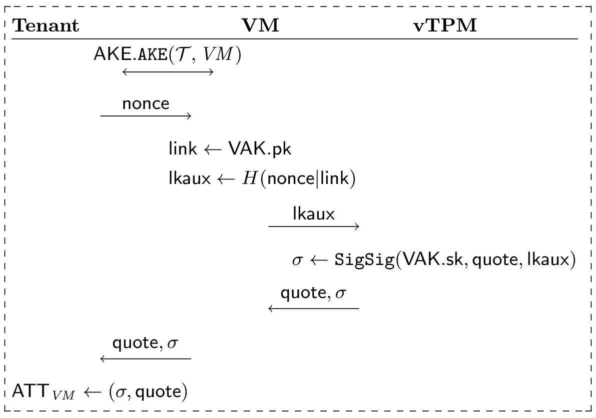


Fig. 6. VM Attestation

4.5 Verification

Hypervisor attestation verification. We instantiate the algorithm $\{0,1\} \leftarrow \texttt{VfHAttest}(\mathsf{ATT}_{\mathcal{H},\mathcal{T}}, \mathsf{nonce}_{\mathcal{T}}, \mathsf{link}_{\mathcal{T}})$ as depicted in Figure 4.5. The tenant opens the commitment c at the relevant index,then checks that it opens to the concatenation of the nonce $\mathsf{nonce}_{\mathcal{T}}$ and linking information $\mathsf{link}_{\mathcal{T}}$ (if this fails, the algorithm outputs 0). Then the tenant verifies the ZK-SNARK proof and outputs 1 if both verifications succeed.

```
$\texttt{VfHAttest}(\mathsf{ATT}_{\mathcal{H},\mathcal{T}}, \mathsf{nonce}_{\mathcal{T}}, \mathsf{link}_{\mathcal{T}}) \rightarrow \{0,1\}$ :
Parse $\mathsf{ATT}_{\mathcal{H},\mathcal{T}}$ as $(\pi_{ZK}, \pi_{i_{\mathcal{T}}}, c, i_{\mathcal{T}})$
$\mathsf{VC}.\texttt{Ver}((\mathsf{nonce}_{\mathcal{T}} | \mathsf{link}_{\mathcal{T}}), c, i_{\mathcal{T}}, \pi_{i_{\mathcal{T}}})$
$\mathsf{ZKP}.\texttt{SkVer}(CRS, R, x_{ZK}, \pi_{ZK})$
If all verification pass output 1, otherwise 0
```

Fig. 7. Hypervisor Attestation Verification

Verification of VM attestation. In this case, the verification is straightforward, as the tenant only retrieves the lkaux value and checks that the signature and received quote verify.

```
$\texttt{VfVMAttest}(\mathsf{ATT}_{VM}, \mathsf{nonce}, \mathsf{link}) \rightarrow \{0,1\}$ :
Parse $\mathsf{ATT}_{VM}$ as $(\mathsf{quote}, \sigma)$
$\mathsf{lkaux} \leftarrow H(\mathsf{nonce} | \mathsf{link})$
$\texttt{SigVer}(\mathsf{VAK.pk}, \mathsf{quote}, \sigma, \mathsf{lkaux})$
If all verifications work output 1, otherwise 0
```

Fig. 8. VM Attestation Verification

4.6 Linking attestations

The link algorithm $\{0,1\} \leftarrow \texttt{Link}(\mathsf{ATT}_{\mathcal{H},\mathcal{T}}, \mathsf{nonce}_{\mathcal{T}}, \mathsf{link}_{\mathcal{T}}, \mathsf{ATT}_{VM}, \mathsf{nonce}, \mathsf{link})$ will attempt to link the hypervisor and the VM attestation given in input. Any party in possession of the input values can run the linking – however, note that attestation quotes are only received over mutually-authenticated secure channels.

The party verifying the linking first verifies the two attestations – if both come through, then the verifier checks that the linking information for the VM is included in the linking information for the hypervisor.

```
$\texttt{Link}(\mathsf{ATT}_{\mathcal{H},\mathcal{T}}, \mathsf{nonce}_{\mathcal{T}}, \mathsf{link}_{\mathcal{T}}, \mathsf{ATT}_{VM}, \mathsf{nonce}, \mathsf{link}) \rightarrow \{0,1\}$:
$\texttt{VfHAttest}(\mathsf{ATT}_{\mathcal{H},\mathcal{T}}, \mathsf{nonce}_{\mathcal{T}}, \mathsf{link}_{\mathcal{T}})$
$\texttt{VfVMAttest}(\mathsf{ATT}_{VM}, \mathsf{nonce}, \mathsf{link})$
Check $\mathsf{link} \in \mathsf{link}_{\mathcal{T}}$
If all verifications work output 1, otherwise 0
```

5 Security Analysis

In this section we show that our construction guarantees inter-tenant privacy, configuration-hiding, and layer-linking. To prove that our scheme provides inter-tenant privacy we first introduce a special property of AKE schemes, namely Parter-hiding, and shows that TLS 1.3 has such a characteristic in section 5.1. In section 5.4 we also define a new property for vector commitment in order to prove the linking security of our construction.

5.1 Partner-hiding AKE

Our solution requires secure channels, constructed from AKE schemes. Indeed, consider the VM attestation algorithm from Section 4. The tenant and VM use a mutually-authenticated AKE to establish a secure channel over which attestation data is sent. This is sufficient to ensure that attestation quotes remain confidential for an adversary that controls neither the tenant nor the TPM.

However, channel-security is insufficient for inter-tenant privacy, where an adversary (possibly a collusion of tenants) must be unable to know if another tenant's VMs exist, or not, on the same machine as the adversary's. With regular AKE, this cannot be guaranteed even with mutual authentication. We require a stronger assumption, which we dub *Partner-Hiding*, in which an adversary not in possession of the long-term credentials of either endpoint cannot learn whether it faces a real or simulated entity as one endpoint. This property is not trivial to guarantee: some cipher suites of TLS 1.2 are not partner-hiding. TLS 1.3, however, does provide initiator-hiding properties, which we will put to use in our multi-tenant attestation protocol.

Security game. We consider two-party AKE protocols, for which the endpoints are parties $\mathcal{P} \in \mathbb{P}$. The protocol runs in *sessions* between an *instance* of one endpoint and an instance of the other. We denote i-th instance of party $\mathcal{P}$ as $\pi_{\mathcal{P}}^{i}$. Each $\mathcal{P}$ is associated with a tuple of long-term parameters $(\mathsf{sk}, \mathsf{pk})$ and each instance keeps track of the following *attributes*:

- $\pi_{\mathcal{P}}^{i}.\mathsf{sid}$: the session identifier of instance $\pi_{\mathcal{P}}^{i}$ is a concatenation of session-specific values, which might be public (included in public information, such as the transcript) or secret. The session identifier is protocol-specific.
- $\pi_{\mathcal{P}}^{i}.\mathsf{pid}$: the partner identifier of instance $\pi_{\mathcal{P}}^{i}$, which must be a party $\mathcal{Q} \in \mathbb{P} \setminus \mathcal{P}$.
- $\pi_{\mathcal{P}}^{i}.\alpha$: the acceptance flag α takes three values: $\perp$ (which stands for unset), 0 (reject), and 1 (accept). It models the result of the authentication performed by $\pi_{\mathcal{P}}^{i}.\mathsf{pid}$.
- $\pi_{\mathcal{P}}^{i}.\mathsf{k}$: the session key of instance $\pi_{\mathcal{P}}^{i}$, which starts out as equal to a special symbol $\perp$, but may take a true value once that key has been computed.

The AKE protocol is run between an *initiator* (*i.e.*, the party instance that starts the protocol) and the *responder* (*i.e.*, the party instance that goes second).

We define Partner-Hiding in terms of an adversary $\mathcal{A}$ that is a Person-in-the-Middle. The security game will, in a nutshell, guarantee that an adversary is unable to tell the difference between an interaction with a real, uncorrupted party, and an interaction with a simulator (which only has access to the security parameter, but not to any of the private keys generated in the game). Whereas this weak form of partner-hiding suffices for our needs, we also provide in the appendix a stronger notion, in which arbitrary corruptions are possible.

In our weak partner-hiding game, the adversary can control honest parties and instances by means of oracles:

- $\pi_{\mathcal{P}}^{i} \leftarrow \mathsf{oNewSession}(\mathcal{P}, \mathcal{Q}, \mathsf{role})$: the (honest) session creation oracle will initiate a new instance $\pi_{\mathcal{P}}^{i}$ with partner identifier $\pi_{\mathcal{P}}^{i}.\mathsf{pid} = \mathcal{Q}$, such that $\pi_{\mathcal{P}}^{i}$ plays the role designated by role (either initiator or responder) in its session.
- $m^* \leftarrow \mathsf{oSend}(\pi_{\mathcal{P}}^{i}, m)$: the (honest) sending oracle models sending message m to an already-existent instance $\pi_{\mathcal{P}}^{i}$. It is expected that $\pi_{\mathcal{P}}^{i}$ returns a message m^*, which is the protocol-specific reply (potentially an error symbol $\perp$) as a response. A special $m = \mathtt{Start}$ sent to a instance of an initiator is used to jump-start the session (thus yielding m^* as the first message of the actual session).
- $k \leftarrow \mathsf{oReveal}(\pi_{\mathcal{P}}^{i})$: the revelation oracle allows the adversary to learn already-established session keys k.
- $\pi_{\mathcal{P}}^{i} \leftarrow \mathsf{oNewSession}_{b,\mathsf{role}}(\mathcal{P}, \mathcal{Q})$: this is the left-or-right version of the $\mathsf{oNewSession}$ oracle above, for which the roles will be restricted according to which notion we want to guarantee between initiator- and responder-hiding. If $b = 0$, this oracle creates an instance of the party $\mathcal{P}$, with partner identifier $\mathcal{Q}$, such that $\mathcal{P}$ will have a role as either the initiator or the responder of the session. On the i-th call to the oracle $\mathsf{oNewSession}_{b}(\mathcal{P}, *)$, the created instance will be indexed as $\pi_{\mathcal{P}}^{i}$. The oracle forwards the handle $\pi_{\mathcal{P}}^{i}$ to the adversary. If $b = 1$, the oracle call is forwarded to a simulator Sim, which is only given the security parameter, but no party information.
- $m^* \leftarrow \mathsf{oSend}_{b}(\pi_{\mathcal{P}}^{i}, m)$: this left-or-right version of the sending oracle allows the message m to be either forwarded to $\pi_{\mathcal{P}}^{i}$ (if $b = 0$) or to the simulator Sim otherwise. In both cases the adversary expects a message m^*. As before, a special message $m = \mathtt{Start}$ will jump-start the session.

The security game begins with the setup of all the honest parties $\mathcal{P} \in \mathbb{P}$. The adversary receives all the public keys, whereas the challenger keeps track of all the private keys. The simulator will be given no information at all, apart from the security parameter.

There are two phases to the game. In the learning phase, the adversary will use the honest session-creation and sending oracles, as well as the session-key revelation oracle, in order to observe honest sessions and interact with the honest parties.

In the second phase of the game, the adversary gains access only to the left-or-right instance-creation and sending oracles. We distinguish between the two following notions:

– **Initiator-hiding.** In this case, the $\mathsf{oNewSession}_{b,\mathsf{role}}$ oracle has role set to `Initiator`. Hence, in the challenge phase, the adversary will only be able to create new instances that are protocol initiators.
– **Responder-hiding.** Conversely, in this case $\mathsf{oNewSession}_{b,\mathsf{role}}$ oracle has role set to `Responder`. Hence, in the challenge phase, the adversary will only be able to create new instances that are protocol responders.

Finally, the adversary will be allowed a final learning phase, identical to the first one. When the adversary is ready to end the game, it will output a bit d, which will be its guess for the bit b used by the challenger during the challenge phase.

It should be noted that at the transition to each new phase, all ongoing sessions are aborted.

```
Game $G_{\mathsf{InitHide}}(\lambda, N_{\mathcal{P}})$
Game setup for all $\mathcal{P} \in \mathbb{P}$ with $|\mathbb{P}| = N_{\mathcal{P}}$
$b \xleftarrow{r} \{0,1\}$
$\mathsf{state} \leftarrow \mathcal{A}^{\mathsf{oNewSession}(\cdot,\cdot,\cdot),\mathsf{oSend}(\cdot,\cdot),\mathsf{oReveal}(\cdot)}(1^\lambda)$
$\mathsf{state} \leftarrow \mathcal{A}^{\mathsf{oNewSession}_{b,\mathsf{Initiator}}(\cdot,\cdot),\mathsf{oSend}_b(\cdot,\cdot)}(1^\lambda, \mathsf{state})$
$d \leftarrow \mathcal{A}^{\mathsf{oNewSession}(\cdot,\cdot,\cdot),\mathsf{oSend}(\cdot,\cdot),\mathsf{oReveal}(\cdot)}(1^\lambda)$
$\mathcal{A}$ wins iff.: $d = b$
```

Fig. 9. The initiator-hiding game.

Definition 4 (Initiator-Hiding security). *Consider an authenticated key-exchange protocol* AKE*. This protocol is* $(N_{\mathcal{P}}, q_{\mathsf{oNewSession}}, q_{\mathsf{oNewSession}_{b,\mathsf{Role}}}, \epsilon)$-initiator hiding *if for any PPT adversary* $\mathcal{A}$ *making at most* $q_{\mathsf{oNewSession}}$ *queries to the (learning)* $\mathsf{oNewSession}$ *oracle and at most* $q_{\mathsf{oNewSession}_{b,\mathsf{Role}}}$ *queries to the (challenge)* $\mathsf{oNewSession}_{b,\mathsf{role}}$ *oracle, if we denote* $\mathsf{Adv}^{\mathsf{IHide}}_{\mathsf{AKE}}(\mathcal{A}) := \left|\Pr[\mathcal{A} \text{ wins } G_{\mathsf{InitHide}}(\lambda, N_{\mathcal{P}})] - \frac{1}{2}\right|$*, then it holds that:* $\mathsf{Adv}^{\mathsf{IHide}}_{\mathsf{AKE}}(\mathcal{A}) \leq \epsilon$.

The value $\mathsf{Adv}^{\mathsf{IHide}}_{\mathsf{AKE}}(\mathcal{A})$ *is the* advantage *of adversary* $\mathcal{A}$*. If* ϵ *is asymptotically negligible in the security parameter, then we call the authenticated key-exchange protocol* initiator-hiding.

Lemma 1. *The full TLS 1.3 handshake with mutual authentication is initiator-hiding under the following assumptions: all parties use particular configurations (*e.g.*, groups) and extensions with equal probability, the protocol uses collision-resistant hash functions, and the signature scheme is Existentially Unforgeable against Chosen Message Attacks (EUF-CMA).*

Proof (proof sketch). We first note that previous work [2] proved a slightly-different (and fundamentally stronger) degree of privacy for the TLS 1.3 full handshake, but only for handshakes with unilateral authentication.

The simulator we consider is fairly simple. For each call of $\mathsf{oNewSession}_{b,\mathsf{Initiator}}$, the simulator presents the adversary with an instance handle, which we label $\pi^i_{\mathcal{P}}$ (even if the simulator himself does not actually know the instance is supposed to belong to $\mathcal{P}$). Subsequent calls of oSend_b will be made to those instances that have been previously created, and notably:

– For $\mathsf{oSend}_b(\cdot, m = \mathsf{Start})$ calls, the simulator generates input consistent with the Client Hello of any client (recall that all configurations and extensions are equally likely).

- When fed with an $\mathsf{oSend}_b(\cdot, m)$ call for the server's first message (Server Hello, etc.), the simulator follows protocol, aborting if the server's choice of element or extension are inconsistent with its own. If all goes well, the simulator computes the handshake secret and subsequent keys. There is no response for this message expected from the client, so the simulator also sends no reply $m = \emptyset$.
- When fed with an $\mathsf{oSend}_b(\cdot, m)$ call for the server's second message (encrypted Certificate Request, Certificate, CertificateVerify...), the adversary uses the computed handshake keys to authenticate and decrypt the contents (aborting if AEAD fails). Then, the simulator proceeds with the verification of the Certificate signatures, and also the $\mathsf{CertificateVerify}$ message. If any of the verifications fail, then the simulator aborts the session. As before, the server expects no message in response so the simulator also sends no response.
- For all other messages sent, the simulator uniformly sends no reply, and it aborts the session at the end of the server's final message in its suite of messages [6].

For our proof, we make the following game hops:

$\mathbb{G}_0$: the original initiator-privacy game.

$\mathbb{G}_1$: the original game, except that we eliminate collisions in the nonce and DH elements used by instances of honest initiators (in the learning and challenge phases). This happens except with probability $\binom{(q_{\mathsf{oNewSession}}+q_{\mathsf{oNewSession}_{b,\mathsf{Role}}})}{2}$.

$\mathbb{G}_2$: the same as $\mathbb{G}_1$, except that we abort if in any two of the sessions created, the hash over the Client and Server Hellos coincide. As per $\mathbb{G}_1$, at least one input is unique in each session, notably the client's input. As a result, the two games are identical except if the content signed in the $\mathsf{CertificateVerify}$ message to have no collisions. At this game hop, we lose the advantage of the hash function against collision-resistance.

$\mathbb{G}_3$: this game is identical to $\mathbb{G}_2$ except that the adversary wins outright if it can, in the challenge phase, produce a successful forgery of the server's $\mathsf{CertificateVerify}$ message (note that in the initiator-privacy game against TLS 1.3, the initiator is the client; hence, in the challenge phase, the adversary will always play the role of responder, since it cannot create any responder instances). For the reduction, at this step we have to first guess which responder the adversary will choose to impersonate (*i.e.*, which party), in order to inject the challenge key-pair from the EUF-CMA game into that party. This counts for a factor $\frac{1}{N_{\mathcal{P}}}$. We also note that the challenger and the reduction have the means of verifying successful forgeries, as they know all the public keys and are one of the endpoints of the conversation (and can thus compute handshake keys). The reduction essentially works as follows:
- The reduction generates keys for $N_{\mathcal{P}} - 1$ parties, but not for the one party that we denote $\mathcal{P}^*$ which it has guessed will be impersonated by the adversary.
- During the learning phases, the reduction will faithfully be able to simulate sessions because it owns the private keys of all but the target party (which is at most one of the endpoints of any created session).
- During the challenge phase, the reduction chooses a bit b and simulates the challenge phase perfectly, but in addition also verifies each $\mathsf{CertificateVerify}$ message sent to an instance of any party $\mathcal{P} \neq \mathcal{P}^*$ which was created by an $\mathsf{oNewSession}_{b,\mathsf{Initiator}}(\mathcal{P}, \mathcal{P}^*)$ query (*i.e.*, $\mathcal{P}^*$ is the expected partner identifier of that instance). As soon as a forgery appears, it will be used by the reduction to win its game.
- If no forgery appears and the adversary $\mathcal{A}$ ends its game, the reduction aborts.

We note that in this case, if the adversary makes no forgery, then there is no distinction between $\mathbb{G}_2$ and $\mathbb{G}_3$, while if the adversary produces a forgery, then $\mathbb{G}_3$ is distinct from $\mathbb{G}_2$ from the point of view of the adversary (but in that case, we violate the EUF-CMA assumption for the signature scheme).

$\mathbb{G}_4$ This game is identical to $\mathbb{G}_3$, except that the protocol is no longer TLS 1.3, but rather, the challenger aborts all initiator challenge sessions (*i.e.*, sessions run by instances created by $\mathsf{oNewSession}_{b,\mathsf{Initiator}}$ queries) by default after the server's first pack of messages (so when the client's Certificate information and Finished message is expected). As per our last game, we

[6] We note that this is a much more limited version of a simulator than we could potentially build. Indeed, our simulator could continue to handle messages such as the encrypted Server Finished message – but we choose not to, because this step is not necessary.

have removed the possibility that the adversary produces a valid signature in *any* of the challenge sessions – since those signatures are generated on unique content (as per $\mathbb{G}_2$) and thus no replays from the learning phase is possible. As a result, none of the sessions created during the challenge phase will proceed further than the verification (of the CertificateVerify message) by the client. We thus incur no loss of security at this game hop. Thus, at this point the two worlds ($b = 0$ and $b = 1$) are identical from the point of view of the adversary, since the simulator follows the TLS 1.3 protocol to the letter up to, and including the server's CertificateVerify message. The adversary's winning probability is $\frac{1}{2}$.

5.2 Inter-tenant privacy

We examine the inter-tenant privacy provided by our PP-MTA protocol. We give first the intuition why our scheme guarantees this property, and then formalize the statement into a theorem.

Intuition. In the inter-tenant privacy game, the adversary, which represents one, or potentially a collusion of malicious tenants, aims to distinguish whether the target machine, which the adversary's own VMs are located on, also contains VMs belonging to other tenants or not.

One way the adversary could win is by creating first a VM of its own, and then attempting to create as many VMs as possible on behalf of another, honest tenant, until the machine is overwhelmed. We prevent this by enforcing a bound on the maximum number $N_{\mathcal{T}}$ of tenants, and on the maximum number N_{VM} of VMs per tenant for each machine. We ensure that the physical machine can host at least $N_{\mathcal{T}} \cdot N_{VM}$ VMs.

While the adversary learns no information from requiring attestations from its own VMs, it could potentially win by attempting to make a VM supposedly belonging to another tenant provide an attestation. If the VM is truly hosted on the target device, the device is aware of its existence, its public key, and its relationship with the tenant. If the VM is not hosted on the device, then the latter has no record of that VM's supposed public key. As a result, the mere guarantee of authentication in the AKE protocol does not suffice, and we need the partner-hiding property. The latter informally guarantees that the adversary (who does not know the honest tenant's private key) can never get far enough into the protocol in order to identify whether that particular VM exists, or not, on the machine.

The final source of potential information for the attacker is the hypervisor attestation process. All tenants, honest or malicious, have the right to demand a hypervisor attestation, which will include linking information to all the VMs present on the device (including the honest tenant's). There are three counter-mechanisms we employ against such attacks:

- At setup, the hypervisor sets the (fixed) size of the vector commitment to be $N_{\mathcal{T}}$. Then when computing a hypervisor attestation, it randomly associates tenant with indices between 1 to $N_{\mathcal{T}}$. That is to say, regardless of the order in which the adversary demands the registration of its own and other tenants' VMs, the adversary will have equal probability to be associated with any given index.
- The linking information to the VMs (their public keys) is included (in a hidden form) in a vector commitment. Opening information is provided to each authenticated tenant, for the index it is associated with. In other words, if the adversary authenticates by using its own credentials, the most it will find will be opening information linking the attestation to its own VMs. Attempting to impersonate an honest tenant will not work, as the attestation is sent over a secure channel generated upon the execution of an AKE protocol (with mutual authentication).
- Finally, note that the security of the channel guarantees that even if the adversary uses its hypervisor attestation oracle on behalf of a different tenant, the transcript it receives only contains an encrypted attestation and linking information.

Formalization. We formalize the following security statement for our inter-tenant privacy scheme.

Theorem 1 (Inter-tenant privacy). *Let* PP-MTA *be a multi-tenant attestation scheme; this scheme provides inter-tenant privacy if: the AKE protocol used during VM attestation is initiator-hiding, the secure-channel establishment protocol used during hypervisor attestation is ACCE-secure (providing authentication and secure-channel properties), and if the vector commitment guarantees* *the hiding property. More formally, if there exists an adversary $\mathcal{A}$ that breaks the inter-tenant privacy of* PP-MTA *with advantage* $\mathsf{Adv}^{\mathsf{TPriv}}_{\mathsf{PP\text{-}MTA}}(\mathcal{A})$*, then there exist adversaries* $\mathcal{B}_1, \ldots, \mathcal{B}_4$ *such that:*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{TPriv}}_{\mathsf{PP\text{-}MTA}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{IHide}}_{\mathsf{AKE}}(\mathcal{B}_1) &+ N_{\mathcal{T}} \cdot \mathsf{Adv}^{\mathsf{Auth}}_{\mathsf{AKE}}(\mathcal{B}_2)\\ &+ q_{\mathsf{oHAttest}} \cdot \mathsf{Adv}^{\mathsf{SC}}_{\mathsf{AKE}}(\mathcal{B}_3)\\ &+ q_{\mathsf{HAttest}^*} \cdot \mathsf{Adv}^{\mathsf{VCHide}}_{\mathsf{VC}}(\mathcal{B}_4),\end{aligned}$$

where q_{oHAttest} *represents the number of queries the adversary makes to the* oHAttest *oracle, and* q_{HAttest^*} *is the number of honest hypervisor-attestation sessions started by the adversary (on its own behalf) in its* PP-MTA *game.*

Proof (Proof sketch). The proof will proceed in the following game hops:

$\mathbb{G}_0$: The original game.

$\mathbb{G}_1$: Identical to $\mathbb{G}_0$, but we modify the authenticated key-exchange protocol such that, whenever the tenant requests the attestation of a VM that it does not own, the challenger simulates the protocol according to the simulator in the initiator-hiding game (no knowledge of the private or public keys is necessary). The adversary can distinguish between games $\mathbb{G}_0$ and $\mathbb{G}_1$ only with an advantage of at most $\mathsf{Adv}^{\mathsf{IHide}}_{\mathsf{AKE}}()$.

$\mathbb{G}_2$: Identical to $\mathbb{G}_1$, except that, whenever the adversary attempts a hypervisor attestation on behalf of an honest tenant (so by using the `HAttest` algorithm, rather than the oHAttest oracle), the attestation quote is replaced by an error symbol $\perp$. The adversary can only distinguish between the two if the adversary manages to impersonate an honest tenant. The reduction will first guess which tenant the attacker will target (losing a factor $N_{\mathcal{T}}$), hence giving us a loss equalling the second term of the bound above.

$\mathbb{G}_3$: Identical to $\mathbb{G}_2$, except that, at the first oHAttest query from the adversary, the challenger replaces the correct attestation quote and linking information by a message of the same length, but consisting only of 1s. We claim that the adversary only notices this if it can break the security of the channel over which the quote is sent.

$\mathbb{G}_4 \rightarrow \mathbb{G}_{2+q_{\mathsf{oHAttest}}}$ In each game $\mathbb{G}_{2+i}$, for $i \in \{2, \ldots, q_{\mathsf{oHAttest}}\}$, we proceed as in $\mathbb{G}_3$ for the i-th oHAttest query. At each time we lose a term $\mathsf{Adv}^{\mathsf{SC}}_{\mathsf{AKE}}(\mathcal{B}_3)$.

$\mathbb{G}_{3+q_{\mathsf{oHAttest}}}$: This game is identical to game $\mathbb{G}_{2+q_{\mathsf{oHAttest}}}$, except that, for the first direct hypervisor demand from the adversary (*i.e.*, uses of the `HAttest` algorithm rather than the oHAttest oracle), the challenger now replaces the input for each index of the vector commitment not corresponding to the adversary's position by a random value, ensuring that the resulting value in the commitment is different from the value it would have had had the challenger behaved normally (this restricts the adversary's choice of positions for which the opening is available in the commitment-hiding game). The adversary cannot distinguish between games $\mathbb{G}_{2+q_{\mathsf{oHAttest}}}$ and $\mathbb{G}_{3+q_{\mathsf{oHAttest}}}$ is exactly the advantage against commitment-hiding.

$\mathbb{G}_{4+q_{\mathsf{oHAttest}}} \rightarrow \mathbb{G}_{2+q_{\mathsf{oHAttest}}+q_{\mathsf{HAttest}^*}}$: Proceed to modify, in game $\mathbb{G}_{2+q_{\mathsf{oHAttest}}+i}$ the vector commitment for the i-th hypervisor attestation demand, for $i \in \{2, 3, \ldots q_{\mathsf{HAttest}^*}\}$ in the same way as the previous game. At each time the difference between each two successive games is the advantage against the vector commitment.

Analysis: At this point, the adversary has no better means than guessing, as the two worlds will be identical from its point of view, thus yielding the given bond.

Limitations to our guarantee. Our security model and proof holds against a broad class of attackers – but is not universally valid. For instance, if the separation (in terms of physical resources) between the tenant spaces and the VMs is not correctly set up, a tenant will naturally be aware of other VMs on the same machine. Moreover, multiple side-channel attacks are possible, exploiting, for instance, a longer response time than usual by the TPM (*i.e.*, the TPM was busy on another attestation at that time). Another avenue of attack would exploit the network, learning, for instance, the destination of a hypervisor attestation that is not the attacker's own. Such attacks are valid and deserve future investigations.

5.3 Hypervisor Configuration Privacy

We now delve into the hypervisor configuration-privacy property. We recall that in multi-tenant environments, an independent entity usually owns the physical machines hosting the VMs – and as a result, keeping the configuration of the meachine private from the tenants is a worthwhile goal.

Intuition. To begin with, note that the only moment when the configuration-privacy of the hypervisor is exposed is during the hypervisor attestation (which is generated by the physical TPM). The hypervisor receives a signed quote from the TPM (this communication takes part within the machine itself), then forwards a *proof* that it is in possession of a signed quote, which is consistent with a configuration $\mathcal{H}.\mathsf{Conf} \in \mathcal{CONF}$. In particular, the hypervisor computes (and later sends) $\mathsf{ZK\text{-}SNARK}\{(\mathsf{quote}, \sigma) : \mathtt{SigVer}(\mathsf{AK.sk}, \mathsf{quote}, \sigma, c) == 1 \ \wedge \mathsf{quote}.\mathcal{H}.\mathsf{Conf} \in \mathcal{CONF}\}$.

Our proof is straight-forward: by the zero-knowledge property of the ZK-SNARK, no information is revealed about quote and in particular about the configuration of the machine. In particular, the adversary will have no more than $\frac{1}{|\mathcal{CONF}|}$ probability to distinguish the actual configuration.

Formalization. We formalize the following security statement for our inter-tenant privacy scheme.

Theorem 2 (Configuration privacy). *Let* $\mathsf{PP\text{-}MTA}$ *be a multi-tenant attestation scheme; this scheme provides configuration privacy if: the ZK-SNARK is zero-knowledge, and the set* $\mathcal{CONF}$ *is large (size is exponential in the size of the security parameter). More formally, if there exists an adversary* $\mathcal{A}$ *that breaks the inter-tenant privacy of* $\mathsf{PP\text{-}MTA}$ *with advantage* $\mathsf{Adv}^{\mathsf{CPriv}}_{\mathsf{PP\text{-}MTA}}(\mathcal{A})$*, then there exists and adversary* $\mathcal{B}$ *such that:* $\mathsf{Adv}^{\mathsf{CPriv}}_{\mathsf{PP\text{-}MTA}}(\mathcal{A}) \leq q_{\mathsf{oHAttest}} \cdot \mathsf{Adv}^{\mathsf{ZK}}_{\mathsf{ZK\text{-}SNARK}}(\mathcal{A})$ *where* q_{oHAttest} *is the number of queries* $\mathcal{A}$ *makes to the hypervisor-attestation algorithm.*

Proof (Proof sketch.). We use a hybrid argument, replacing in each game hop the true attestation quote by a simulated attestation (using the simulator of the ZK-SNARK). This makes a total of q_{oHAttest} game hops, in which we lose at each time $\mathsf{Adv}^{\mathsf{ZK}}_{\mathsf{ZK\text{-}SNARK}}(\mathcal{B})$. At the end of this sequence of games, every true attestation has been replaced with a simulated one, which does not depend on $\mathcal{H}.\mathsf{Conf}$. As a result, the adversary has no better alternative than to guess the bit b input to the configuration-privacy game.

5.4 Collision Resistant Vector Commitment

The goal of a malicious hypervisor could be to find a collision by keeping the values of a given tenant with its correct index but modifying the other positions to find a collision with a previous committed vector. An other attack could also happened if the nonce of a given tenant would match with a previous nonce. In that case, the hypervisor could simply send the old attestation with the same proof of membership in the vector. Finally, the malicious hypervisor could also compute a collision which occurs with probability $\frac{q}{2^{n+1}}$ for q attestation queries and vector commitment of size n.

We thus define the collision resistance of vector commitment to avoid the above attacks. Note that this property has not been formalized in the context of vector commitment and should be also considered as of independent interest. We propose a security game, $G_{\mathsf{VC-Coll}}(\lambda, n)$ (see Figure 10), to define the collision resistance of vector commitment. This game is the same as for hash functions but for vector commitment: the challenger computes the setup algorithm to send the public parameters to $\mathcal{A}$ which outputs to (different) vectors. We say that $\mathcal{A}$ wins the game if and only if the commitments are equal.

Game $G_{\mathsf{VC-Coll}}(\lambda, n)$

$\{\mathsf{ppar}\} \leftarrow \mathsf{VC}.\mathtt{Setup}(1^\lambda, n)$

$(v, v') \leftarrow \mathcal{A}(\mathsf{ppar})$

$\mathcal{A}$ **wins** iff.: $\exists i$ such that $v[i] \neq v'[i]$ and $\mathsf{VC}.\mathtt{Com}(v) = \mathsf{VC}.\mathtt{Com}(v')$

Fig. 10. The collision resistance game for vector commitment.

Definition 5 (VC collision-resistance). *We say that* VC *is* $(\lambda, n)-$collision resistant *if for all adversary $\mathcal{A}$, the probability of winning game $G_{\mathsf{VC-Coll}}(\lambda, n)$ is negligible.*

Our construction uses Merkle trees, which are collision-resistant as stated by the following lemma:

Lemma 2. *A* VC *scheme, based on binary Merkle Tree, is collision resistant, assuming that the hash function H is collision resistant.*

Proof (sketch). The proof is done by reduction, we suppose that there exists $\mathcal{A}$ winning the collision resistance game for VC and show that we can construct $\mathcal{B}$, using $\mathcal{A}$ as a subroutine, winning the collision resistance of H.

If such $\mathcal{A}$ exists, then $\mathcal{B}$ could simply recompute the merkle tree of v and v' and then, starting from the root of each tree, search for collision (which is linear complexity) and return the output.

5.5 Linkability Security

Our PP-MTA protocol has a linking property, which is stated as the following theorem.

Theorem 3 (Linkability security). *Let* PP-MTA *be a multi-tenant attestation scheme; this scheme provides linkability security if: the hash function H and* VC *are collision resistant, the* ZK-SNARK *is sound, and the signature scheme* SIG=(`SigKGen`,`SigSig`,`SigVer`) *is EUF-CMA. More formally, if there exists an adversary $\mathcal{A}$ that breaks the linkability security of* PP-MTA *with advantage $\mathsf{Adv}^{\mathsf{Link}}_{\mathsf{PP\text{-}MTA}}(\mathcal{A})$, then there exist adversaries $\mathcal{B}_1, \ldots, \mathcal{B}_5$ such that:*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{Link}}_{\mathsf{PP\text{-}MTA}}(\mathcal{A}) \leq \frac{1}{N_{\mathcal{P}}} &+ \mathsf{Adv}^{\mathsf{Coll}}_{\mathsf{H}}(\mathcal{B}_1) + \mathsf{Adv}^{\mathsf{Coll}}_{\mathsf{VC}}(\mathcal{B}_2)\\ &+2 \cdot (N_{VM} \cdot \mathsf{Adv}^{\mathsf{VCBind}}_{\mathsf{VC}}(\mathcal{B}_3)\\ &+\mathsf{Adv}^{\mathsf{ZK}}_{\mathsf{ZK\text{-}SNARK}}(\mathcal{B}_4) + \mathsf{Adv}^{\mathsf{EUF-CMA}}_{\mathsf{Sig}}(\mathcal{B}_5))\end{aligned}$$

where q_{att} is the number of queries $\mathcal{A}$ makes to the oHAttest *and* oVMAttest *oracles, and n is the size of committed vectors.*

Intuitively, this theorem states that, in order to link two components that are not on the same platform, an adversary needs to break at least one assumption. Our model consider malicious tenants which have (legitimate) access to all the VMs of both platforms; yet, the only possibility for the adversary to provide attestations that will verify as being linked (*i.e.*, `Link` returns 1) is to forge an attestation from the hypervisor.

Proof (proof sketch). We give the game hops for our proof.

$\mathbb{G}_0$: Initial linkability game $G_{\mathsf{Link}}(\lambda, N_{\mathcal{P}})$.

$\mathbb{G}_1$: The challenger guesses parties $\mathcal{P}$ and $\mathcal{Q}$ outputted by $\mathcal{A}$. There are two platforms composed each of one hypervisor and N_{VM} VM. The probability or right guess is $\frac{1}{4N_{VM}}$.

$\mathbb{G}_2$: We rule out that $\mathsf{lkaux} = \mathsf{lkaux}'$ meaning that $H(\mathsf{nonce}\|\cdot) = H(\mathsf{nonce}'\|\cdot)$ for $\mathsf{nonce} \neq \mathsf{nonce}'$. This corresponds to the collision resistance of H.

$\mathbb{G}_3$: We ensure the uniqueness of committed vectors by removing collisions, this corresponds to a factor $\mathsf{Adv}^{\mathsf{Coll}}_{\mathsf{VC}}(\mathcal{B}_2)$.
At this point, the only way for the adversary to win the game is to forge an attestation for $\mathcal{P}$ or $\mathcal{Q}$. The next games refer to rule out the fact that $\mathcal{A}$ outputs $\mathsf{ATT}_{\mathcal{P}}$ (or $\mathsf{ATT}_{\mathcal{Q}}$) which its attestation corresponds to a value stored in $\mathcal{L}_{Att}$. The games $\mathbb{G}_4$, $\mathbb{G}_5$ and $\mathbb{G}_6$ ensure that $\mathcal{P}$'s attestation does not correspond to another one stored in $\mathcal{L}_{Att}$.

$\mathbb{G}_4$: The adversary can try to forge an opening thus making a commitment opens to a different message than the initial one. This corresponds to violating the position binding property, thus we loose a factor $N_{VM} \cdot \mathsf{Adv}^{\mathsf{VCBind}}_{\mathsf{VC}}(\mathcal{B}_3)$.

$\mathbb{G}_5$: This game ensures that the soundness property of the ZK-SNARK holds. If the proof of the committed vector verifies for other values (*e.g.*, adding a VM from another platform into link) then the adversary is able to forge a proof. This corresponds to $\mathsf{Adv}^{\mathsf{ZK}}_{\mathsf{ZK\text{-}SNARK}}(\mathcal{B}_4)$.

$\mathbb{G}_6$: We ensure that the adversary cannot forge a signature of the quote, this corresponds to lose a factor $\mathsf{Adv}^{\mathsf{EUF-CMA}}_{\mathsf{Sig}}(\mathcal{B}_5)$.

$\mathbb{G}_7$: We repeat games $\mathbb{G}_4$, $\mathbb{G}_5$ and $\mathbb{G}_6$ for party $\mathcal{Q}$. At this point the adversary cannot win the game.

6 Implementation

We provide a proof-of-concept implementation of the scheme described in Section 4 in Python, with some parts related to the ZK-SNARK written in Rust. We used this implementation to design benchmarks and evaluate the performance of the scheme. Specifically, we focused on the performance of the VM attestation and hypervisor attestation compared to a traditional attestation. We also demonstrate the scaling properties of our scheme experimentally.

6.1 Implementation and experiment details

In what follows, we describe some of the details of our implementation and tools we used. We also describe our experimental setup.

TPM libraries. To communicate with the (physical or virtual) TPM, we used the software provided by tpm2-software community [1], relying on TPM software stack (TSS) – an API specified by the TCG.

Vector Commitment. We implemented our vector commitment scheme using a binary Merkle Tree, using the pymerkletools library [28] combined with a basic hash-based commitment scheme.

SNARK. We used bellperson [13] which implements a preprocessing circuit-specific crs snark [15] for a rank-1 constraint system (R1CS) over a bls12-281 curve, as well as several gadgets for circuit design. Additionally we used bellperson-nonnative [26]. a library to compute arbitrary-precision arithmetic operations inside SNARKs.

SNARK Circuit design. The circuit computes an RSA PKCS#1 v1.5 signature verification and set membership verification. We implemented the RSA verification for a 2048-bit modulus and a fixed exponent of 0x10001. Moreover, the circuit is designed for a nonce of length 32. Our implementation also assumes that the TPM will use sha256 as its hash algorithm. Hence, the size of the quote is fixed and thus the number of constraints in the circuit only depends on the size of the configuration set. The CRS will only need to be recomputed for a different size set (if using a circuit-specific-CRS SNARK).

Limitations. Our current implementation is basic and contains no network communication, as the latter is not necessary for the basic feasibility performance measurements we aim for here. However, we also provide a demonstration script, which simulates a use-case scenario into a single process and gives an idea of the flow of the protocol in a real situation.

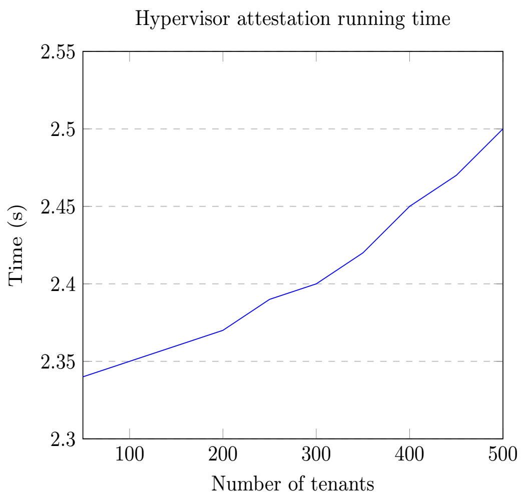
Setup. Our tests and benchmarks were carried out on a laptop running Ubuntu 20.04.5 with an Intel i7-10875H CPU (16 cores), 32GB RAM and a STMicroelectronics ST33TPHF2XSPI TPM. The VM attestation benchmarks were ran inside a KVM/QEMU 4.2.1 VM (running on the same laptop) with virtual TPM provided by swtpm 0.6.2 (libtpms 0.9).

Experimental method. The Rust part of the code was measured using the Criterion library [16] which run the function to benchmark 100 times after a warm-up phase, then compute statistics over these samples. For the Python code we implemented a custom decorator, which works in a similar fashion and provides statistics over these runs.

6.2 Benchmark results

We now present our results. First we compare a traditional attestation with our privacy preserving hypervisor attestation and we show how our scheme scales with the number of tenant. This experiment has been carried using a set of possible configuration of size 128. Hence we provide measurement of the SNARK computation for different set size to show of it affect the performances of our construction. Finally we provide some result about the linking procedure and shows that it has a negligible impact.

Attestation and scaling. Table 11b compares a traditional attestation and our hypervisor attestation. We can see that the ZK-SNARK adds a significant overhead but the total time is still low enough (2.4s) for practical use. This is especially true with a high number of tenants requests (like in 5G and beyond). With a classic sequential processing of the requests it would take minutes to answer all the tenants whereas in our case despite having a relatively high base computing time, the attestation scales very well as depicted on Figure 11a.



(a) Scaling (configuration set of size 128).

Attestation

	Mean	Median
Traditional (s)	0.94	0.94
Hypervisor (s)	2.40	2.40
SNARK (s)	1.46	1.46
Commitment (ms)	9.06	8.98

Verification

	Mean	Median
Traditional (ms)	2.42	2.36
Hypervisor (ms)	25.06	25.05
SNARK (ms)	25.02	24.99
Commitment (ms)	0.043	0.063

(b) Time to perform attestation

Fig. 11. Benchmarks

SNARK. The hypervisor-attestation benchmarks presented above are given for a set of 128 configurations. That set size directly impacts the number of constraints in our SNARK circuit. Table 1 shows how the setup, prover, and verifier algorithm performances change with set size.

Table 1. Performance variations of the ZK-SNARK in the configuration-set size. Median value over 100 samples.

Set Size	32	64	128	256	512
Number of constraints	213565	222301	239774	274719	344609
Setup (s)	18.07	18.75	20.04	26.59	31.86
Prover(s)	1.43	1.44	1.46	2.45	2.5
Verifier(ms)	14.93	18.71	24.41	41.70	72.17

Linking. Table 2 presents our measurements of the time required for the linking of a VM and a hypervisor attestation quote. Note that the measurements below do not correspond to the full algorithm presented in section 4.6 but only include the verification of the linking information. The linking information inside the hypervisor attestation depends on the number of VM owned by the tenant, which impacts the overall performance. Even in spite of such variations, our scheme provides very fast, easy linking.

Table 2. Time to perform linking depending on the number of VM hosted by the hypervisor.

Number of VM	50	100	150
Linking time (μs)	27.89	56.74	84.40

7 Conclusion and Discussion

In this work, we proposed a scalable and efficient TPM attestation scheme for multi-tenant environments. Our scheme does not require modification to the TPM nor unrealistic trust assumptions (e.g., Attestation proxy). It provides strong privacy for both tenants and the hypervisor, and guarantees layer-binding.

Our scheme achieves privacy by relying on vector commitment and ZK-SNARK. The latter primitive incurs a relatively high overhead but it remains stable even with a drastically high number of attestation requests (which is the case in multi-tenant environments like 5G) without requiring any TPM modification. In addition, if in some cases the configuration hiding property is not needed – but still have a high number of attestation requests –, due to the modularity of our construction, our scheme can still work efficiently by simply omitting the ZK-SNARK module.

Acknowledgements We thank the anonymous reviewers for their constructive comments. This study was partially supported by the French ANR, grant 18-CE39-0019 (MobiS5).

References

1. Linux tpm2 & tss2 software (2022)
2. Arfaoui, G., Bultel, X., Fouque, P., Nedelcu, A., Onete, C.: The privacy of the TLS 1.3 protocol. PoPETs (2019)
3. Arfaoui, G., Fouque, P., Jacques, T., Lafourcade, P., Nedelcu, A., Onete, C., Robert, L.: A cryptographic view of deep-attestation, or how to do provably-secure layer-linking. In: ACNS 2022 (2022)
4. Berger, S., Goldman, K., Pendarakis, D., Safford, D., Valdez, E., Zohar, M.: Scalable attestation: A step toward secure and trusted clouds. In: IC2E (2015)
5. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In: ITCS (2012)
6. Brickell, E., Camenisch, J., Chen, L.: Direct anonymous attestation. In: CCS '04 (2004)
7. Catalano, D., Fiore, D.: Vector commitments and their applications. In: PKC (2013)
8. Chen, L., Landfermann, R., Löhr, H., Rohe, M., Sadeghi, A.R., Stüble, C.: A protocol for property-based attestation. In: ACM STC (2006)
9. Chen, L., Löhr, H., Manulis, M., Sadeghi, A.R.: Property-based attestation without a trusted third party. In: ISC (2008)
10. Eckel, M., Fuchs, A., Repp, J., Springer, M.: Secure attestation of virtualized environments. In: Hölbl, M., Rannenberg, K., Welzer, T. (eds.) ICT (2020)
11. ETSI: GS NFV-SEC 007 v1.1.1. Tech. rep. (2017)
12. Fajiang, Y., Jing, C., Yang, X., Jiacheng, Z., Yangdi, Z.: An efficient anonymous remote attestation scheme for trusted computing based on improved cpk. Electronic Commerce Research (2019)
13. Filecoin: bellperson (2022)
14. Foy, K.: Keylime software is deployed to ibm cloud (2021)
15. Groth, J.: On the size of pairing-based non-interactive arguments. In: EUROCRYPT (2016)
16. Jorge, A., Heisler, B.: criterion (2022)
17. Keylime dev: Keylime (2022)
18. Larsen, B., Debes, H.B., Giannetsos, T.: Cloudvaults: Integrating trust extensions into system integrity verification for cloud-based environments. In: ESORICS (2020)
19. Lauer, H., Kuntze, N.: Hypervisor-based attestation of virtual environments. In: UIC-ATC (2016)
20. Poritz, J.A., Schunter, M., Herreweghen, E.V., Waidner, M.: Property attestation — scalable and privacy-friendly security assessment of peer computers. Tech. rep., IBM (2004)
21. Ruan, A., Martin, A.: Repcloud: Achieving fine-grained cloud tcb attestation with reputation systems. In: STC (2011)
22. Sadeghi, A.R., Stüble, C.: Property-based attestation for computing platforms: Caring about properties, not mechanisms. In: NSPW (2004)
23. Santos, N., Rodrigues, R., Gummadi, K.P., Saroiu, S.: Policy-Sealed data: A new abstraction for building trusted cloud services. In: USENIX Security (2012)
24. Schear, N., Cable, P.T., Moyer, T.M., Richard, B., Rudd, R.: Bootstrapping and maintaining trust in the cloud. In: ACSAC (2016)
25. Schiffman, J., Moyer, T., Vijayakumar, H., Jaeger, T., McDaniel, P.: Seeding clouds with trust anchors. In: CCSW (2010)
26. Setty, S.: bellperson-nonnative (2022)
27. TCG: Virtualized trusted platform architecture specification. Tech. rep. (2011)
28. Tierion: pymerkletools (2022)
29. Xin, S., Zhao, Y., Li, Y.: Property-based remote attestation oriented to cloud computing. In: CIS (2011)
30. Zhang, T., Lee, R.B.: Cloudmonatt: An architecture for security health monitoring and attestation of virtual machines in cloud computing. In: ISCA (2015)