Privacy-preserving Attestation for Virtualized Network Infrastructures

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

[1] Orange Innovation
[2] XLIM, University of Limoges

Abstract. In multi-tenant cloud environments, physical resources are shared between various parties (called tenants) through the use of virtual machines (VMs). Tenants can verify the state of their VMs by means of *deep-attestation*: a process by which a (physical or virtual) Trusted Platform Module –TPM – generates attestation quotes about the integrity state of the VMs. Unfortunately, most existing deep-attestation solutions are either: limited to single-tenant environments, in which tenant *privacy* is irrelevant; are inefficient in terms of *linking* VM attestations to hypervisor attestations; or provide privacy and/or linking, but at the cost of modifying the TPM hardware.

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 unaware of whether or not the physical machine hosting its VMs also contains other VMs (belonging to other tenants); **(ii) Configuration privacy**: the hypervisor's configuration, used in the attestation process, remains private with respect to the tenants requiring a hypervisor attestation; and **(iii) Layer linking**: our protocol enables tenants to link hypervisors with the VMs, thus obtaining a guarantee that their VMs are running on specific physical machines.

Our solution relies on vector commitments and ZK-SNARKs. We build on the security model of Arfaoui *et al.* and provide both formalizations of the properties we require and proofs that our scheme does, in fact attain them. Our protocol is scalable, and our implementation results prove that it is viable, even for a large number of VMs hosted on a single platform.

Keywords: Deep Attestation · NFV · Multi-tenant

1 Introduction

The use of *virtualization* has fundamentally revolutionized infrastructures, in terms of both dynamics and flexibility. This is perhaps most visible in the context of mobile networks, where *Network Function Virtualization (NFV)* allows operators to use continuously-evolving networks, in which Virtual Network Functions (VNFs) can easily be added, removed, or migrated. Such VNFs (which are virtual machines – VMs – managed by a hypervisor) implement services like routing and authentication, allowing servers to scale up, scale out, or deploy new services on demand. A similar use-case appears in cloud-computing environments.

The attractive flexibility of virtualization, however, induces an inevitable loss of control and trust in the resulting infrastructure. A solution recommended by the European Telecommunications Standards Institute (ETSI) is the use of *remote attestation* [15] for the verification of the components of a virtualized infrastructure. Remote attestation enables a prover, typically a component on a virtual platform (*e.g.*, a VM) to prove to an authorized verifier that it conforms to some specifications (and thus that it has specific properties). The verifier will then trust this component and later extend its service. In this paper, we specifically focus on the type of attestation that verifies the integrity state of a component.

Attesting a virtual component usually implies attesting the underlying virtualization infrastructure: a process called *deep attestation*. To implement deep attestation, ETSI first proposed single-channel attestation [16, 17], which verifies, simultaneously, the integrity of the VM, that of its associated hypervisor, and their *layer binding*, *i.e.*, the fact that the VM is managed by the designated hypervisor. While sound from the point of view of security, this approach scales badly for a large number of VMs. Hence, ETSI's second proposal, multiple-channel attestation [16, 17] sacrifices the layer-binding property to provide more efficient attestation.

At ACNS 2022, Arfaoui *et al.* [2] proposed a solution which achieves the best of both worlds, achieving layer-binding with practically the same efficiency as multiple-channel attestation. Unfortunately, like other solutions before it [13, 2, 25], Arfaoui *et al.*'s protocol is unsuited to multi-tenant environments [14], in which virtual components situated on various platforms may belong to different entities called *tenants*, with possibly-different (and conflicting) security goals.

The aim of this work is to address the topic of deep-attestation in multi-tenant environments. We specifically consider a use-case similar to that of Keylime [23, 32], a solution employed in use-cases such as cloud infrastructures, which provides system-integrity monitoring based on TPM attestations. Keylime allows for the deployment of an agent on each target VM; the agent sends, every few seconds, an attestation to a cloud verifier. Keylime is efficient and works well even in very large cloud-infrastructures [20], but is unfortunately limited to VM attestations (rather than to linked hypervisor/VM attestation) and moreover provides no tenant- or 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, and provides strong privacy for both the tenants and for the provider of the physical infrastructure.

1.1 Our contribution

We consider a typical multi-tenant architecture as shown in Figure 1. The hypervisor is equipped with a hardware Trusted Platform Module (TPM) [37] and spawns a virtual TPM (vTPM) [3] 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 check the attestation (and layer-binding) of its VMs and the hypervisor.

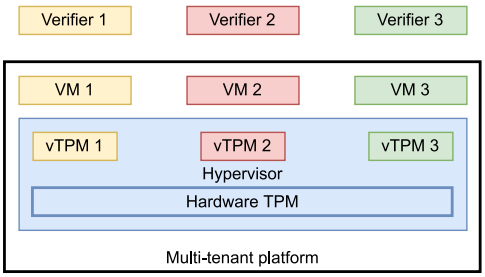

Fig. 1. A multi-tenant environment with three VMs, each belonging to a different tenant. Each tenant has its own verifier for attestation.

Our work makes a triple contribution:

A new protocol. We propose a primitive called privacy-preserving multi-channel attestation (PP-MTA), which has the attractive security of single-channel attestation, but the scalability of multiple-channel attestation *in a multi-tenant environment*. As described in Section 1.3, we instantiate PP-MTA with a protocol that guarantees attestation, layer-binding, and in addition:

Inter-tenant privacy: No tenant can learn whether other tenants share the same resources that host its VMs using our scheme.

Configuration privacy: The hypervisor attestation basically convinces a verifier that the hypervisor has a valid *configuration*. However, since the hypervisor might not belong to the tenant, we allow the hypervisor to keep that configuration *private*.

Moreover, our scheme achieves these strong properties –for the first time in the literature– without requiring modifications to generic TPMs. As described in the technical overview, our construction relies on vector-commitment schemes and ZK-SNARKs [5].

Formal analysis. We formally model and prove the security and privacy of our protocol. We extend the layer-binding properties presented in [2] to a multi-tenant environment, then define, *for the first time in the literature*, the properties of *inter-tenant privacy* and *hypervisor configuration-privacy*.

We then *prove* that our PP-MTA scheme guarantees these properties. Our proofs rely on the security of the ZK-SNARKs, vector-commitment scheme, and (minimally) on an ACCE-secure secure-channel establishment protocol. Interestingly (and contradicting naïve intuition), our proofs require stronger-than-standard binding properties for the vector commitment, and also a stronger privacy property for the secure-channel establishment property. Thus, we formally define: (i) *collision-resistance* as a new property of vector-commitment schemes; and (ii) *partner-hiding* as a property of authenticated key-exchange schemes. We also prove that the vector commitment we employ in our PP-MTA scheme (relying on Merkle trees) is collision-resistant, while the TLS 1.3 handshake that we use guarantees a flavour of partner-hiding, namely initiator-hiding.

Both properties, and their impact in providing privacy, are of independent interest.

Implementation. We show the feasibility of our solution by providing a proof-of-concept implementation of our protocol, for which we describe several benchmarks. Despite relying on SNARKs, known to have poor performance, our scheme remains fast enough for real use application, with the hypervisor attestation requiring 2.5s and the attestation-verification, 25ms.

1.2 Background and Related work

Apart from prior work already mentioned on deep attestation, our work builds on a vast literature of TPM-based remote-attestation, for which we recall some basics below. We then review existing work on property-based attestation, and attestation in cloud-based environments. Finally, at the end of this section, we single out two papers which we discuss in more detail: the layer-linking single-tenant protocol of Arfaoui *et al.* [2] and the privacy-preserving AKE definition in the work of Schäge *et al.* [31].

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.

Property-based attestation. Classical TPM attestations follow a binary approach that reveals the configuration of the target (notably, a function of the PCR values), which is both a privacy threat and impractical, as the verifier must know all possible trusted configurations to validate the attestation. *Property-based Attestation (PBA)* [29] overcomes some limitations by verifying that the target satisfies some high-level properties instead of binary measurements. A specific property can be achieved by different configurations. We can distinguish two main families of PBAs: (1) PBAs like [27, 9], which use a Trusted Third Party (TTP), whose existence cannot always be guaranteed; (2) PBAs like [10, 18], for which the hypervisor uses a zero-knowledge proof of membership, confirming that its configuration belongs to a set of valid configurations – at the expense of modifying TPM specifications.

TPM attestation in the cloud. Multi-tenant attestation is particularly useful in cloud-like environments, where multiple tenants share physical resources. Several techniques allow a tenant to check and monitor their cloud nodes, [23, 32] or the trusted state of the virtualization platform, while preventing the full disclosure of the platform configuration[40]. In the Keylime scenario presented above, we face a multi-tenant privacy-sensitive scenario, in which potentially thousands of attestation quotes must be computed by the resource-constrained TPM. Whereas Keylime provides no privacy and no layer-linking, and PBA offers privacy, but at the expense of TPM modifications or a TTP, we present a solution for provable privacy, layer-linking, which is effcient and scalable.

Other solutions, like Scalable Attestation [4] enables to assess the trust level of either guest or host environments by combining trusted and secure boot, but without linking.

When evaluating the trust level of a guest environment, a common approach is to centralize attestation in the cloud through a cloud verifier – either providing attestation of both host and guest [33] or combining this approach with PBA [42]. Policy-Sealed Data [30] is an alternative to our approach, but which requires a *trusted* cloud verifier.

The same centralized attestation pattern exists when multiple nodes must attest one another. Examples include privacy-preserving attestation [28], or attestation frameworks for container-based service function chains [24, 12].

While similar to ours, such approaches either require TPM modifications, do not apply to the generic use-case depicted in Figure 1, require unrealistic trust assumptions (like a trusted verifier), or do not provide the strong privacy we desire.

Comparison to [2, 31]. Our work comes closest to [2, 31]. We provide a comparison to these recent publication below.

Comparison to [2]. Arfaoui *et al.*'s recent publication allows to understand, model, and prove the security of single-tenant, linkable deep-attestation. In their scenario, an authorized verifier can request the attestation of VMs and the hypervisor supposedly hosted on the same platform. The protocol proposed in [2] provides linkable attesstation without modifications to the standard TPM, by ensuring that hypervisor attestations also incidentally attest the public keys corresponding to the VMs hosted on the same machine. The protocol provides the following properties (cf. [2]:

- **Authentication**: No quotes are issued to any entity excepting the verifier;
- **Authorization**: Quote contents are confidential except to an authorized verifier (which, however, will learn a function of the hypervisor's configuration);
- **Linking**: A hypervisor quote and a VM quote are only linkable if the two are hosted on the same physical machine.

Unfortunately, this solution is not practical in multi-tenant environments for which, additionally, no tenant is also the platform owner (the use case of Keylime[23]). The protocol designed by Arfaoui *et al.* embeds in the hypervisor attestation the public keys of *all the VMs* hosted on the platform, which provides linkability, but not privacy – for either the tenants (as they learn attestation keys belonging to other tenants' VMs), nor for the platform owner (the hypervisor's configuration is leaked).

While our solution reuses parts of the layer-linking of [2], our main focus is on privacy – and as a consequence, our protocol uses vector commitments and ZK-SNARKs in order to provide scalable, privacy-preserving, efficient, linkable attestation

Comparison to [31]. Schäge *et al.* considered privacy-preservation in authenticated key-exchange (AKE). While not related to attestation, their notion of PPAKE is similar to the property of Partner-Hiding AKE defined here – which we need to prove inter-tenant privacy.

There is, however, a subtle difference between partner-hiding AKE and the left-or-right privacy indistinguisability described by [31]. The guarantee defined by [31] ensures that a Person-in-the-Middle, situated between the initiator and the responder in an AKE protocol, cannot link sessions featuring say the same initiator (out of a set of possible initiators). By contrast, we need a deniability-like notion: a legitimate responder must be unable to tell whether it is interacting with a real or a simulated initiator. Our ultimate goal is to guarantee that no collusion of malicious tenants can tell whether other VMs (corresponding to other, honest tenants) are hosted on a specific platform as their VMs.

1.3 Technical Overview

In our scenario, various tenants may own VMs, hosted on a platform owned by a platform-owner. Each tenant knows identifying data on its own VMs and may request either a VM or a hypervisor attestation. We want the following properties:

- **Inter-tenant privacy:** A collusion of tenants cannot tell whether that platform contains other VMs than their own, or not;
- **Hypervisor-configuration privacy:** A collusion of tenants or other parties cannot learn the configuration of the platform (out of a set of configurations);
- **Linking:** Hypervisor and VM attestations are only linkable if the VM is hosted on the machine managed by the hypervisor.

We reuse the layer-linking strategy of the protocol of [2]: We embed auxiliary information in the hypervisor's attestation quote, making it linkable to VM quotes. However, we want to simultaneously guarantee strong *privacy*, both for the tenants and the platform-owner. We also want our solution to scale well, even in cloud environments such as Keylime, for which many attestation requests are made simultaneously by various tenants.

Solution outline. A naïve approach is to create a single-channel deep-attestation quote every time a tenant requests the attestation of their VMs. This approach provides two of the three properties above: inter-tenant privacy and layer-linking. However, two problems remain: hypervisor-configuration and also scalability.

Hypervisor-configuration is highly desirable when tenants and their VMs are hosted on a platform whose owner wants to retain the privacy of each machine's configuration even with respect to tenants whose VMs run on those machines. If, during attestation, (a function of) the attested PCR values leaks to a tenant, this reveals part of the hypervisor's configuration. In order to guarantee that property, the attestation quotes need (simultaneously) to: (i) be signed by a physical root of trust (TPM); (ii) only provide a proof that the measurement on the PCR values has a "correct" value, rather than directly giving away that value. In our solution, we provide hypervisor-configuration privacy by including a zero-knowledge proof of knowledge, called a SNARK. Whereas we could make the TPM compute the ZKP itself rather than relying on SNARKs, this would require TPM modifications, which we want to avoid.

The second problem with our naïve solution is scalability. Single-channel deep-attestation requires a (slow) physical TPM to generate linkable attestations. In a use-case like that of Keylime, hundreds of clients may request attestations every second, so the use of single-channel attestation is impractical. We in fact include a way of *batching* hypervisor attestations for multiple tenants together.

Unfortunately, batching is difficult when it must guarantee both inter-tenant privacy and layer-linking. Since layer-linking requires the hypervisor attestation to be linkable to independent VM attestation: thus, the hypervisor has to leak some information regarding which VMs it is managing. For batched hypervisor attestations, we will want a single attestation to allow one tenant, say Tenant A, to link the hypervisor with all of its VMs on that machine – but simultaneously, that attestation must allow a different tenant (Tenant B) to do the same with its own VMs. This is challenging when we want Tenant A to know nothing about Tenant B's VMs – or even the very existence of Tenant B.

To bridge the gap between layer-linking and inter-tenant privacy, we leverage vector-commitment schemes, which will store (in a hidden form) linking information to all the VMs hosted on the hypervisor. The vector is artificially padded so that, irrespective of the number of VMs hosted, each tenant will be faced with the same commitment size. Each tenant will only open the positions of the vector commitment corresponding to its own VMs, learning nothing about any other VMs potentially stored on them.

System model. In this work we will distinguish between attestation and privacy adversaries. For attestation, we consider the generally adopted generic system model of [34] for hardware-assisted computing platforms. This model distinguishes several classes of adversaries of increasing strengths, including: application-level (*e.g.*, unprivileged) attackers; co-residents (*e.g.*, VM-to-VM threats); system-level adversaries (*e.g.*, untrusted OS/hypervisor); network-level threats (*e.g.*, , untrusted I/O peripherals); and the invasive attacker with access to the physical infrastructure, an all-powerful threat to confidentiality and integrity. In this work the most relevant types of adversaries are co-resident and system-level ones, capturing VM-to-VM (or tenant-to-tenant) and VM to/from hypervisor threats. However, we do not consider side-channel attacks.

In the case of privacy we only consider network adversaries and malicious verifiers, since any adversary with direct access to other tenants' VMs or the hypervisor will immediately break privacy. We discuss more on privacy below.

Scheme overview. Our scheme begins with an initial setup, allowing parties to register long-term credentials and set up the virtualization platform. When a tenant wants a new VM, it registers with the platform, which allows the tenant to learn some information on that VM (and enable them to later establish secure-channels and to link attestations).

Attestations are of two types. Whenever a tenant tries to simply attest a VM, it will send a request to the VM (over a secure channel), and another request to the hypervisor, along with a nonce for each. The VM attestation proceeds as in [2] (since individual VM attestations are autonomous and do not jeopardize inter-tenant privacy). The hypervisor attestation will proceed in the following way. When receiving the request the hypervisor buffers it if the TPM is busy. As soon as the TPM is free, the hypervisor can retrieve linking information for the tenant and concatenate it with that tenant's nonces. It does the same for every other buffered tenant request.

Then it assigns a random position in a vector for each tenant, places the concatenated linking-values for each tenant at the assigned position, and commits to this obtained vector. If the number of requesting tenants is smaller than the number of positions in the vector (recall, we made the vector constant size), the empty positions are filled with dummy random data. The hypervisor can now request a quote to the TPM, sending the commitment as the nonce. When it receives the answer, it computes a SNARK confirming the validity of the quote without revealing the precise configuration. It also computes for each tenant an opening to the vector for its attributed position. Each tenant will receive the SNARK proof and the opening to the positions in the vector-commitment to which it can access. The verification and linking of the two attestations then proceeds as in [2].

Remark. Our work is motivated by the privacy problems which occur for deep attestation in multi-tenant environments. While we do not set out to *create* privacy within the physical platform (this must be enforced by correct separations of resources), we do, however *preserve* those properties at the attestation layer.

2 Preliminaries

Our multi-tenant attestation scheme relies on vector-commitments and ZK-SNARKs, which we briefly describe below before presenting our contributions.

2.1 Vector commitment

A commitment scheme $\mathsf{COM} = (\mathsf{COM.Setup}, \mathsf{COM.Com}, \mathsf{COM.Ver})$ is a cryptographic primitive that allows a party $\mathcal{P}$ to commit to a message m with some randomness r by producing a commitment c and opening information o: $(c, o) \leftarrow \mathsf{COM.Com}(m; r)$ It is then possible to verify that c is a valid commitment for message m by using the verification algorithm $\mathsf{COM.Ver}(m, c, o)$.

The commitment c should be *hiding* (it must reveal nothing about the committed message m) and *binding* (a commitment c will not open to a different message than m). In particular, given the opening information o, as well as c and m, anyone can verify the validity of c with respect to m.

Introduced by Catalano and Fiore in 2011 [7], vector commitment schemes allow for a commitment to a list of values, rather than a single message. Additionally, the opening of vector commitments is done by position, *i.e.*, one computes opening information for each value committed in the list, potentially separately. In this paper we will use this in order to provide tenants (owners of VMs) to only open attestation-specific information that is relevant to the VMs they own, and not to others. In contrast to classic commitment schemes, vector commitments do not always have the hiding property – which we require here. Thankfully, one can obtain this property by combining the vector-commitment scheme with a classical commitment scheme. Instead of directly committing to vector $v = (v_1, v_2, ..., v_n)$, one can apply a commitment scheme $\mathsf{COM.Com}$ to each value in the vector and *then* commit $v = (\mathsf{COM.Com}(v_1), \mathsf{COM.Com}(v_2), ..., \mathsf{COM.Com}(v_n))$, then during the reveal phase the opening of the commitment at position i will be added to the proof that m_i is in the committed vector.

We note that, while the scheme of Catalano and Fiore also featured updates to both the values committed to and to the opening information, we do not require this for our paper. Our vector commitment schemes will be of the form $\mathsf{VC} = (\mathsf{VC.Setup}, \mathsf{VC.Com}, \mathsf{VC.Open}, \mathsf{VC.Ver})$, such that:

- $\mathsf{VC.Setup}(1^\lambda, q) \rightarrow \mathsf{ppar}$: The setup algorithm (also called Key Generation by Catalano and Fiore) takes in input a security parameter in unary, and the length q of the vectors committed to, and outputs public parameters ppar, which include the message space $\mathcal{M}$.
- $\mathsf{VC.Com}(v) \rightarrow (c, aux)$: The commitment algorithm takes in input a vector $v \in \mathcal{M}^q$ (a vector of q entries, each entry a message in $\mathcal{M}$) and outputs a commitment c and an auxiliary value aux.
- $\mathsf{VC.Open}(m, i, aux) \rightarrow \pi_i$: The opening algorithm is run by the party that computes the commitment. On input an index $i \in \{1, \ldots, q\}$, a message m which will be at some position i within a vector commitment, and the index i itself, this algorithm outputs a proof π_i that m is the i-th message of the vector v associated with the commitment c.

– $\underline{\mathsf{VC.Ver}(m, c, i, \pi_i) \rightarrow b \in \{0,1\}}$: The verification algorithm takes in input a message m, a vector commitment c an index $i \in \{1, 2, \dots q\}$ and a proof π_i, and outputs a bit that indicates either that the message m is assumed to be the message committed to at the i-th position of v (in this case $b = 1$) or not ($b = 0$).

Security properties. We require the following properties for our vector-commitment schemes:

Position Binding : For every i and every polynomially-bounded adversary $\mathcal{A}$ the advantage $\mathsf{Adv}^{\mathsf{VCBind}}_{\mathsf{VC}}(\mathcal{A}) := \Pr[\exists i, \pi, \pi^* \text{ and } m, m^* \in \mathcal{M} : m \neq m^*, \mathsf{VC.Ver}(m, c, i, \pi) = \mathsf{VC.Ver}(m^*, c, i, \pi^*) = 1]$ is negligible.
Hiding : The hiding property for the vector-commitment scheme is defined in terms of the security experiment $G_{\mathsf{VCHide}}(\lambda)$ presented below.

Game $G_{\mathsf{VCHide}}(\lambda)$
$\mathsf{ppar} \leftarrow \mathsf{VC.Setup}(1^\lambda, q)$
$b \xleftarrow{r} \{0,1\}$
$(v_0, v_1 \in \mathcal{M}^q, \ell \in (\mathbb{N}_+)^m, \text{s.t. } m \leq q) \leftarrow \mathcal{A}(\mathsf{ppar})$
Abort if $\exists \ell_j \subset \ell$ s.t. $v_{0\ell_j} \neq v_{1\ell_j}$
$(c_b, aux) \leftarrow \mathsf{VC.Com}(v_b)$
$\forall j \in \{1, \dots, m\}$ compute $\pi_{\ell_j} \leftarrow \mathsf{VC.Open}(v_{b\ell_j}, i, aux)$
$d \leftarrow \mathcal{A}(c_b, aux, \{\pi_{\ell_j}\}_{j=1}^m)$
$\mathcal{A}$ wins iff. $b = d$

Fig. 2. The vector-commitment hiding game.

The adversary's advantage against the hiding game is defined as:

$$\mathsf{Adv}^{\mathsf{VCHide}}_{\mathsf{VC}}(\mathcal{A}) := \left| \Pr[\mathcal{A} \text{ wins } G_{\mathsf{VCHide}}(\lambda)] - \frac{1}{2} \right|. \tag{1}$$

There exist a variety of vector-commitment scheme with different characteristics, such as the commitment or proof size. For example, a commitment scheme can be constructed using a Merkle tree with $O(1)$ commitment size and $O(\log q)$ proof size for a vector of size q.

2.2 ZK-SNARKs

Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge (ZK-SNARK) are generic NIZK proof systems (permitting to prove validity of any NP statement for some witness w without revealing w) but for which the argument of knowledge is *succinct*: namely the proof grows sublinearly in the witness size and the verification time is, similarly, sublinear. For example, given a Boolean circuit, one can prove knowledge on an assignment of values input into the circuit that render the circuit output to TRUE, without revealing the assignment. By converting generic computations into instances of an NP problem it is possible to prove that inputs for some public computation yield a specific public result without revealing the inputs. ZK-SNARKS allow to make some of those input values public, while other input values remain private. Moreover, ZK-SNARK are non-interactive and thus only require one message from the prover to the verifier, namely the proof, which can later be verified by multiple independent verifier.

A ZK-SNARK consists of the following algorithms:

– $\underline{\mathsf{ZKP.Setup}(R) \rightarrow (\mathsf{CRS}, \tau)}$: The setup algorithm takes as input an NP relation R and then ouputs a common reference string –CRS– consisting of public proving and verification parameters, and a simulation trapdoor τ.
– $\underline{\mathsf{ZKP.Prove}(\mathsf{CRS}, R, x_{ZK}, w_{ZK}) \rightarrow \pi_{ZK}}$: The proving algorithm takes as input the CRS, a statement x_{ZK} and a witness w_{ZK} for which $(x_{ZK}, w_{ZK}) \in R$, and output a ZK-proof π_{ZK}.

- $\underline{\mathsf{ZKP.SkVer}(\mathsf{CRS}, R, x_{ZK}, \pi_{ZK}) \rightarrow b \in \{0,1\}}$: The verification algorithm takes in input a proof, the relation, a statement, and the CRS. It outputs 1 if the proof is accepted with respect to the CRS, relation, and statement, or 0 if the proof is rejected. An accepted (valid) proof convinces the verifier that the prover held a witness for the statement with respect to the provided relation.
- $\underline{\mathsf{ZKP.SkSim}(\mathsf{CRS}, \tau, R, x_{ZK}) \rightarrow \pi_{ZK}}$: On input the CRS, a simulation trapdoor, the relation, and a statement output a simulated proof.

Thus, there are two ways of generating a valid proof: either by holding a legitimate witness and using the $\mathsf{ZKP.Prove}$ algorithm, or by having access to the trapdoor τ and using $\mathsf{ZKP.SkSim}$.

Properties. ZK-SNARKS have the following properties:

Completeness : For all $(x_{ZK}, w_{ZK}) \in R$ with $(\mathsf{CRS}, \tau) \leftarrow \mathsf{ZKP.Setup}(R)$ and $\pi_{ZK} \leftarrow \mathsf{ZKP.Prove}(\mathsf{CRS}, R, x_{ZK}, w_{ZK})$, $\Pr[\mathsf{ZKP.SkVer}(\mathsf{CRS}, R, x_{ZK}, \pi_{ZK}) = 1] = 1$ (intuitively, all proofs correctly generated using a valid witness will verify as valid).
Knowledge Soundness : For all PPT adversaries $\mathcal{A}$ there exists an extractor $\mathsf{Ext}_{\mathcal{A}}$ and a negligible function ϵ with $(\mathsf{CRS}, \tau) \leftarrow \mathsf{ZKP.Setup}(R)$ and $((x_{ZK}, \pi^*); w_{ZK}) \leftarrow \mathcal{A}||\mathsf{Ext}_{\mathcal{A}}$ such that $\Pr[(x_{ZK}, w_{ZK}) \notin R \wedge \mathsf{ZKP.SkVer}(\mathsf{CRS}, R, x_{ZK}, \pi_{ZK}) = 1] = \epsilon(\lambda)$.
Zero-knowledge : For all $(x_{ZK}, w_{ZK}) \in R$ with $(\mathsf{CRS}, \tau) \leftarrow \mathsf{ZKP.Setup}(R)$ the distributions $D_0 = \{\pi_0 \leftarrow \mathsf{ZKP.Prove}(\mathsf{CRS}, x_{ZK}, w_{ZK})\}$ and $D_1 = \{\pi_1 \leftarrow \mathsf{ZKP.SkSim}(\mathsf{CRS}, \tau, x_{ZK})\}$ are statistically close.

Many different ZK-SNARK schemes exist and provide different properties. Notably we can differentiate pre-processing SNARK from transparent SNARK. Pre-processing SNARK require a trusted setup to compute the CRS as they generate a so called "toxic waste". Moreover some of the pre-processing SNARK scheme are circuit specific meaning that a different CRS must be generated for different circuit.

3 Model

Our security model applies to the virtualization architecture in Figure 1, in which tenants associated with unique identities $\mathcal{T}$ can register[3] a number of virtual machines VM on a hypervisor $\mathcal{H}$.

Each hypervisor $\mathcal{H}$ will have a physical root of trust represented by a TPM TPM. Moreover in order to provide a meaningful inter-tenant privacy notion, we assume that each physical machine (with a unique hypervisor $\mathcal{H}$) upper-bounds the number of tenants $N_{\mathcal{T}}$ that it can host, and also the number of VMs N_{VM} that each tenant can have on $\mathcal{H}$. Such bounds do exist in practice, usually driven by physical constraints. In our case, for the sake of legibility, we will assume *universal* bounds (all hypervisors may only have $N_{\mathcal{T}}$ and N_{VM} per tenant), rather than *local*, hypervisor-specific ones.

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

3.1 Primitive syntax

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

$\underline{\mathtt{Setup}(1^\lambda) \rightarrow \{\mathsf{ppar}, \mathsf{spar}\}}$: On input a security parameter, 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.

[3] We allow tenants (who are potential privacy adversaries) to arbitrarily choose the physical machine and hypervisor hosting their VMs. This strong security model allows attackers to thus exploit multiple physical platforms before choosing their target.

$\mathtt{HSetup}(\mathsf{ppar}) \to \{\mathcal{H}.\mathsf{pk}, \mathcal{H}.\mathsf{sk}, \mathsf{AK.pk}, \mathsf{AK.sk}, \mathcal{H}.\mathsf{Conf}, \mathcal{H}.\mathsf{state}\}$:
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; this value is initialized to $\emptyset$.

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

$\mathtt{VMReg}(\mathcal{H}, \mathcal{T}.\mathsf{sk}, \mathsf{VMdesc}) \to \{(VM, \mathsf{VAK.pk}), \mathsf{VAK.sk}, \mathcal{H}.\mathsf{state}\} \cup \perp$: This is 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 $\perp$. Else, 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 $\perp$, otherwise it returns, to the tenant, the handle VM and the public keys and $\mathsf{VAK.pk}$.

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

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

$\mathtt{VMAttest}\langle VM(\mathsf{VAK.sk}), \mathcal{T}(\mathcal{T}.\mathsf{sk}, \mathsf{nonce})\rangle \to \{\mathsf{ATT}_{VM}\} \cup \perp$: 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 $\perp$ (typically if the tenant does not own VM) or a VM attestation ATT_{VM}.

$\mathtt{VfVMAttest}(\mathsf{ATT}_{VM}, \mathsf{nonce}, \mathsf{link}) \to \{0,1\}$: On 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.

$\mathtt{Link}(\mathsf{ATT}_{\mathcal{H},\mathcal{T}}, \mathsf{nonce}_{\mathcal{T}}, \mathsf{link}_{\mathcal{T}}, \mathsf{ATT}_{VM}, \mathsf{nonce}, \mathsf{link}) \to \{0,1\}$: on 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 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 Adversary model

Our threat model features both attestation and privacy adversaries. Privacy is usually orthogonal to typical security notions in attestation. Attestation seeks to protect against internal adversaries, with a direct access to the target platform and its files; however, the privacy we can aim for with respect to insiders is only limited. Our protocol preserves privacy, but does not create it: in order for privacy to be guaranteed, we need to ensure first that privacy attackers (such as tenants) only have limited access to the physical platform. Tenants that share information (or VMs) with another tenants will lose their privacy. Note that access to the platform can be gained in different ways, some legitimate (*e.g.*, hypervisor API calls) and some malicious (*e.g.*, side channel).

In our privacy models, the hypervisor and VMs are honest (trusted). Our adversaries are typically collusions of malicious tenants that can actively send messages during the protocol, working together with a Dolev-Yao network attacker, which can eavesdrop, modify, insert and delete messages.

On the other hand, layer-linking is defined with respect to classical attestation adversary – in which the attacker has direct, insider access to the platform. In this model, the tenant and verifier are trusted but we consider a Dolev-Yao attacker as well as software adversaries from [34] which can compromise the software of any VM (co-resident) or hypervisor (system level). However, we rule out hardware adversaries, including side-channel attacks.

Remark: side-channels. Many side-channel attacks (SCAs) have been explored in the context of virtualization, on physical, system, and application levels. Some are related to the micro-architecture [6], controlled-channels (*i.e.*, page-fault attacks) [36], cache-timing attacks [41] or speculative execution [8, 39]. While numerous counter-measures exist (addressing SCAs at each layer or in cross-layer mode), industrial platforms [34] do not consider SCAs and seldom implement counter-measures against them [26]. Although in this work we will not consider SCAs, we do recommend the use of available counter-measures to address such threats.

Security/privacy notions. We formally define the privacy properties required for our protocol in Sections 3.3 and 3.4. For attestation security we require an extension of the linking property formalized by [2], 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 Appendix 3.5.

3.3 Inter-tenant Privacy

Inter-tenant privacy is defined by means of a game between a challenger $\mathcal{G}$ and an adversary $\mathcal{A}$. The challenger runs the setup algorithm $\mathtt{Setup}(1^\lambda)$, then sets up a hypervisor $\mathcal{H}$ by running $\mathtt{HSetup}(\mathsf{ppar})$. Then, $\mathcal{G}$ initiates $\mathcal{L}_H := \emptyset$ and $\mathcal{L}_C := \emptyset$. The challenger finally draws a random bit $b \xleftarrow{r} \{0,1\}$. 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 the bit b . On input a set of VM descriptions $\overline{\mathsf{VMDesc}_i}$, this oracle internally runs the key-generation algorithm $\mathtt{TKGen}(\mathsf{ppar})$, receiving either $\perp$ (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 $\perp$: if $b = 1$, the oracle runs $\mathtt{VMReg}(\mathcal{H}, \mathcal{T}.\mathsf{sk}, \mathsf{VMDesc}_i)$ 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: $\mathcal{T}.\mathsf{pk}, \{\mathsf{VAK.pk}_i\}_{i=1}^{\ell}$. Note that, if $\ell > N_{VM}$, then 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 $\mathcal{L}_C$.
- $\underline{\mathsf{oVMReg}(\mathcal{T}, \mathsf{VMDesc})}$: on input a (registered) tenant $\mathcal{T} \in \mathcal{L}_H$ and a VM with description $\overline{\mathsf{VMDesc}}$, this oracle internally runs $\mathtt{VMReg}(\mathcal{H}, \mathcal{T}.\mathsf{sk}, \mathsf{VMDesc})$ and, 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. Note that a malicious tenant can always register a new VM by running the $\mathtt{VMReg}$ algorithm directly.
- $\underline{\mathsf{oHAttest}(\mathcal{T})}$: on input a registered tenant $\mathcal{T} \in \mathcal{L}_H$, this oracle simulates a run of $\mathtt{HAttest}$ between the honest tenant and the hypervisor $\mathcal{H}$. The adversary gains a transcript τ_{HAtt} of the communication (possibly a single symbol $\perp$ in case of error – for instance if $\mathcal{H}$ does not exist or if $\mathcal{T}$ has no VMs registered on $\mathcal{H}$). Importantly, 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 of $\mathcal{H}$ and the VMs currently existing on the machine.

Definition 1 (Inter-tenant privacy). *A PP-MTA scheme* $\mathsf{PP\text{-}MTA} = (\mathtt{Setup}, \mathtt{HSetup}, \mathtt{TKGen}, \mathtt{VMReg}, \mathtt{HAttest}, \mathtt{VMAttest}, \mathtt{VfHAttest}, \mathtt{VfVMAttest}, \mathtt{Link})$ *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 $\mathsf{PP\text{-}MTA}$. *Asymptotically, we call a PP-MTA scheme* inter-tenant private *if* ϵ *is a negligible function of the security parameter* λ.

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

Fig. 3. The inter-tenant privacy game.

3.4 Hypervisor Configuration Privacy

In the hypervisor configuration-privacy game, the adversary gets access to the following oracle:

$\mathsf{oChooseConfig}_b(\mathcal{H}.\mathsf{Conf}_0, \mathcal{H}.\mathsf{Conf}_1) \rightarrow \{\mathsf{OK}\} \cup \perp$: This oracle can only be called once. On 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 $\mathcal{H}.\mathsf{Conf}_1 \in \mathcal{CONF}$, then also that $\mathcal{H}$ has not yet been set up (*e.g.*, through `HSetup`). If either verification fails, the oracle outputs $\perp$. If the verification succeed, then the oracle calls `HSetup`, forcing the picked hypervisor configuration $\mathcal{H}.\mathcal{H}.\mathsf{Conf}$ to be $\mathcal{H}.\mathsf{Conf}_b$.

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

Fig. 4. The configuration-privacy game.

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

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:*

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

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 $\frac{1}{|\mathcal{CONF}|}$).

3.5 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 [2].

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 $\mathtt{Setup}(1^{\lambda})$, returns ppar to $\mathcal{A}$, then sets up an hypervisor $\mathcal{H}$ by running $\mathtt{HSetup}(\mathsf{ppar})$ 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:

- $\mathsf{oHAttest}(\mathbb{S}_i) \rightarrow (\mathsf{ATT}_{\mathcal{H},\mathcal{T}})$: this oracle simulates a run of the $\mathtt{HAttest}$ 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 $\mathcal{L}_{Att}$.
- $\mathsf{oVMAttest}(VM) \rightarrow (\mathsf{ATT}_{VM})$: this oracle simulates a run of the $\mathtt{VMAttest}$ 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 $\mathcal{L}_{Att}$;
- $\mathtt{Link}(\mathcal{P}||\mathcal{Q}) = 1$.

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* $\mathsf{PP\text{-}MTA} = (\mathtt{Setup}, \mathtt{HSetup}, \mathtt{TKGen}, \mathtt{VMReg}, \mathtt{HAttest}, \mathtt{VMAttest}, \mathtt{VfHAttest}, \mathtt{VfVMAttest}, \mathtt{Link})$ *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 $\mathsf{PP\text{-}MTA}$. *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 $\mathsf{PP\text{-}MTA}$ primitive using a signature scheme ($\mathtt{SigKGen}, \mathtt{SigSig}$, $\mathtt{SigVer}$), an unkeyed collision-resistant hash function H, a vector commitment scheme ($\mathsf{VC}.\mathtt{Setup}$, $\mathsf{VC}.\mathtt{Com}$, $\mathsf{VC}.\mathtt{Open}$, $\mathsf{VC}.\mathtt{Ver}$), a ZK-SNARK scheme ($\mathsf{ZKP}.\mathtt{Setup}$, $\mathsf{ZKP}.\mathtt{Prove}$, $\mathsf{ZKP}.\mathtt{SkVer}$, $\mathsf{ZKP}.\mathtt{SkSim}$), and a secure-channel establishment protocol (in practice TLS 1.3): $\mathsf{AKE} = (\mathsf{AKE}.\mathtt{KGen}, \mathsf{AKE}.\mathtt{AKE}, \mathsf{AKE}.\mathtt{Enc}, \mathsf{AKE}.\mathtt{Dec})$.

We present our scheme by functionality: Section 4.1 describes the setup steps; then Section 4.2 delves into the VM registration; next, in Section 4.3 we describe arguably the most important and novel component of our scheme: the hypervisor attestation; in Section 4.4 we outline VM attestations; and in Section 4.5 we finish by describing the linking of hypervisor and VM attestations.

4.1 Setup

In this step, we instantiate the global setup algorithm $\texttt{Setup}$ and the hypervisor-setup algorithm $\texttt{HSetup}$, then set up the tenants by generating their long-term parameters.

Global setup. The goal here is to instantiate the scheme's global public and private parameters: $\{\mathsf{ppar}, \mathsf{spar}\} \leftarrow \texttt{Setup}(1^\lambda)$. Universal bounds $N_{\mathcal{T}}$ and N_{VM} are chosen, according to limits allowed by providers to tenants.

Then, a set of *plausible* configurations $\mathcal{CONF}$ is chosen (depending on the hardware). The choice of the configuration set is independent of the tenants, malicious or honest. We will moreover assume in the security proof that all values in $\mathcal{CONF}$ have equal probability of occurring on the given machine.

We set up the vector commitment and ZK-SNARK schemes:

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

$$(CRS, \tau) \leftarrow \mathsf{ZKP}.\texttt{Setup}(R)$$

Notice that the vector-commitment length is a constant $N_{\mathcal{T}}$. Then 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 $\mathcal{H}.\mathsf{Conf} \xleftarrow{r} \mathcal{CONF}$ 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:

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

We depict in Figure 5 the registration algorithm. Note that the VM is hosted on, and managed by the hypervisor: thus, the exchange between them is local to the device.

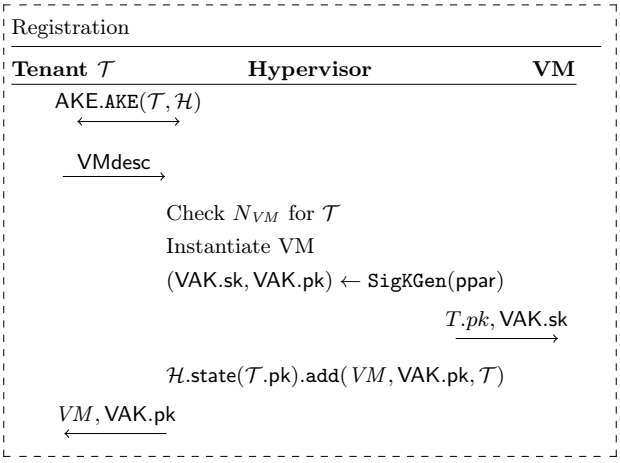

Fig. 5. Registration, the tenant (successfully) registers a new VM of description VMdesc.

When a registration request is made, $\mathcal{H}$ and $\mathcal{T}$ run AKE.AKE (using their long-term credentials) to open an authenticated secure channel. At the end of a successful AKE protocol session, both parties compute (a number of) session keys. All future communication takes place over a secure channel (*i.e.*, all future messages are encrypted using AKE.Enc on the sending end, and decrypted using AKE.Dec on the receiving end).

Over this secure channel, $\mathcal{T}$ requests the registration of a VM with description VMdesc. The hypervisor verifies it can still allow tenant to register a new VM (regarding N_{VM}). If this is not so, the algorithm aborts. Otherwise, the hypervisor generates attestation (signature) keys for the newly-registered VM: (VAK.sk, VAK.pk) ← SigKGen(ppar). The keys VAK.pk and $\mathcal{T}$.sk will be stored in the vTPM corresponding to the new VM.

Once the VM created, $\mathcal{H}$ updates its internal state $\mathcal{H}$.state with entries as (VM, 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

One of the most novel procedures in our construction is the hypervisor attestation. We briefly recall the challenge: we want to have a hypervisor attestation which will be linkable to VM attestations, but which guarantees inter-tenant privacy and hypervisor-configuration privacy. We instantiate the algorithm as :

$$\{\mathsf{ATT}_{\mathcal{H},\mathcal{T}}\} \leftarrow \mathtt{HAttest}\langle \mathcal{T}(\mathcal{T}.\mathsf{sk}, \mathsf{nonce}_{\mathcal{T}}), \mathcal{H}(\mathcal{H}.\mathsf{sk}, \mathsf{AK.sk}, \mathcal{H}.\mathsf{state}, \mathcal{H}.\mathsf{Conf})\rangle$$

Recall that [2] shows how to achieve layer linking for a single-tenant environment. The idea is to embed, into the hypervisor attestation, elements that uniquely characterise each VM. During hypervisor attestation, the hypervisor retrieves public attestation key stored on each vTPM managed by the hypervisor. Those values are concatenated with the nonce and hashed to obtain new nonce.

In the multi-tenant context, that would not work, as information about *all* the VMs hosted on the machine leaks to all the recipients of the attestation. We adapt the original idea of [2] to the multi-tenant setting by the use of vector commitments, and subsequent use of SNARKs.

Authenticated key-exchange. At the beginning of the attestation process, the hypervisor and tenant establish a secure channel with mutual authentication. All subsequent communication takes place over a secure channel.

Preparation of vector commitment. An attestation can be requested by one or more tenants (authenticated over a secure channel), each providing a nonce to the hypervisor. The hypervisor randomly chooses, for each tenant, an index i between 1 and $N_{\mathcal{T}}$.

Once the indices are set, the hypervisor retrieves the VAK.pk of all the VMs registered by the tenant(s) that requested an attestation and then concatenates, for each tenant, the nonce that

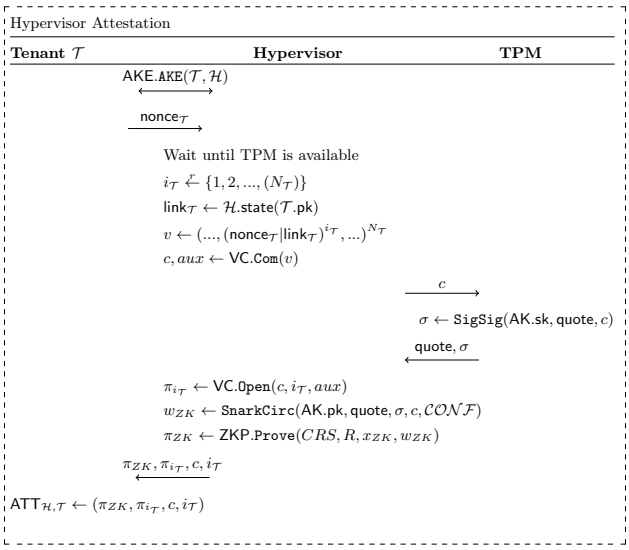

Fig. 6. Hypervisor Attestation, only the i-th tenant is represented but we can see the aggregation of all the nonce through commitment which allow a single TPM operation.

tenant provided and each of the public keys of the VMs it owns. The list of $\mathsf{VAK.pk}$ of the VM a tenant owns constitutes the linking information for that tenant (*link* in Fig. 6). If the number of requesting tenants is less than $N_{\mathcal{T}}$, empty position are filled with random values. Thus, the commitment vector is always of constant size.

The vector commitment must be hiding, in order to hide, from each tenant, all the values for which it will not (later on) receive opening information.

Note that using multiple, diverse nonces given by multiple tenants is scalable. Say that two or more tenants request hypervisor attestation while the TPM is busy. Without the nonce-aggregation step, those requests would be treated separately, which is inefficient. As it is, the 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 in the physical device (the tenant has no access to the communication). The hypervisor submits to the 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 can now compute a proof it has a valid attestation. Note first that the attestation quote (and corresponding signature) reveal the configuration of the hypervisor – which we want to keep private from the tenants. Instead, the hypervisor needs to prove that it has received a valid attestation from the TPM, for a configuration within the set $\mathcal{CONF}$, with respect to the nonce c, *i.e.*, it needs to prove that $\mathsf{ZK\text{-}SNARK}\{(\mathsf{quote}, \sigma) : \mathtt{SigVer}(\mathsf{AK.pk}, \mathsf{quote}, \sigma, c) == 1 \ \wedge \mathsf{quote}.\mathcal{H}.\mathsf{Conf} \in \mathcal{CONF}\}$.

Consider Algorithm 1, which consists of verifying the signature and then checking that the content of the attestation is among the set $\mathcal{CONF}$. The algorithm outputs 1 if both are true and 0 otherwise.

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.

Opening. Finally, the hypervisor needs to provide to each tenant the vector-commitment opening on its allowed positions. The tenant uses the opening to check its nonce and find the linking information. The hypervisor finally sets, for each tenant: $\mathsf{ATT}_{\mathcal{H},\mathcal{T}}$ consisting of: 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.

Algorithm 1 The snark circuit

```
procedure SnarkCirc(AK.pk, quote, σ, c, CONF)
    if SigVer(AK.pk, quote, σ, c) == 1 and quote.H.Conf ∈ CONF then
        return 1
    else
        return 0
    end if
end procedure
```

Hypervisor attestation verification. Upon receiving an attestation $\mathsf{ATT}_{\mathcal{H},\mathcal{T}}$, each tenant verifies the attestation and links it with its VM attestations – by a process described in Section 4.5. For the hypervisor attestation verification, we instantiate the algorithm $\{0,1\} \leftarrow \mathtt{VfHAttest}(\mathsf{ATT}_{\mathcal{H},\mathcal{T}}, \mathsf{nonce}_{\mathcal{T}}, \mathsf{link}_{\mathcal{T}})$. The tenant verifies that the opening information for the relevant index of the vector commitment c opens to the concatenation of the nonce $\mathsf{nonce}_{\mathcal{T}}$ and linking $\mathsf{link}_{\mathcal{T}}$. If this verification fails, the algorithm outputs 0; if it succeeds, the tenant will verifies the ZK-SNARK proof:

$\mathtt{VfHAttest}(\mathsf{ATT}_{\mathcal{H},\mathcal{T}}, \mathsf{nonce}_{\mathcal{T}}, \mathsf{link}_{\mathcal{T}}) \rightarrow \{0,1\}$:

Parse $\mathsf{ATT}_{\mathcal{H},\mathcal{T}}$ as $(\pi_{ZK}, \pi_{i_{\mathcal{T}}}, c, i_{\mathcal{T}})$

$\mathsf{VC}.\mathtt{Ver}((\mathsf{nonce}_{\mathcal{T}}|\mathsf{link}_{\mathcal{T}}), c, i_{\mathcal{T}}, \pi_{i_{\mathcal{T}}})$

$\mathsf{ZKP}.\mathtt{SkVer}(CRS, R, x_{ZK}, \pi_{ZK})$

If all verification pass output 1, otherwise 0

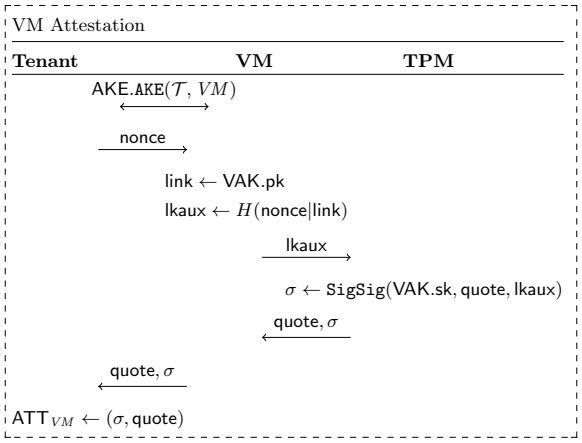

Fig. 7. VM Attestation

4.4 VM Attestation

The algorithm for VM attestation generates a quote such that only the tenant which owns the VM can actually attest it: $\{\mathsf{ATT}_{VM}\} \leftarrow \mathtt{VMAttest}(VM(\mathsf{VAK.sk}, \mathsf{VAK.pk}), \mathcal{T}(\mathcal{T}.\mathsf{sk}, \mathsf{nonce}))$. We achieve this similarly to traditional attestation; however, instead of using the nonce received from the verifier, the VM will hash the concatenation of the nonce and its public endorsement key VAK.pk.

We depict VM attestation in Figure 7. The tenant and VM run an AKE protocol to establish a secure channel, over which $\mathcal{T}$ requests an attestation and forwards a nonce nonce. The VM retrieves the linking information (specifically 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.

Verification of VM attestation. The verification algorithm, $\{0,1\} \leftarrow \mathtt{VfVMAttest}(\mathsf{ATT}_{VM}, \mathsf{nonce}, \mathsf{link})$, is straightforward: the tenant retrieves the lkaux value as

above; then verifies the validity of the signature on the received quote. If the verification succeeds, it outputs 1 otherwise, 0.

$\texttt{VfVMAttest}(\mathsf{ATT}_{VM}, \mathsf{nonce}, \mathsf{link}) \rightarrow \{0,1\}$:

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

4.5 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

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 $\pi^i_{\mathcal{P}}$. 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*:

- $\pi^i_{\mathcal{P}}.\mathsf{sid}$: the session identifier of instance $\pi^i_{\mathcal{P}}$ 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.
- $\pi^i_{\mathcal{P}}.\mathsf{pid}$: the partner identifier of instance $\pi^i_{\mathcal{P}}$, which must be a party $\mathcal{Q} \in \mathbb{P} \setminus \mathcal{P}$.
- $\pi^i_{\mathcal{P}}.\alpha$: 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 $\pi^i_{\mathcal{P}}.\mathsf{pid}$.

- $\pi^i_{\mathcal{P}}.\mathsf{k}$: the session key of instance $\pi^i_{\mathcal{P}}$, 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:

- $\pi^i_{\mathcal{P}} \leftarrow \mathsf{oNewSession}(\mathcal{P}, \mathcal{Q}, \mathsf{role})$: the (honest) session creation oracle will initiate a new instance $\pi^i_{\mathcal{P}}$ with partner identifier $\pi^i_{\mathcal{P}}.\mathsf{pid} = \mathcal{Q}$, such that $\pi^i_{\mathcal{P}}$ plays the role designated by role (either initiator or responder) in its session.
- $m^* \leftarrow \mathsf{oSend}(\pi^i_{\mathcal{P}}, m)$: the (honest) sending oracle models sending message m to an already-existent instance $\pi^i_{\mathcal{P}}$. It is expected that $\pi^i_{\mathcal{P}}$ returns a message m^*, which is the protocol-specific reply (potentially an error symbol $\perp$) as a response. A special $m = \texttt{Start}$ 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).
- $k \leftarrow \mathsf{oReveal}(\pi^i_{\mathcal{P}})$: the revelation oracle allows the adversary to learn already-established session keys k.
- $\pi^i_{\mathcal{P}} \leftarrow \mathsf{oNewSession}_{b,\mathsf{role}}(\mathcal{P}, \mathcal{Q})$: 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 $\pi^i_{\mathcal{P}}$. The oracle forwards the handle $\pi^i_{\mathcal{P}}$ 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.
- $m^* \leftarrow \mathsf{oSend}_b(\pi^i_{\mathcal{P}}, m)$: this left-or-right version of the sending oracle allows the message m to be either forwarded to $\pi^i_{\mathcal{P}}$ (if $b = 0$) or to the simulator Sim otherwise. In both cases the adversary expects a message m^*. As before, a special message $m = \texttt{Start}$ 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.

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

Fig. 8. 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)* 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 [1] 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 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 [4].

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 $\binom{(q_{\mathsf{oNewSession}} + q_{\mathsf{oNewSession}_{b,\mathsf{Role}}})}{2}$.

[4] 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.

$\mathbb{G}_2$: 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 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 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 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 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 $\mathsf{oHAttest}$ 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 $\mathsf{oHAttest}$ 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 $\mathsf{oHAttest}$ 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 position-binding and hiding properties for vector commitments ensure that the verifier (*i.e.*, tenant) does not learn any information about other participants. This is because the attestation's content regarding a given tenant is correct (position binding) and other data of the committed vector is indistinguishable from random (hiding). However, a malicious hypervisor could replay an old attestation by finding a collision on the vector commitment.

Those attacks do not alter the security guarantees of the tenant (*i.e.*, position binding/hiding), but the verifier cannot be convinced that the vector commitment has been correctly computed (the hypervisor did not manipulate other positions). The collision resistance for vector commitment, combined with the position binding, ensures that the vector is *fresh*, meaning no replays are possible.

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 9), 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.

$$\frac{\text{Game } G_{\mathsf{VC-Coll}}(\lambda, n)}{\{\mathsf{ppar}\} \leftarrow \mathsf{VC.Setup}(1^\lambda, n)}$$
$$\frac{(v, v') \leftarrow \mathcal{A}(\mathsf{ppar})}{\mathcal{A} \textbf{ wins iff.: } \exists i \text{ such that } v[i] \neq v'[i] \text{ and } \mathsf{VC.Com}(v) = \mathsf{VC.Com}(v')}$$

Fig. 9. 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. The code will be open-source.

Implementation details. In what follows, we describe some of the more significant details of our implementation.
TPM libraries: To communicate with the (physical or virtual) TPM, we used the software provided by tpm2-software community [11], 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 [38] combined with a basic hash-based commitment scheme.

SNARK: We used bellperson [19] which implements a preprocessing circuit-specific CRS SNARK [21] 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 [35]. a library to compute arbitrary-precision arithmetic operations inside SNARKs.

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.

Benchmark results. 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).

The Rust part of the code was measured using the Criterion library [22] 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.

VM Attestation: Table 1 presents our comparative results for the VM attestation in our protocol versus a basic VM attestation (both are computed by vTPM). The difference between these two procedures is that, in the case of our protocol, we have to compute the hash of the concatenation of the linking information (the public key) and the nonce before signing and verify the signature. Hence, the overhead introduced by our scheme is very limited.

	Mean	Median
Basic Attestation	103.66	103.44
VM Attestation	104.78	104.24
Basic Attestation Verification	2.53	2.50
VM Attestation Verification	2.54	2.53

Table 1. Time in ms to perform Basic vs VM Attestation.

Hypervisor Attestation: Table 2 show the difference in time between a traditional attestation and a hypervisor attestation. In this table the results are given for a set of size 128 and a for a request of attestation from 100 tenant each of them with one VM. In this case the TPM attestation by itself is computed on a hardware TPM. As the table show most of the overhead is due to the ZK-SNARK for both the attestation and the verification.

	Mean	Median
Basic Attestation (s)	0.94	0.94
Total Hypervisor Attestation (s)	2.40	2.40
Snark Proof (s)	1.46	1.46
Commitment and Opening (ms)	9.06	8.98
Basic Attestation Verification (ms)	2.42	2.36
Total Hypervisor Attestation Verification(ms)	25.06	25.05
Snark proof Verification (ms)	25.02	24.99
Nonce Membership Verification (ms)	0.043	0.063

Table 2. Time to perform Basic vs Hypervisor Attestation

The graph of Figure 10 shows the hypervisor attestation scaling with the number of tenants. Although the complexity increases with the number of tenants and VMs, the overall performance loss is not overwhelming, mostly due to our aggregation mechanism.

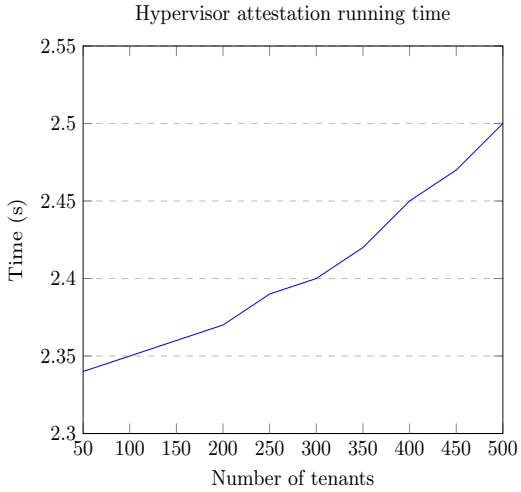

Fig. 10. How hypervisor attestation scales with the number of tenants, for a configuration-set of size 128.

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 3 shows how the setup, prover, and verifier algorithm performances change with set size.

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

Table 3. Performance variations of the ZK-SNARK in the configuration-set size. Median value over 100 samples.

Linking Table 4 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 4.5 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.

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

Table 4. Performance of the SNARK depending on the size of the set. Median value over 100 samples.

7 Conclusion and Discussion

In this work we proposed a scalable and efficient TPM attestation scheme for multi-tenant environments. The scheme requires no modifications to the TPM, no unrealistic trust assumptions (like an attestation proxy), and provides strong privacy for both the tenants and the hypervisor. Moreover our approach provides layer-binding.

Our scheme achieves privacy by relying on vector commitments and SNARKs. The latter primitive, in particular, incurs a high overhead – but has the merit of not requiring TPM modifications. Although our work is a stepping stone towards designing privacy-preserving multi-tenant attestation, more research is needed into aspects of present-day virtualization such as migration, which we leave as future work.

References

1. Arfaoui, G., Bultel, X., Fouque, P., Nedelcu, A., Onete, C.: The privacy of the TLS 1.3 protocol. Proc. Priv. Enhancing Technol. **2019**(4), 190–210 (2019)
2. 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: Ateniese, G., Venturi, D. (eds.) Applied Cryptography and Network Security - 20th International Conference, ACNS 2022, Rome, Italy, June 20-23, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13269, pp. 399–418. Springer (2022)
3. Berger, S., Caceres, R., Goldman, K.A., Perez, R., Sailer, R., van Doorn, L.: vTPM: Virtualizing the trusted platform module. In: 15th USENIX Security Symposium (USENIX Security 06). USENIX Association, Vancouver, B.C. Canada (Jul 2006)
4. Berger, S., Goldman, K., Pendarakis, D., Safford, D., Valdez, E., Zohar, M.: Scalable attestation: A step toward secure and trusted clouds. In: 2015 IEEE International Conference on Cloud Engineering. pp. 185–194 (2015). https://doi.org/10.1109/IC2E.2015.32
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. pp. 326–349. ACM (2012)
6. Brasser, F., Müller, U., Dmitrienko, A., Kostiainen, K., Capkun, S., Sadeghi, A.R.: Software grand exposure: Sgx cache attacks are practical. In: Proceedings of the 11th USENIX Conference on Offensive Technologies. p. 11. WOOT'17, USENIX Association, USA (2017)
7. Catalano, D., Fiore, D.: Vector commitments and their applications. In: Kurosawa, K., Hanaoka, G. (eds.) Public-Key Cryptography – PKC 2013. pp. 55–72. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
8. Chen, G., Chen, S., Xiao, Y., Zhang, Y., Lin, Z., Lai, T.H.: Sgxpectre: Stealing intel secrets from sgx enclaves via speculative execution. In: 2019 IEEE European Symposium on Security and Privacy (EuroSI&P). pp. 142–157 (2019)
9. Chen, L., Landfermann, R., Löhr, H., Rohe, M., Sadeghi, A.R., Stüble, C.: A protocol for property-based attestation. In: Proceedings of the First ACM Workshop on Scalable Trusted Computing. p. 7–16. STC '06, Association for Computing Machinery, New York, NY, USA (2006)
10. Chen, L., Löhr, H., Manulis, M., Sadeghi, A.R.: Property-based attestation without a trusted third party. In: Proceedings of the 11th International Conference on Information Security. p. 31–46. ISC '08, Springer-Verlag, Berlin, Heidelberg (2008)
11. tpm2-software community: Linux tpm2 & tss2 software (2022)
12. Debes, H.B., Giannetsos, T., Krontiris, I.: Blindtrust: Oblivious remote attestation for secure service function chains (2021)
13. Eckel, M., Fuchs, A., Repp, J., Springer, M.: Secure attestation of virtualized environments. In: Hölbl, M., Rannenberg, K., Welzer, T. (eds.) ICT Systems Security and Privacy Protection. pp. 203–216. Springer International Publishing, Cham (2020)
14. ETSI: Network functions virtualisation (nfv); use cases. Group specification, European Telecommunications Standards Institute (10 2013)
15. ETSI: Network functions virtualisation (nfv); nfv security; problem statement. Group specification, European Telecommunications Standards Institute (10 2014)
16. ETSI: Network functions virtualisation (nfv); trust; report on attestation technologies and practices for secure deployments. Group report, European Telecommunications Standards Institute (10 2017)
17. ETSI: Network functions virtualisation (nfv); security; report on nfv remote attestation architecture. Group report, European Telecommunications Standards Institute (11 2019)
18. 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 **19**(3), 689–718 (Sep 2019)
19. Filecoin: bellperson (2022)
20. Foy, K.: Keylime software is deployed to ibm cloud (2021)
21. Groth, J.: On the size of pairing-based non-interactive arguments. In: Fischlin, M., Coron, J.S. (eds.) Advances in Cryptology – EUROCRYPT 2016. pp. 305–326. Springer Berlin Heidelberg, Berlin, Heidelberg (2016)
22. Jorge, A., Heisler, B.: criterion (2022)
23. Keylime dev: Keylime (2022)
24. Larsen, B., Debes, H.B., Giannetsos, T.: Cloudvaults: Integrating trust extensions into system integrity verification for cloud-based environments. In: Computer Security: ESORICS 2020 International Workshops, DETIPS, DeSECSys, MPS, and SPOSE, Guildford, UK, September 17–18, 2020, Revised Selected Papers. p. 197–220. Springer-Verlag, Berlin, Heidelberg (2020)
25. Lauer, H., Kuntze, N.: Hypervisor-based attestation of virtual environments. In: 2016 Intl IEEE Conferences on Ubiquitous Intelligence Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld). pp. 333–340 (2016)

26. Liu, W., Chen, H., Wang, X., Li, Z., Zhang, D., Wang, W., Tang, H.: Understanding tee containers, easy to use? hard to trust (2021)
27. 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)
28. Ruan, A., Martin, A.: Repcloud: Achieving fine-grained cloud tcb attestation with reputation systems. In: Proceedings of the Sixth ACM Workshop on Scalable Trusted Computing. p. 3–14. STC '11, Association for Computing Machinery, New York, NY, USA (2011)
29. Sadeghi, A.R., Stüble, C.: Property-based attestation for computing platforms: Caring about properties, not mechanisms. In: Proceedings of the 2004 Workshop on New Security Paradigms. p. 67–77. NSPW '04, Association for Computing Machinery, New York, NY, USA (2004)
30. Santos, N., Rodrigues, R., Gummadi, K.P., Saroiu, S.: Policy-Sealed data: A new abstraction for building trusted cloud services. In: 21st USENIX Security Symposium (USENIX Security 12). pp. 175–188. USENIX Association, Bellevue, WA (Aug 2012)
31. Schäge, S., Schwenk, J., Lauer, S.: Privacy-preserving authenticated key exchange and the case of ikev2. In: Proceedings of PKC. LNCS, vol. 12111, pp. 567–596 (2020)
32. Schear, N., Cable, P.T., Moyer, T.M., Richard, B., Rudd, R.: Bootstrapping and maintaining trust in the cloud. In: Proceedings of the 32nd Annual Conference on Computer Security Applications. p. 65–77. ACSAC '16, Association for Computing Machinery, New York, NY, USA (2016)
33. Schiffman, J., Moyer, T., Vijayakumar, H., Jaeger, T., McDaniel, P.: Seeding clouds with trust anchors. In: Proceedings of the 2010 ACM Workshop on Cloud Computing Security Workshop. p. 43–46. CCSW '10, Association for Computing Machinery, New York, NY, USA (2010)
34. Schneider, M., Masti, R.J., Shinde, S., Capkun, S., Perez, R.: Sok: Hardware-supported trusted execution environments (2022)
35. Setty, S.: bellperson-nonnative (2022)
36. Shih, M.W., Lee, S., Kim, T., Peinado, M.: T-sgx: Eradicating controlled-channel attacks against enclave programs. In: Network and Distributed System Security Symposium 2017 (NDSS'17). Internet Society (February 2017), https://www.microsoft.com/en-us/research/publication/t-sgx-eradicating-controlled-channel-attacks-enclave-programs/
37. TCG Virtualized Platform Working Group: Virtualized trusted platform architecture specification. Specification, Trusted Computing Group (09 2011)
38. Tierion: pymerkletools (2022)
39. Van Bulck, J., Minkin, M., Weisse, O., Genkin, D., Kasikci, B., Piessens, F., Silberstein, M., Wenisch, T.F., Yarom, Y., Strackx, R.: Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution. In: Proceedings of the 27th USENIX Security Symposium. USENIX Association (August 2018)
40. Xin, S., Zhao, Y., Li, Y.: Property-based remote attestation oriented to cloud computing. In: 2011 Seventh International Conference on Computational Intelligence and Security. pp. 1028–1032 (2011)
41. Yarom, Y., Falkner, K.: Flush+reload: A high resolution, low noise, l3 cache side-channel attack. In: Proceedings of the 23rd USENIX Conference on Security Symposium. p. 719–732. SEC'14, USENIX Association, USA (2014)
42. Zhang, T., Lee, R.B.: Cloudmonatt: An architecture for security health monitoring and attestation of virtual machines in cloud computing. In: Proceedings of the 42nd Annual International Symposium on Computer Architecture. p. 362–374. ISCA '15, Association for Computing Machinery, New York, NY, USA (2015)