Thèse de Doctorat

Université de Limoges
ED 653 – Sciences et Ingénierie
Faculté des Sciences et Techniques – Institut de Recherche XLIM

Thèse pour obtenir le grade de
Docteur de l'Université de Limoges
Informatique

Présentée et soutenue par
Thibaut JACQUES
Le 28 Novembre 2024

Protocoles d'attestation dans les environnements dynamiques

Thèse dirigée par Cristina ONETE

Jury :

Président du jury
Olivier BLAZY, Professeur – LIX – École Polytechnique

Rapporteurs
Sébastien CANARD, Professeur – LTCI – Télécom Paris
Maryline LAURENT, Professeure – Département RST – Télécom SudParis

Examinateurs
Ghada ARFAOUI, Ingénieure de recherche – Orange
Pascal LAFOURCADE, Professeur – LIMOS – Université Clermont Auvergne
Cristina ONETE, Maître de Conférences – XLIM – Université de Limoges

Remerciements

La réalisation de cette thèse a été possible grâce au soutien et à la collaboration de nombreuses personnes, que je tiens à remercier.

Tout d'abord, je souhaite exprimer ma gratitude à mon encadrante académique, Cristina Onete, pour son accompagnement, ses conseils méthodologiques, et son investissement constant dans le suivi de ce travail. Sa rigueur scientifique et son expérience ont été essentielles pour structurer et mener à bien cette recherche. Je tiens également à remercier chaleureusement Ghada Arfaoui pour son rôle central dans le suivi de la thèse au sein d'Orange. Son expertise professionnelle et ses retours précieux ont apporté une perspective complémentaire et enrichissante.

Mes remerciements s'adressent également à Marc Lacoste, pour son soutien et ses contributions, qui ont permis de faire avancer les travaux de recherche de manière significative.

Je remercie l'ensemble de mes co-auteurs et co-autrices pour leur précieuse collaboration dans la réalisation des articles inclus dans ce manuscrit. Leurs contributions scientifiques et leur expertise ont enrichi ces travaux.

Je tiens à exprimer une gratitude particulière envers Maryline Laurent et Sébastien Canard, pour le temps et l'attention consacrés à la relecture et à l'évaluation de mon manuscrit. Leurs commentaires détaillés, constructifs et pertinents ont contribué à améliorer significativement la qualité et la clarté de ce travail. Je remercie également Olivier Blazy et Pascal Lafourcade qui ont accepté de faire partie du jury de thèse.

Je remercie également toutes les personnes au sein d'Orange, ainsi qu'au sein de l'université de Limoges et du laboratoire XLIM, qui m'ont apporté un soutien logistique, et qui m'ont offert un environnement favorable tout au long de la thèse.

Enfin, j'adresse mes sincères remerciements à toutes les personnes qui, directement ou indirectement, ont contribué à la réalisation de cette thèse.

Droits d'auteurs

Contents

Introduction

INTERNET traffic has grown significantly in the last few decades and continues to do so. For example, between the end of 2020 and 2021, incoming IP traffic to the main French network operators increased by 25% [1]. This growth is mainly due to the rise of video services such as Netflix, which represented 53% of global IP traffic in 2021. In order to support this growth while simultaneously reducing network infrastructure (CAPEX) costs and the expense of operating that infrastructure (OPEX), Internet Service Providers have adopted new technologies.

Over the years, the concept of Telco Cloud has emerged as a new paradigm to address the challenges of modern networks. Telco Cloud hints at the numerous similarities between the use case in which the latter runs and those in Cloud Computing, in particular the use of virtualisation and automation. The Telco Cloud infrastructure relies on two fundamental technologies Network Function Virtualization (NFV) and Software-Defined Networking (SDN) to provide a flexible and automated network that can provide isolated virtual network services with specific characteristics on demand.

With new IT technologies comes new security and privacy threats. Network operators should consider these issues carefully, both for reputational and for legal reasons, as more countries adopt network security legislation.

Telco Cloud security is a very broad topic. In this thesis, we examine the benefits of one single technology that can achieve security in Telco Clouds, namely, remote attestation. Remote attestation is a technique that allows a verifier to ensure that a target satisfies certain properties. It relies on the use of a *trusted anchor* (which can be a hardware secure co-processor, a Trusted Execution Environment, a piece software or a hybrid of these elements) to provide evidence about the target. In the context of NFV security, for example, attestation can be used to verify that a network service has been properly deployed (*i.e.*, no adversary has tampered with the service code). Although remote attestation is not a new technique, current approaches are not suitable for the telco cloud environment.

In this thesis, we propose new remote attestation schemes that aim to render remote attestation suitable in practice for NFV usage.

Contributions

To make attestation convenient for Telco Cloud, we propose three remote attestation solutions, designed for distinct use cases. The first two protocols are tailored to be compatible with the Trusted Platform Module (TPM), a cryptoprocessor that offers a specific form of attestation and acts as the trusted anchor. The TPM is a widely

deployed chip but has limited capacity.
In our third scheme we assume that one can use any Trusted Execution Environment (TEE). As this third protocol shows, using a generic TEE can lead to better performance and easier solutions in certain use cases.
The security of each protocol is formally established through formal security models and proofs, whereas its efficiency is assessed by means of implementations and experiments

Linkable Deep Attestation. In the Telco Cloud, network functions (*e.g.*, firewalls, routers, etc.) are virtualized. Hence, verifying the state of a network function involves a verification on the state of the underlying physical infrastructure, as well as the state of the virtualized component. This process is called *Deep Attestation*. There exist two approaches for Deep Attestation. The first is called *single-channel* and allows to attest the function and the infrastructure with a single request, thus providing a strong biding between the function attestation and the infrastructure attestation. In contrast, the *multiple-channel* approach attests independently the function and the infrastructure. This method scales well for an infrastructure hosting a large number of network functions as a single infrastructure attestation is needed. The downside of the multiple-channel approach, however, is a lack of binding between the component attestation and the infrastructure attestation.
As our first contribution, in Chapter 3 we present a hybrid approach to remote attestation, which is as efficient as multiple-channel attestation, but as binding as single-channel attestation. Specifically, our scheme provides a linking mechanism, enabling to check that the attestation of a function is bound to the attestation of the infrastructure. In addition, this contribution introduced for the first time in the attestation-literature, a formalization of the security properties expected of linkable remote attestation. The security definitions are game-based and follow a modular approach, in which new properties can be cryptographically and gradually added to existing security. This work "A Cryptographic View of Deep-Attestation, or how to do Provably-Secure Layer-Linking" was presented at ACNS 2022 [2].

Multi-Tenant Attestation. Our linkable attestation scheme is a basic block for attestation in Telco Cloud, but the scheme was designed with the assumption that everything (*i.e.*, the network functions as well as the infrastructure) is owned by a single entity. However, in practice, it is often true that attestation will occur in a multi-tenancy setting. Typically, in the context of roaming (*i.e.*, connecting a device to a network which is not its native network) an operator might deploy some network functions on top the infrastructure of another operator. In this setting, privacy concerns arise as traditional attestation

constructions, by default, reveal the internal state of the target. Moreover, while the multi-channel approach scales well with numerous VNFs, it does not do well when those VNFs belong to several tenants. When there is a single verifier, the multiple approach allows to only attest the infrastructure once. Unfortunately, if each tenant acts as a verifier for the part of the infrastructure that is relevant to it, there will be as many infrastructure attestation requests as there are tenants.
As our second contribution, in Chapter 4 we show how to build on the linkable scheme to answer to this more complex use case. This new protocol uses a batching mechanism which can answer multiple attestation requests with a single attestation report. In addition our proposed scheme is privacy preserving and relies on zero-knowledge proof. This works, entitled "Towards a Privacy-preserving Attestation for Virtualized Networks" was published at ESORICS 2023 [3].

NFV Collective Attestation. Our two previous schemes focus on the attestation of a single virtual component and the physical infrastructure underlying it. However the use of a single network function does not provide a complete network service. In order to offer a meaningful service, network operators chain functions to form a network service. Such network services can be achieved through the use of several functions scattered on multiple infrastructures and managed by various tenant. Such complex infrastructures could, naïvely, be attested one VNF at a time, sequentially. The schemes we presented in Chapters 3 and 4 could provide this type of attestation quite easily.
However, as a third contribution we describe a more efficient solution, which allows to attest an entire network in a single, collective request. In our approach, the functions making up the service can collaborate to provide an attestation of the entire set in a very efficient way. This method called *collective remote attestation* was introduced in the context of attestation of swarm of Internet of Things (IoT) devices. However, in the context of multi-tenant NFV we need to take into account privacy and deep attestation (not only the function must be attested but also the infrastructure) with binding. Unlike in the two contributions presented in Chapter 3 and Chapter 4, which are tailored to the use of TPMs, this scheme relies on generic TEEs able to internally compute a zero-knowledge proof of knowledge, which will enable privacy-preserving attestation. Then by simply including the infrastructure in the collaborative attestation we can get both infrastructure and function attestation. We show that by verifying some linking information locally we get a global verification of the linking for free. The result is a very efficient protocol, which can attest a network service made up of thousands of functions in less than a second, while also greatly reducing bandwidth with respect to our naïve solutions. This work is currently in submission [4].

Personal Contributions. The three articles presented in this thesis were produced through collaborative efforts. In this paragraph, I outline my individual contributions. For the first one, entitled "A Cryptographic View of Deep-Attestation, or how to do Provably-Secure Layer-Linking", I contributed to the original concept and made the implementation. The second paper, "Towards a Privacy-preserving Attestation for Virtualized Networks", was primarily my work in terms of protocol design and implementation. I also contributed to the security model and to a lesser extent to the proofs. Similarly, the design and implementation of the final paper presented in this thesis, "Efficient and Privacy-Preserving Collective Remote Attestation for NFV", were mainly my work, with contributions to the model and proofs.

Outline

In Chapter 1 we start by describing what exactly a Telco Cloud and an NFV environment are. Then, we introduce remote attestation. Finally, we provide some cryptographic background in Chapter 2.
In Chapter 3 we introduce our linkable deep attestation protocol. This scheme only works in a single tenant setting, but it allows us to introduce a fundamental technique to provide attestation in NFV, namely, an efficient linking mechanism.
Chapter 4 builds on the authorised linked attestation scheme in Chapter 3 in order to provide attestation in a multi-tenant use case. Specifically, we consider a multi-tenant Telco Cloud where the operator hosts network functions of external entities. Each such entities must be able to attest their own network functions.
Finally, we present our third protocol in Chapter 5. This protocol considers a more complex use case where we are not attesting a single platform, but rather, multiple platforms. In this scenario, we need to be able to attest a large number of targets in a very efficient way, without generating too much control traffic in the network.

1

Telco Cloud and Remote Attestation

Contents

NETWORK operators tend to replace traditional network infrastructure with cloud-like network infrastructure to gain flexibility. These cloud-like network infrastructures are called *Telco Cloud*. Telco Cloud relies on two fundamental concepts: NFV, which consists of virtualising networks, and SDN, which allows the automation of these virtualised networks. In this thesis, we focus exclusively on NFV.

1.1 Network Function Virtualization

The basic idea of NFV is to execute network functions (*e.g.*, firewall, router, etc.) as software running on *commodity* hardware (*i.e.*, Commercial Off The Shelf (COTS) hardware) rather than having *dedicated* hardware for each function. These network functions are typically implemented as software running inside a virtualised appliance (*e.g.*, a container or a Virtual Machine (VM)). Such a package is called a Virtual Network Function (VNF) and can provide a very wide variety of network functionalities in an operator's infrastructure.

1.1.1 Architecture

The architecture of NFV is defined by the European Telecommunications Standards Institute (ETSI) [5]. It consists of four main parts, each of which consists of several functional blocks as represented in Figure 1.1:

1. **Network Function Virtualization Infrastructure (NFVI)**: The NFVI corresponds to the hardware infrastructure and resources, as well as the software required to virtualise and abstract this underlying hardware to the layer above. It creates the environment on which VNFs are deployed.
2. **VNFs domain**: It includes the VNFs as well as the Element Management (EM) which is responsible for managing one or more VNFs.
3. **NFV Management and Orchestration (MANO)**: The MANO is the management system of NFV. It consists of the following components:
    - **VNF Manager (VNFM)**: This element is responsible for performing the life-cycle operations of the virtual appliance (creation, update, termination, etc.).
    - **Virtualized Infrastructure Manager (VIM)**: The VIM is responsible for low-level management operations. It allocates the necessary resources, keeps track of the available resources, and monitors the NFVI (resource consumption, faults, etc.).

- **Orchestrator**: The orchestrator implements the network service and provides general orchestration and management of the NFV. It interfaces with both the VNFM and the VIM to delegate lower level operation.
- **Service, VNF and Infrastructure Description**: This component is a data set that contains information on the VNF deployment template, the VNF forwarding graph[1], etc. It is used by the Orchestrator to deploy and organize the VNF.

4. **Support System**: The Operations Support System (OSS) and Business Support System (BSS) are components that provide operational and business services, such as billing, product catalogues, support tickets, etc. They are linked to the orchestrator, which can take into account information from the support system to adapt the management of the service.

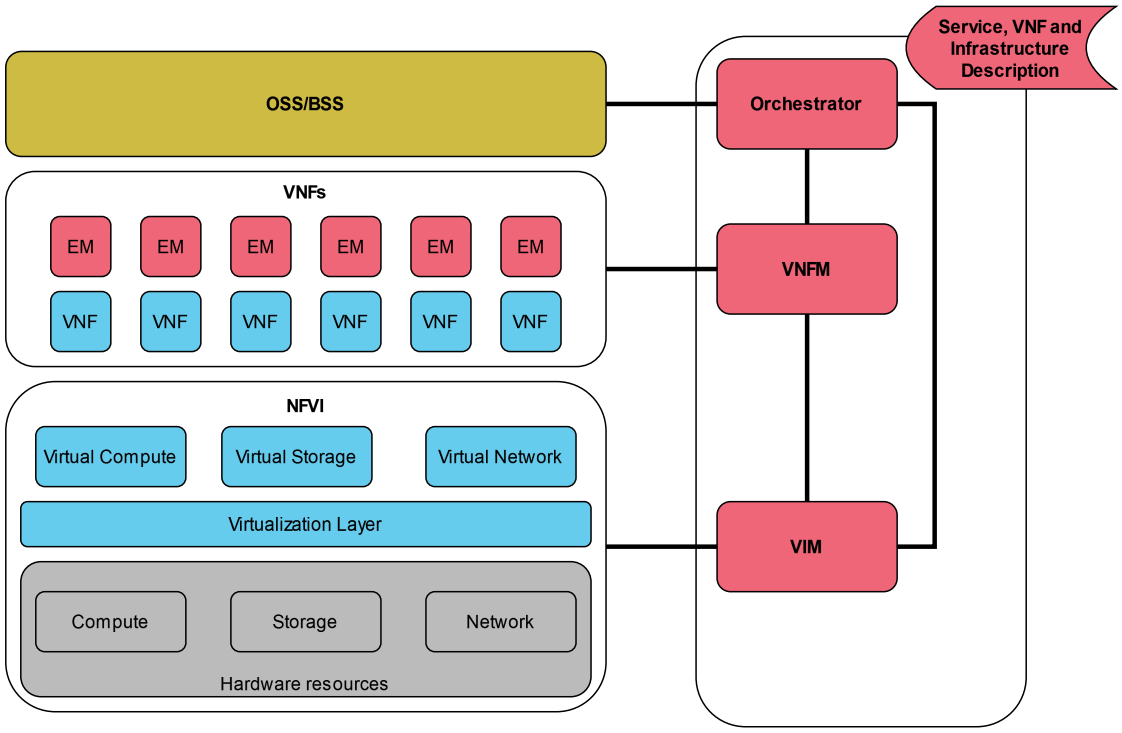


Figure 1.1: The NFV architecture as defined by the ETSI.

A simple abstraction. Although the NFV architecture is intricate, we can adopt a straightforward abstraction for the purpose of this thesis. We leave aside all the management and service components of the NFV architecture. We also consider the NFVI as a component on which the virtual appliance can run, without the details of the hardware

[1] for more information on VNF-FG, please turn to Section 1.1.2.

and virtualization layer. This leads to the architecture shown in Figure 1.2, with an infrastructure on which virtual components, possibly owned and managed by different entities, are being used. We will use the terms infrastructure, NFVI, or hypervisor interchangeably to denote the infrastructure element represented in Figure 1.2. Similarly we will use the terms VNF or VM to denote the virtual component of Figure 1.2.

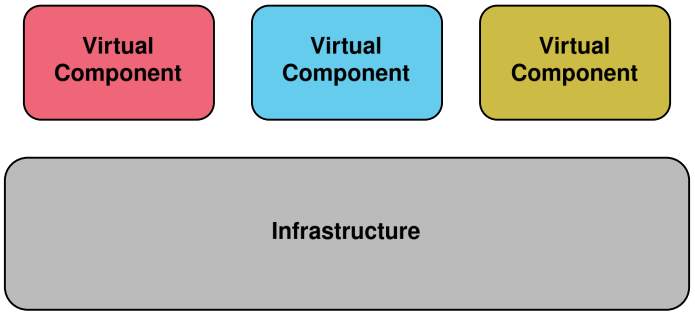


Figure 1.2: The simple architecture abstraction used in this thesis.

1.1.2 Network Services

A single network function alone does not offer much to the end customer. To provide a valuable product, a network operator needs to integrate multiple functions into an advanced network service. Examples of such services include Internet access and virtual private networks. In a traditional, static architecture, the deployment of new network services was laborious due to the necessity of manual hardware installation and configuration of connectivity. NFV simplifies the process by enabling on-demand service deployment and automatic scaling.

Note 1.1.1: SLA

Network services are often associated with a Service-Level Agreement (SLA), a document which describes the service and its expected performance. These performance expectations may relate to quality, reliability, security, etc. The SLA also defines the metrics used to determine whether the service meets the expected level of performance, as well as possible remedies and / or compensation if performance targets are not met.

From a technical point of view, a network service is a set of interconnected VNFs with an entry point (ingress) and an end point (egress). The set of VNFs between the

ingress and the egress does not necessarily define a linear path and the network flow can take different routes. Such multiple paths can exist for a variety of reasons: to provide redundancy, to avoid congestion within a single path, to process network packets differently after a filtering function, etc. This set of interconnected VNFs therefore forms a graph and is called a VNF Forwarding Graph or VNF-FG.
A VNF Forwarding Graph (VNF-FG) provides an abstraction in the form of a network service. This level of abstraction gives the operator flexibility in how it runs network services, as long as the service remains compliant with the SLA. As an example of this flexibility, the operator could for instance spawn a VNF-FG across a variety of NFVI which are not in the same geographical location. The SLA can prevent that if it imposes a strict latency requirement (a graph scattered over a large geographical distance would yield too much latency) or by requiring the NFVI to be located in a specific country (security or sovereignty concern). The operator providing the service can also use the NFVI of another operator, or also use VNFs belonging to another entity. We say that a VNF-FG can be *multi-domain* (*i.e.*, the graph spawns on top of multiple NFVI managed by different entities and potentially scattered across multiple locations) and/or *multi-tenant* (*i.e.*, the VNF are operated by different entities), as depicted in Figure 1.3.

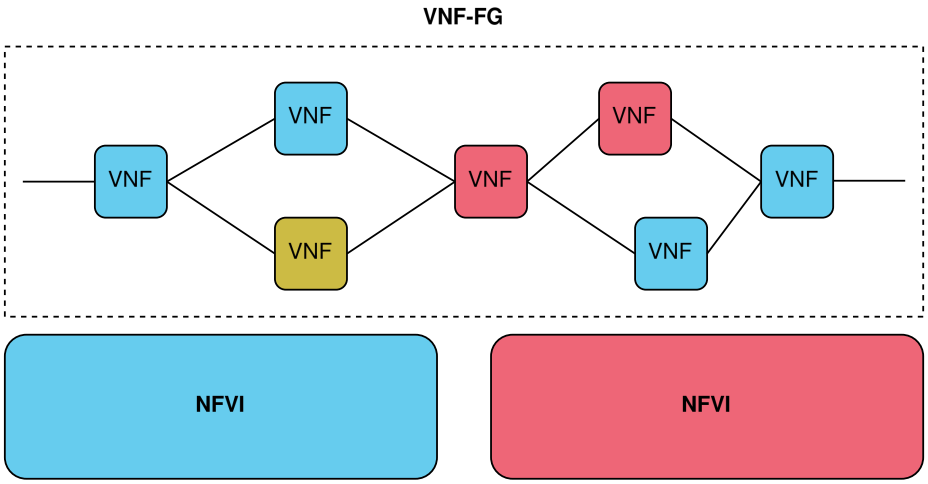


Figure 1.3: A multi-tenant and multi-domain VNF-FG.

1.1.3 Security

Moving from a static architecture, where each function corresponds to a physical appliance deployed in a known geographic location and operated by a single entity, to the Telco Cloud model come with many security and privacy challenges. Consider,

for example, the problem of ensuring that a given network function is in a valid state. In a static (hardware-based) framework, the attack surface of a potential adversary is somewhat limited. The function is implemented on dedicated hardware and in a specific location, with no way to move it. The adversary can only interact by communicating with that function. This changes in the VNF case, in which real functions can be replaced by malicious ones.
Securing an NFV infrastructure involves taking into account a large number of potential adversaries, attacks and vulnerabilities, as shown in the reports published by the European Union Agency for Cybersecurity (ENISA) [6] and ETSI [7].
Remote attestation plays a central role in achieving NFV security. It can be used not only to check that a VNF has been correctly deployed but also to continuously monitor its state. Unfortunately, existing remote attestation schemes do not comply with all the requirements of a Telco Cloud usage as we show in the next section.

1.2 Remote Attestation

Remote attestation has been identified by the ETSI as one of the fundamental security mechanisms for NFV. The concept of attestation is not new, having been introduced in the early 2000s with the Trusted Platform Module (TPM) [8]. Today, remote attestation remains an active area of research, and recent developments and trends such as confidential computing or IoT have renewed interest in the topic. Remote attestation is a broad subject and the term can encompass a wide variety of techniques and schemes. In this section, we introduce the basics of attestation and highlight some topics of particular interest to our goal of using attestation in NFV security.

1.2.1 Definition

Basically, remote attestation is a security mechanism that allows a remote verifier to check that a prover (also called an attester) satisfies certain properties. It is a passive mechanism that can only *detect* an attack, but cannot itself *prevent* a breach or heal an infected target. Attestation typically takes the form of a challenge-response protocol. The verifier sends a challenge to the machine it wants to evaluate. The prover then retrieves evidence that it has the required properties and sends it to the verifier. To reason about, or infer the state of the prover, this evidence must satisfy two requirements: (1) it must come from a trusted component called a Root of Trust (RoT); (2) it must be authenticated.

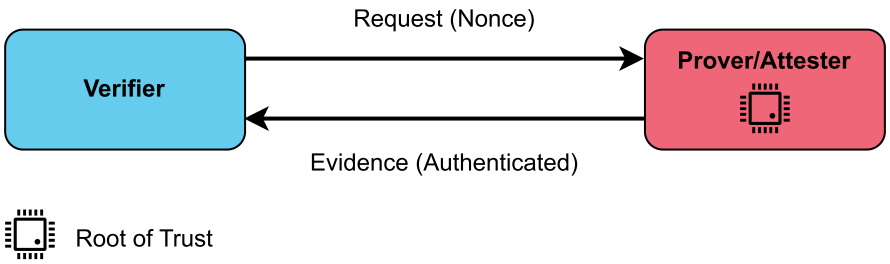


Figure 1.4: A simple view of remote attestation.

1. The prover is responsible for providing evidence of its state. If an adversary has control of the prover, it can easily change the evidence so that the adversary remains undetected and the prover appears to be in a correct state. Therefore, there must be some part of the prover system, the root of trust, which is considered trustworthy and which cannot be corrupted within the considered trust model[2]. A conceptually simple way to implement a RoT is to use an isolated secure hardware component (*i.e.*, a Trusted Execution Environment (TEE) [9]) responsible for the attestation operation. However, there are also software-based RoTs [10]–[12].
2. Obviously, in order to conclude that a piece of evidence is legitimate, the verifier must ensure that the evidence originates with the prover and, more specifically, from the prover's RoT. In addition, the provided evidence must be fresh to prevent replay attacks. Cryptographic mechanisms such as signatures and message authentication codes with secrets sealed within the RoT can ensure the authenticity of the attestation evidence.

TPM Attestation. We illustrate the principle of attestation with the concrete example of the TPM, a secure cryptoprocessor specified by the Trusted Computing Group (TCG). The TPM enables remote attestation of boot integrity and can be used as a hardware RoT. During the boot process, each component, starting from a trusted piece of code called the Core Root of Trust for Measurement (CRTM), measures the next one before loading it (e.g. BIOS/UEFI will measure the code of the bootloader and then launch the bootloader) and stores the measurements securely in the TPM. The prover can later retrieve a TPM signature of these measurements and sends the overall (measurements and signature) to the verifier. In return the verifier can check the signature to ensure the origin of the measurements and reason about the state of the prover based on the received values.

[2]Some security models only consider software adversaries, which cannot alter the RoT. Yet, most RoT can still be affected by hardware attacks.

1.2.2 Deep Attestation

The basic attestation mechanism is not suitable for virtualised environments. In fact, an infrastructure typically only has access to one RoT which can be a problem (*e.g.*, a TPM has limited memory to store measurements). To overcome this limitation, virtual RoTs (vRoTs) have been proposed [13]. For each VM, the hypervisor runs a software implementation of the RoT and transparently gives the guest access to this vRoT. Although vRoT have enabled remote attestation of the VM, they do not provide the same security guarantee as a physical RoT. To increase trust in the vRoT, it is possible to attest the hypervisor in addition to the VM. This process is called *deep attestation*. The ETSI [14] describes two classes of deep attestation schemes: Single-Channel Deep Attestation (SC-DA) and Multiple-Channel Deep Attestation (MC-DA). SC-DA provides a high level of security, but does not scale well with large numbers of virtual machines. In contrast, MC-DA scales very well, but this comes at the expense of security.

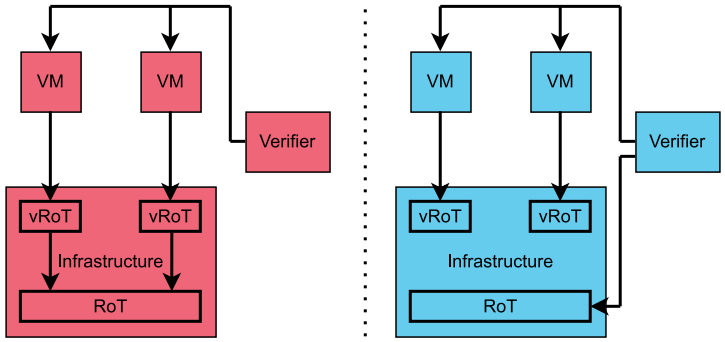


Figure 1.5: Single-channel and Multiple channel Deep Attestation.

Single-Channel. In the single channel method, for each VM, the verifier opens an attestation channel with the VM, as shown in Figure 1.5 on the left. The VM performs a nested attestation: its dedicated vRoT computes the VM attestation that will be then included in the hypervisor attestation (computed using the RoT). Since the VM and hypervisor are attested simultaneously, the verifier can reason not only about the state-validity of the VM and the hypervisor, but also about the link between the two. In other words, the attesting VM and hypervisor can prove they are running on the same physical hardware. The property of layer-linking can be extremely useful, for instance when SLA includes clauses about the geographical location of the VMs. However, it does not scale to a large number of VMs due to the need to perform an expensive hardware attestation for each VM.

Multiple-Channel. The multi-channel technique uses a separate channel to attest the

VM and the hypervisor independently, as shown on the right-hand side of Figure 1.5. Both the VM and the hypervisor can perform attestation normally. The hypervisor only needs to be attested once, while the VMs can be attested in parallel. The unique hypervisor attestation and parallelism gives much greater scalability than the single channel approach, but as VM and hypervisor attestations are fully independent, the verifier cannot reason whether the two are linked or not.

Layer-Linking. A layer-linking mechanism can be used to verify that the VM and hypervisor attestations originate from the same physical infrastructure, even when using MC-DA. The TCG proposes several alternative high-level ideas to achieve this binding for TPM and vTPM in the specifications of the Virtualized Trusted Platform [15]. Layer binding can be achieved by including the vTPM information in the hypervisor attestation. For example, during hypervisor attestation, it is possible to include a vTPM public key in the attestation nonce; a tenant will be able to verify that its VM is running on that hypervisor by checking that the VM's vTPM public key is included in the hypervisor attestation. This method is elaborated on in Chapter 3 in a broader context, not limited to TPM. Building on the work of Chapter 3, we also provide linking in the schemes presented in Chapter 4 and Chapter 5.

1.2.3 Privacy

Attestation raises concerns about the privacy of the attester. By issuing an attestation, a target reveals everything about its internal state. We can protect such information against external adversaries using traditional cryptographic techniques such as encryption, but we can only *trust* that the verifier will not use this information maliciously. However, in certain scenarios, typically a multi-tenant Telco Cloud, business competitors may need to attest their equipment to each other in order to collaborate securely, potentially revealing critical technical aspects that give them an edge over their competitor that they would prefer to keep secret. While attestation inherently reveals information about the target (at least that it is in a trusted state), Sadeghi and Stüble [16] propose a new approach to attestation called Property-Based Attestation (PBA) in which the attester has some form of control over the leakage.
In property-based attestation, instead of providing binary measurements in order to confirm their validity, targets will only prove that they guarantee some high-level properties. Therefore, instead of revealing its full configuration, the prover only proves that its configuration has specific properties. Several technical methods exist to construct such proofs. The first is to use a Trusted Third Party (TTP) [16], [17], for the verification

of a traditional attestation from the prover, and for the certification to the verifier of that prover's validity. While this approach is efficient and requires little modification to existing attestation methods, it does require the presence of an online TTP. An alternative approach to TTP-based PBA features the use of Zero-Knowledge Proofs of knowledge (ZKP) [18], [19]. In this case, the prover will be able to prove that its configuration complies with certain properties without revealing its precise configuration.

Note 1.2.1: Anonymous Attestation

During an attestation one needs to show that the attestations are produced by a valid RoT, and this involves some form of cryptographic authentication that uniquely identifies the RoT. However, authentication makes it possible to link attestations and trace a target based on its attestation reports. This problem leads to the concept of anonymous attestation which can be achieved by using a privacy CA [20] or Direct Anonymous Attestation (DAA) [21]. In this thesis, we consider another aspect of privacy: the target state/configuration privacy instead of the target/RoT identity privacy.

In Chapter 4 and Chapter 5 we will analyse use cases with multi-tenancy which require privacy-preserving attestation. We will show how to use the zero-knowledge proof approach in two different ways.

1.2.4 Collective Attestation

The privacy-preserving and/or layer-linkable attestation methods covered in the previous sections face serious scalability challenges when large numbers of VNFs require attestation. This is typically the case if an operator wants to attest a complete network service (*i.e.*, all the VNFs of the VNF-FG and the underlying infrastructure). In this case, an immediate solution is to attest each device individually (either sequentially or in parallel). However, such an approach will result in a massive flow of messages across the network, which could lead to congestion. It also requires the operator to verify thousands of attestations, which is a time-consuming task. Collective Remote Attestation (or Swarm Attestation) aims to attest large groups of elements in an efficient way. Efficiency is considered in three ways:

- **Bandwidth** : A collective attestation protocol should reduce bandwidth consumption by using some form of aggregation.
- **Proving complexity** : Attesting a group of VNF should be more efficient than

attesting each one sequentially. In other words, the time to compute an attestation should be sub-linear in the number of VNF to be certified.

- **Verification complexity** : Similar to proving, attestation verification should be sub-linear in the number of VNF in the swarm.

SEDA. Collective remote attestation (cRA) was introduced in 2015 by Asokan *et al.* [22] by means of SEDA. This scheme, like most works on cRA, focused on the attestation of IoT swarms. The basic idea of SEDA (later adopted by several solutions) is to compute a spanning tree over the IoT network and flood the attestation request into the tree (through its root) as shown in Figure 1.6. Upon receiving the request, a device retrieves its configuration, signs it and sends the result to its parent node device. The latter can verify the attestation (*i.e.*, the signature), aggregate the results of its children nodes, compute its own attestation and send the overall result (*i.e.*, its attestation and the aggregation of its child nodes attestations) to its corresponding parent node.
Since each device computes its own attestation individually and the verification is done by multiple devices, the burden is “parallelizable" and shared over the network; hence, this approach is much more efficient than a simple sequential attestation. In addition, aggregating the results reduces the amount of required bandwidth.
More precisely SEDA works in two phases: an offline and an online phase. During the offline phase devices are initialized in a trusted way by the swarm owner. Each device generates its cryptographic credentials and then establishes a symmetric keypair with each one of its neighbours. Finally neighbouring nodes exchange their expected configurations. Hence at the end of the offline phase each node has a symmetric keypair for secure communication with every neighbour and knows their expected configuration. The online phase can be triggered on demand by the verifier. To do so, the verifier can chose any node in the swarm. Then a spanning tree rooted at the chosen node is computed over the swarm. The verifier can now broadcast a request starting from the root and forwarded by each node to its child nodes. Once a node receives the request, it can attest to its parent node in a traditional manner using the keypair computed during the offline phase for secure communication. When they receive attestations from child nodes, parents nodes verify them using the expected configuration provided during the offline phase. Then, the parent produces an aggregate indicating whether the subtree rooted at this point has a correct configuration or not. Thus an attestation report in SEDA is made of the attestation of the device itself, as well as an aggregate. This process is repeated up to the root which sends the final attestation to the verifier. If both the aggregate and the root attestation are valid then the swarm is in a correct state.

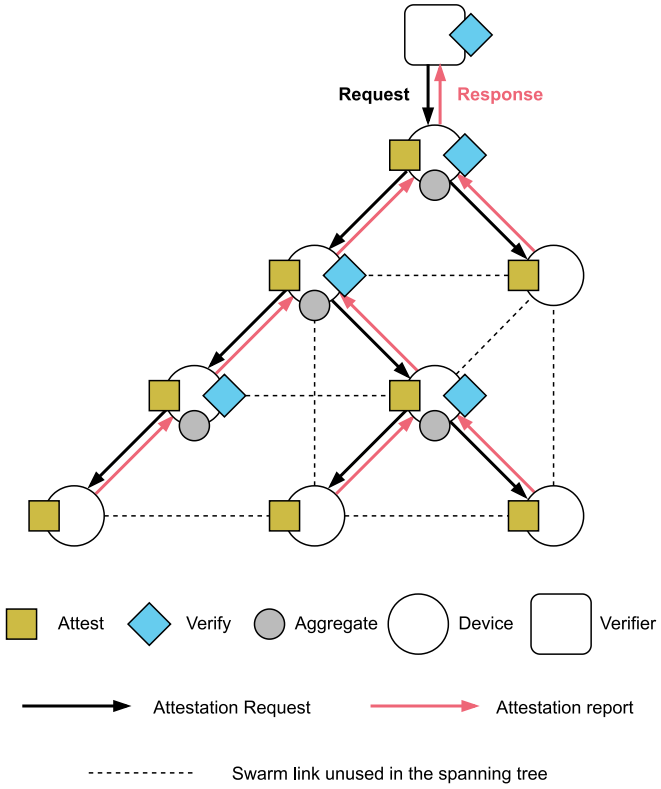


Figure 1.6: Tree based collective attestation online phase.

Tree-Based Collective Attestation. SEDA belongs to a broader category of collective attestation protocols, which we call *tree-based* attestation. Other tree-based protocols include SANA [23], which works similarly to SEDA but no longer requires the verification of the attestation and aggregate at each parent node; instead, nodes will only aggregate attestations as they are received using a custom aggregate signature scheme. With SANA, anyone in possession of the public keys and expected configurations of the swarm can check the results. This is not a strong assumption in an IoT case, where the configurations are often made public, but it can be in the use-case we are considering, that of network function virtualization.
Another attestation scheme based on spanning trees is LISA[24]. It also introduces a metric called Quality of Swarm Attestation (QoSA) which measures the information given by the attestation report (*i.e.*, SEDA is Binary-QoSA as it only give a binary result). LISA comes in two flavours which provide different levels of QoSA: a synchronous one where reports are aggregated similarly to SEDA and an asynchronous one, in which devices directly forward the result upward to parent nodes.

Highly Dynamic Networks. Interestingly, while solutions using spanning trees are effective, they are not very realistic in the use-case they consider: that of IoT swarm

attestation. This is because spanning trees require a topology of the network that is static and constant throughout the attestation process, which is not a typical scenario for dynamic IoT swarms. Alternative means of IoT-based swarm attestation exist in the literature and can handle dynamic networks, albeit at the price of a high complexity [25]–[29].

In the case we consider, that of network function virtualization and VNF-FG, the assumption of a static network is much more realistic as a VNF-FG is entirely specified by a graph descriptor. Hence, we can avoid the more complex and less efficient protocols in the literature, and adopt tree-based solutions instead. Unfortunately, existing tree based collective attestation protocols do not consider virtualized networks and are not privacy preserving. In Chapter 5 we show how to design a collective attestation scheme for a network service.

1.3 Conclusion

While promising in terms of adaptability and increased potential of providing new innovative services, Telco-Cloud architectures come with increased risks to security and privacy, as compared to traditional, hardware-based ones. Obviously, securing complex and dynamic architectures such as NFV is very challenging and many of the aspects have associated research problems. In this thesis, we single out and focus on remote attestation exclusively.

While, as described, some aspects of attestation are well established with standard implementation of industrial maturity (*e.g.*, Basic TPM Attestation), some remain still in the realm of academic work (*e.g.*, Collective Attestation). The consequence is that attestation is not yet ready to be used in the Telco Cloud environment. In particular, reconciling the three properties of scalability, privacy, and layer-linking has proved elusive. Our goal in this thesis is to bridge the gap towards achieving practical attestation for a Telco cloud. In addition, we aim to achieve security, privacy, and layer-linking in a *provable* way for the first time in the literature. Finally, in order to prove the viability of our proposals, we also report on the proof-of-concept implementations that accompany each protocol.

2

Cryptographic Preliminaries

Contents

In this chapter we review the cryptographic schemes that we will be using throughout this thesis. We also define the security properties that we require from these schemes in a formal way.

2.1 Collision-Resistant Hash Functions

A collision resistant hash function is a function that converts arbitrarily long data into a fixed size string. Because inputs are of arbitrary size and outputs are of fixed size, several different input values will be associated with the same output thus forming a collision. Collisions are inevitable, but they often represent a security risk. Collision resistant hash functions are not collision free, but they render finding collisions computationally difficult.

Definition 2.1.1 (Hash Function)**.** *An unkeyed hash function* $\mathsf{H} : \mathcal{M} \to \mathcal{B}$ *is a deterministic polynomial-time function that maps a string from an arbitrarily long message space* $\mathcal{M}$ *to a fixed-size digest space* $\mathcal{B}$*.*

Definition 2.1.2 (Collision Resistance)**.** *Let* $G_{\mathsf{CR}}(\lambda)$ *be a game in which the adversary* $\mathcal{A}$ *is given access to an unkeyed hash function* H *and must outputs two messages* m_1 *and* m_2*. It wins the game if* $\mathsf{H}(\mathsf{m}_1) = \mathsf{H}(\mathsf{m}_2)$ *and* $\mathsf{m}_1 \neq \mathsf{m}_2 \in \mathcal{M}$*. We define* $\mathsf{Adv}^{\mathsf{CR}}_{\mathsf{H}}(\mathcal{A})$ *as the probability that* $\mathcal{A}$ *wins the game. We say that a hash function* H *is collision resistant if, for all PPT* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(.)$ *such that* $\mathsf{Adv}^{\mathsf{CR}}_{\mathsf{H}}(\mathcal{A}) \leq \mathsf{negl}(\lambda)$*.*

2.2 Commitment Schemes

A commitment scheme is a cryptographic primitive that allows a party P to commit to a message m by producing a commitment string c and an opening string r. Later, P can reveal the opening r to allow to verify that the message m is associated with the commitment c.

Definition 2.2.1 (Commitment)**.** *A commitment scheme* $\mathtt{COM}$ *is a triplet of algorithms:*

$\mathtt{COM.Setup}(1^\lambda) \to \mathsf{ppar}_{\mathtt{COM}}$: The setup algorithm takes the security parameter (in unary) 1^λ as input and outputs the public parameters $\mathsf{ppar}_{\mathtt{COM}}$.

$\mathtt{COM.Commit}(m, r) \to c$: This algorithm, given a message m and a random string r, produces a commitment c.

$\mathtt{COM.Verify}(m, c, r) \to \{0, 1\}$: Given a message m, a random string r, and a commitment c, the verification algorithm outputs 1 if c is a commitment of m and 0 if it is not.

The commitment c should be *hiding* (it must not reveal anything about the committed message m) and *binding* (a commitment c will not open for any message other than m). In particular, given the opening information r, as well as c and m, anyone can check the validity of c with respect to m.

Definition 2.2.2 (Computational Binding)**.** *We say that a commitment scheme is* binding *if every PPT adversary that outputs a tuple* (c, m_1, r_1, m_2, r_2) *has at most a negligible advantage* $\mathsf{Adv}^{\mathtt{Bind}}_{\mathtt{COM}}(\mathcal{A}) = \Pr[m_1 \neq m_2 \wedge \mathtt{COM.Verify}(m_1, c, r_1) = 1 \wedge \mathtt{COM.Verify}(m_2, c, r_2) = 1]$ *to win.*

Definition 2.2.3 (Perfect Hiding)**.** *We say that a commitment scheme* COM *has perfect hiding if for all messages* $m_0, m_1 \in \mathcal{M}$*, where* $\mathcal{M}$ *is the message space, and for all* $c \in \mathcal{C}$*, where* $\mathcal{C}$ *is the commitment space it holds that :*

$$\Pr[\mathtt{COM.Commit}(m_0, r) = c] = \Pr[\mathtt{COM.Commit}(m_1, r') = c]$$

Note that with perfect hiding, even an unbounded adversary can only distinguish between m_0 and m_1 with probability $\frac{1}{2}$. This is analogous to the perfect secrecy of the one-time pad [30]. We also give a weaker computational definition, which will be useful in the next section.

Definition 2.2.4 (Computational Left-or-Right Hiding)**.** *Let* $G_{\mathsf{ComHide}}(\lambda)$ *be a game in which the challenger runs the* COM.Setup *algorithm and selects a random bit* b*. The adversary* $\mathcal{A}$ *then has access to a left-or-right commit oracle* $\mathsf{oCommit}_b(m_0, m_1)$*, which outputs* $c_b \leftarrow \mathtt{COM.Commit}(m_b, r)$ *for some random string* r *and message* m_b *depending on the value of the bit* b*. The game ends when* $\mathcal{A}$ *outputs a bit* b'*.* $\mathcal{A}$ *wins the game if* $b' = b$*. We define the advantage* $\mathsf{Adv}^{\mathtt{Hide}}_{\mathtt{COM}}(\mathcal{A})$ *as the probability that the adversary* $\mathcal{A}$ *wins the game. A commitment scheme is* hiding *if there exists a negligible function* $\mathsf{negl}(.)$ *such that* $\mathsf{Adv}^{\mathtt{Hide}}_{\mathtt{COM}}(\mathcal{A}) = \left|\Pr[\mathcal{A} \text{ wins } G_{\mathsf{ComHide}}(\lambda)] - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda)$*.*

Theorem 2.2.1. *Perfect hiding implies computational left-or-right hiding.*

Proof. Note that perfect hiding implies that given a commitment c the probability that it is the commitment of some message m is exactly the same for any message $m \in \mathcal{M}$. Therefore the outputs of $\mathsf{oCommit}_b(m_0, m_1)$ have exactly the same probability of corresponding to the commitment of m_0 or m_1. Thus, the adversary can only randomly guess b', and we have $\mathsf{Adv}^{\mathtt{Hide}}_{\mathtt{COM}}(\mathcal{A}) = 0$. □

2.3 Vector Commitments

Vector commitment schemes, introduced by Catalano and Fiore in 2011 [31], allow for a commitment to a list of values, rather than to single message. In addition, the opening of vector commitments is done by position, *i.e.*, one computes opening information for each committed value in the list, potentially separately. Unlike classical commitment schemes, vector commitments do not always have the hiding property needed for this thesis. Fortunately, Catalano and Fiore suggest that this property can be obtained by combining the vector commitment scheme with a classical commitment scheme. Instead of committing directly to the vector $v = (v_1, v_2, ..., v_n)$, one can apply a commitment scheme to each value in the vector and *then* commit $v = \big(\texttt{COM.Commit}(v_1), \texttt{COM.Commit}(v_2), ..., \texttt{COM.Commit}(v_n)\big)$, 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 will introduce a formal definition of hiding and show that the above technique yields a hiding scheme.

2.3.1 Syntax and definitions

We note that the syntax of Catalano and Fiore featured an update algorithm to both the values committed to and to the opening information. In this thesis we do not need these features and we provide a definition of vector commitment that contains only four algorithms but for which we will, however, also require a hiding property.

Definition 2.3.1 (Vector commitment)**.** *A vector commitment scheme* VCO *is made of the following algorithm :*

$\texttt{VCO.Setup}(1^\lambda, q) \rightarrow \mathsf{ppar}$: 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 $\mathcal{M}$.

$\texttt{VCO.Com}(v) \rightarrow (c, aux)$: The commitment algorithm takes as 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.

$\texttt{VCO.Open}(m, i, aux) \rightarrow \pi_i$: The opening algorithm is executed by the party that computes the commitment. Given an index $i \in \{1, \ldots, q\}$, a message m that is at some position i within a vector, and the index i itself, this algorithm gives a proof π_i that m is the i-th message of the vector v associated with the commitment c.

$\texttt{VCO.Ver}(m, c, i, \pi_i) \rightarrow \{0, 1\}$: The verification algorithm takes as 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 indicating whether the message m is assumed to be the message committed to at the i-th position of v (in this case 1) or not (0).

Definition 2.3.2 (Conciseness)**.** *A vector commitment scheme is concise if the size of the commitment is constant and independent of the length q of the vector, and the proof size is constant or $O(\log q)$.*

The position binding property says that it should not be possible to produce an opening for a given vector commitment at some possible position i such that it opens to two different messages.

Definition 2.3.3 (Position Binding)**.** *For every polynomially bounded adversary $\mathcal{A}$, the advantage* $\mathsf{Adv}^{\mathsf{VCBind}}_{\mathsf{VCO}}(\mathcal{A}) := \Pr[(c, i, m, m^*, \pi, \pi^*) \leftarrow \mathcal{A}(\mathsf{ppar}) : m \neq m^*, \mathtt{VCO.Ver}(m, c, i, \pi) = \mathtt{VCO.Ver}(m^*, c, i, \pi^*) = 1]$ *is negligible.*

The hiding property says that the a vector commitment should not reveal any information about any of the values in the vector. Note that the definition of hiding we give only holds if the two vectors submitted by the adversary are of the same size. In fact, for some schemes, the opening size depends on the size of the vector and thus reveals information about that size. Even so, we require hiding vector commitment, which must not reveal any information about the commited values.

Let $G_{\mathsf{VCHide}}(\lambda)$ be the following game :

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

Figure 2.1: The vector-commitment hiding game.

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

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

Definition 2.3.4 (Hiding)**.** *We say that a vector commitment scheme* VCO *is hiding if for every PPT $\mathcal{A}$ the advantage* $\mathsf{Adv}^{\mathsf{VCHide}}_{\mathsf{VCO}}(\mathcal{A})$ *is negligible.*

Theorem 2.3.1. *Let* COM *be a left-or-right hiding commitment scheme and* VCO *be a binding but non-hidding vector commitment scheme. Then we can construct a vector commitment scheme* VCO′ *that satisfies the biding and hiding properties. The new scheme works as follows :*

- $\underline{\texttt{VCO.Setup}'(1^\lambda, q) \rightarrow \mathsf{ppar}}$*: This algorithm computes* $\mathsf{ppar}_{\texttt{COM}} \leftarrow \texttt{COM.Setup}(1^\lambda)$ *and* $\mathsf{ppar}' \leftarrow \texttt{VCO.Setup}(1^\lambda, q)$ *and outputs* $\mathsf{ppar} \leftarrow (\mathsf{ppar}_{\texttt{COM}}, \mathsf{ppar}')$*.*
- $\underline{\texttt{VCO.Com}'(v) \rightarrow (c, aux, r)}$ *: This algorithm computes* $v' = (\texttt{COM.Commit}(v_1, r_1), \texttt{COM.Commit}(v_2, r_2)$*, ...,* $\texttt{COM.Commit}(v_n, r_n))$ *and sets* $r = (r_1, ..., r_n)$ *a vector storings the opening of each commitment. Then it computes* $(c, aux) \leftarrow \texttt{VCO.Com}(v')$ *and outputs* (c, aux, r)*.*
- $\underline{\texttt{VCO.Open}'(m, i, aux, r_i) \rightarrow \pi_i}$ *: This algorithm computes* $\pi'_i \leftarrow \texttt{VCO.Open}(m, i, aux)$ *and outputs* $\pi_i \leftarrow (r_i, \pi'_i)$*.*
- $\underline{\texttt{VCO.Ver}'(m, c, i, \pi_i) \rightarrow \{0, 1\}}$ *: This algorithm parses* π_i *as* (r_i, π'_i) *and computes* $m' \leftarrow \texttt{COM.Commit}(m, r_i)$*. Then it outputs* $b \leftarrow \texttt{VCO.Ver}'(m', c, i, \pi'_i)$*.*

Proof. (Binding) This is immediate since both VCO and VCO′ are meant to be binding vector commitment schemes. □

Proof. (Hiding) Assuming that there exists an adversary $\mathcal{A}$ which wins the $G_{\mathsf{VCHide}}(\lambda)$ game, we show that we can construct an adversary $\mathcal{B}$ which wins $G_{\mathsf{ComHide}}(\lambda)$. We let $\mathcal{B}$ simulate the $G_{\mathsf{VCHide}}(\lambda)$ game to $\mathcal{A}$. When $\mathcal{A}$ submits its two vectors v_0 and v_1 together with the set of indices $\{l_1, ..., l_m\}$ for which it wants an opening, $\mathcal{B}$ first checks whether v_0 and v_1 have a matching value for each position l. If this is not the case it aborts. Otherwise for each l it computes $c_l \leftarrow \texttt{COM.Commit}(v_{0_l}, r_l) = \texttt{COM.Commit}(v_{1_l}, r_l)$ for some random string r_l. Then, for each position k for which $\mathcal{A}$ has not requested an opening, it queries its own left-or-right commit oracle from the $G_{\mathsf{ComHide}}(\lambda)$ game $c_k \leftarrow \mathsf{oCommit}_b(v_{0_k}, v_{1_k})$. At this point $\mathcal{B}$ has a vector v of committed values of size equal to that of v_0 and v_1 where the commitment comes from its oracle $\mathsf{oCommit}_b$ or is computed by itself with the associated opening r_l. Hence it can use the vector commitment algorithms to compute the commitment of this vector and all the requested openings and pass them to $\mathcal{A}$. Let b' be the value of the bit in the $G_{\mathsf{ComHide}}(\lambda)$ game and b be the value of the bit in the $G_{\mathsf{VCHide}}(\lambda)$ game. Notice that when $b' = 0$ the view of $\mathcal{A}$ is exactly the same as in the $G_{\mathsf{VCHide}}(\lambda)$ where $b = 0$. The same is true for $b' = b = 1$. So if $\mathcal{A}$ can correctly guess the value of b, then $\mathcal{B}$ knows the value of b' which is equal to b. □

2.3.2 A possible construction

In Chapter 4 we will introduce a new property for vector commitment schemes. We will also show that vector commitment schemes based on Merkle trees guarantee this property. Therefore, we spend some time here describing Merkle trees and how they naturally provide vector commitments.

Merkle Tree. A Merkle tree is a data structure that allows messages to be aggregated into a single short value. It takes the form of a binary tree, where each leaf is a hash of a message. Each parent is computed by concatenating both child values and computing the hash of the concatenation. The root of the tree obtained in this way is the final aggregated value, which has a constant size equal to the digest size. A proof of membership of message i can be obtained by getting all nodes on the path from the i-th leaf up to the root. The verifier can then compute the hash of the message and use the nodes to recompute the root of the tree, thus ensuring that this root was calculated using the message i. We give an example in Figure 2.2. Four messages are aggregated, each stored in a leaf. The nodes in blue represent the elements that make up the proof that m1 is contained in the tree. Given m1 and the proof consisting of $H(m2)$ and $H(H(m3)|H(m4))$, a verifier, can recompute the root and check that it is the actual root.

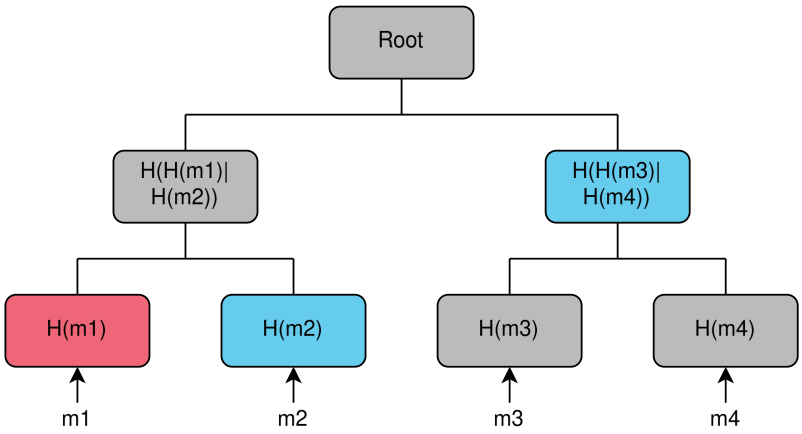


Figure 2.2: Merkle tree based vector commitment.

Vector commitment construction. It is straightforward to construct a vector commitment scheme from a Merkle tree. The commit algorithm simply computes the tree with the leaves storing the vector elements and outputs the root as the commitment. The opening for message i is the proof that element i is contained in the tree. The verification is the verification of the Merkle tree proof. The hiding property can be easily added to this construction. Instead of computing a simple hash of the message to get a

leaf of the tree, one can compute a hiding commitment of the message.

2.4 Signature Schemes

A digital signature scheme allows one to sign a document (*i.e.*, a string of characters). This signature can later be verified by anyone. A valid signature indicates that the document has not been altered and that it originated with the signer. Digital signatures are public key schemes and users must begin by generating a pair of keys: the private signing key sk and the public verification key pk.

Definition 2.4.1 (Signature)**.** *A signature scheme* $\mathtt{SIG}$ *is a tuple of four algorithms:*

$\underline{\mathtt{SIG.Setup}(1^\lambda) \rightarrow \mathsf{ppar}}$: The setup algorithm takes as input the security parameter (in unary) 1^λ and outputs public parameters ppar.

$\underline{\mathtt{SIG.KeyGen}(\mathsf{ppar}) \rightarrow (\mathsf{sk}, \mathsf{pk})}$: The key generation process produces a pair of secret and public keys $(\mathsf{sk}, \mathsf{pk})$, the former used for signing and the latter used for verifying.

$\underline{\mathtt{SIG.Sign}(\mathsf{sk}, m) \rightarrow \sigma}$: This algorithm takes as input the signing key sk, a message m and outputs a signature σ.

$\underline{\mathtt{SIG.Verify}(\mathsf{pk}, m, \sigma) \rightarrow \{0,1\}}$: During verification, given a public verification key pk, a message m and a signature σ, the algorithm returns 1 if is the signature is valid for message m and nonce nonce under pk. Otherwise, it returns 0.

A signature scheme can guarantee various degrees of security, depending on the goal (*e.g.*, to sign a specific message or any message) and the capabilities of the adversary (*e.g.*, access to a signature oracle). However, one of the most used security property is Existential Unforgeability under Chosen Message Attacks (EUF-CMA). Intuitively, EUF-CMA implies that an adversary, even with access to a signature oracle, shouldn't be able to produce a valid signature (*i.e.*, a forgery) on any new message (though prior signatures can be replayed). This is a strong definition (but not the strongest), since we are considering a "simple goal" (*i.e.*, the adversary can forge signature on any message) and a powerful adversary (*i.e.*, with access to a signing oracle).

Definition 2.4.2 (EUF-CMA)**.** *Let* $G_{\mathsf{EUF-CMA}}(\lambda)$ *be a game in which an adversary must try to forge a signature on a fresh message, while having adaptive access to a signing oracle. Next, we define* $\mathrm{Adv}^{\mathtt{EUF-CMA}}_{\mathtt{SIG}}(\mathcal{A})$ *as the probability that the adversary* $\mathcal{A}$ *wins the game. A signature scheme is* $(q_{\mathtt{SIG.Sign}}, \epsilon)$*-EUF-CMA if any adversary making at most* $q_{\mathtt{SIG.Sign}}$ *queries to the signing oracle has an advantage of at most* ϵ *to win the* $G_{\mathsf{EUF-CMA}}(\lambda)$ *game. Asymptotically, a signature scheme is EUF-CMA if this probability is negligible for all PPT* $\mathcal{A}$*.*

2.5 Zero-Knowledge Proofs of Knowledge

Zero-knowledge proofs[32] are schemes that allow one to prove that a statement is true without revealing anything other than the truth of the statement. For example, consider a system of equations; with a zero-knowledge proof, it is possible to convince a verifier that this system of equations has a solution without revealing that solution. In this example, a prover can prove more than just the existence of a solution. They can prove that they know that solution without revealing anything about it. We call this a zero-knowledge proof of knowledge. We begin this section with some definitions and informal discussion and end with a formal treatment.

Definition 2.5.1 (Language)**.** *Let f be a function which maps a finite string x into a binary value : $f : \{0,1\}^* \rightarrow \{0,1\}$. A language $\mathcal{L}_f = \{x : f(x) = 1\}$ is the set of strings x for which f outputs 1. Alternatively, we can define a language in terms of relations. Let $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$ be a relation. We define the language $\mathcal{L}_\mathcal{R} = \{x : \exists w \mid (x,w) \in \mathcal{R}\}$ as the set of all strings x for which there is a string w such that $(x,w) \in \mathcal{R}$. We call x a* statement *and w a* witness *for x.*

Definition 2.5.2 (NP Language)**.** *Let $\mathcal{R}$ be a relation and $\mathcal{L}_\mathcal{R} = \{x : \exists w \mid (x,w) \in \mathcal{R}\}$ a language defined by $\mathcal{R}$. We say that $\mathcal{L}_\mathcal{R}$ is an NP language if there exists an algorithm $\mathcal{R}_\mathcal{L}$ for deciding membership in $\mathcal{R}$ in polynomial time and a polynomial p such that $\mathcal{L}_\mathcal{R} = \{x : \exists w \mid |w| \leq p(|x|) \wedge \mathcal{R}_\mathcal{L}(x,w) = 1\}$.*

Interactive Proof System. One way of capturing the idea of a proof is to introduce two parties, a *prover* and a *verifier*. For a given statement, the prover and the verifier can interact, and at the end of the interaction, given a proof provided by the prover, the verifier should be convinced that this statement is true. Therefore, we can define an interactive proof system for a language $\mathcal{L}_\mathcal{R}$ as a pair of interactive algorithms (P,V), where V runs in polynomial time. Such a proof system should be complete (*i.e.*, every true statement has a valid proof) and sound (*i.e.*, only true statements should have a valid proof).
Note that every language $\mathcal{L}_\mathcal{R} \in$ NP has an interactive proof system. For a statement x, the prover only needs to output the witness w. The verifier can check in polynomial time if $(x,w) \in \mathcal{R}$ by checking that $\mathcal{R}_\mathcal{L}(x,w) = 1$.

Zero-knowledge. A proof system that is zero-knowledge allows a prover to convince a verifier that a statement is true without revealing anything, and in particular, a verifier shouldn't be able to learn anything more by interacting with the prover than it could on his own. We can formalize this idea by introducing an algorithm called a simulator S which, given the statement and without interacting with P, should produce an

output that is indistinguishable (statistically or computationally) from the output of a verifier V interacting with P.

Theorem 2.5.1. *Assuming there exists a hiding and binding commitment scheme, then every NP language has a zero-knowledge interactive proof system.*

This result was proven in [33]. The idea is to show that the 3-graph colouring problem has a zero-knowledge interactive proof system. Since this is an NP-complete problem, any other NP-problem can be reduced to it in polynomial time. This yields a polynomial verifier for any NP problem. The 3 graph colouring problem asks if the vertices of a graph can be coloured with a maximum of three different colours such that no adjacent vertices have the same colour. We give a very high-level view of how to prove this in zero-knowledge. The prover randomly samples 3 colours (each encoded in a number) and for each, commits to the assigned colour according to the solution of the problem. The prover sends the committed graph to the verifier. The verifier selects a random edge and sends it to the prover. Given the edge, the prover sends the verifier the openings of the commitments of the two end vertices. The verifier can now open the two vertices and check whether the colours are different. If they are not, then the verifier is sure that the prover lied; otherwise the prover might know a valid solution with low probability (*i.e.*, there is a high probability that even without a valid solution, the verifier has chosen 2 vertices with different colors). If we repeat this process (with a new random color), we increase the chance of picking an edge where the two vertices are of the same color if the prover is lying. If we repeat this a large number of times with success each time, then we have high confidence that the problem is solved.
These results have important practical consequences. Consider the problem of finding a valid assignment for an arithmetic circuit. This problem is NP, a zero-knowledge interactive proof system must exist for proving that it can be solved. However, arithmetic circuits also form a Turing-complete language. This means that we can prove, for any computation, that it gives some specific output without revealing the input. This is the basic idea behind zk-SNARK (zero-knowledge Succinct Non-interactive Argument of Knowledge), which we introduce later in this section.

Proof of knowledge and knowledge soundness. As previously stated we can consider a stronger form of soundness. Namely, that a prover not only proves that a problem has a solution, but also that it knows that solution. Hence we need to define what it means to "know" a witness w for a statement x such that $(x, w) \in \mathcal{R}$. This gives rise to a property called *knowledge soundness*. Intuitively, a protocol has knowledge soundness if there exists an algorithm called an extractor E that interacts with the prover P and

can extract the witness if the proof is valid.
The simultaneous existence of an extractor and a simulator seems contradictory. Both can exist if we allow them to rewind the protocol (*i.e.*, they can replay some part of the interactions).
Note that, in the case of the simulator, it is often implied that the verifier will follow the protocol when interacting with the simulator. This leads to a weaker notion of zero knowledge, called Honest Verifier Zero-Knowledge (HVZK). In the remainder of this thesis, however, we will only consider non-interactive proofs of knowledge for which no interaction is needed thus eliminating the need for an honest verifier.

From interactive to non-interactive. Many interactive proof systems tend to have a common structure. The prover starts by generating a committed value, and then the verifier responds with a random challenge. Finally, given the challenge, the prover generates the proof. This is the case for the 3-color graph proof. The commitment is the list of committed colours for each vertex. The challenge is an edge. The final proof is the opening for the vertices of the edge. Such a three-way move protocol is called a Sigma protocol, and there is a method called the Fiat-Shamir heuristic [34] which can be used to create non-interactive proofs from interactive ones. The idea is that the prover can create the challenge by itself and does not need to interact with the verifier. To do this, the prover uses a random oracle and queries it with all the public parameters of the proof as well as the commitment. The output of the random oracle is used in the challenge. The prover outputs the commitment, the random oracle outputs, and the proof. The verifier can thus recompute the challenge by querying the random oracle to check that the challenge is valid, and then verify the proof. Note that in this context, zero-knowledge and knowledge soundness can't be proved by rewinding. Instead, the proof relies on the programmability of the random oracle.

Non-Interactive Zero-Knowledge Proof of Knowledge. We now have all the elements we need to provide a formal definition of the tool we will use during this thesis, namely non-interactive zero-knowledge proof of knowledge.

Definition 2.5.3 (Non-Interactive Zero-Knowledge Proof of Knowledge)**.** *A* NIZK *is composed of the following algorithms:*

$\underline{\texttt{NIZK.Setup}(1^\lambda) \to (\texttt{ppar}_{\texttt{NIZK}})}$: The setup algorithm takes as input the security parameter and then outputs public parameters $\texttt{ppar}_{\texttt{NIZK}}$.

$\underline{\texttt{NIZK.Pr}(\texttt{ppar}_{\texttt{NIZK}}, x, w) \to \pi}$: The proof algorithm takes as input public parameters $\texttt{ppar}_{\texttt{NIZK}}$, a statement x, and a witness w and outputs a ZK proof π.

$\underline{\texttt{NIZK.Ver}(\texttt{ppar}_{\texttt{NIZK}}, x, \pi) \to b \in \{0,1\}}$: The verification algorithm takes in input a proof, a

statement, and the public parameters $\mathsf{ppar}_{\mathtt{NIZK}}$. It outputs 1 if the proof is accepted and 0 if it is rejected.

Definition 2.5.4 (Knowledge Soundness)**.** *For all PPT adversaries $\mathcal{A}$ there exist an extractor* $\mathsf{Ext}_{\mathcal{A}}$ *and a negligible function* $\mathsf{negl}()$ *with* $\mathsf{ppar}_{\mathtt{NIZK}} \leftarrow \mathtt{NIZK.Setup}(1^{\lambda})$ *and* $((x,\pi);w) \leftarrow \mathsf{Ext}_{\mathcal{A}}(\mathsf{ppar}_{\mathtt{NIZK}})$ *such that* $\mathsf{Adv}^{\mathsf{KS}}_{\mathsf{NIZK}}(\mathcal{A}) = \Pr[(x,w) \notin \mathcal{R} \wedge \mathtt{NIZK.Ver}(\mathsf{ppar}_{\mathtt{NIZK}}, x, \pi) = 1] = \mathsf{negl}(\lambda)$.

Definition 2.5.5 (Zero-knowledge)**.** *For all $(x,w) \in \mathcal{R}$ and for all PPT adversaries $\mathcal{A}$ there exist a simulator such that* $\mathsf{Adv}^{\mathsf{ZK}}_{\mathsf{NIZK}}(\mathcal{A}) = |\Pr[(\mathsf{ppar}_{\mathtt{NIZK}}) \leftarrow \mathtt{NIZK.Setup}(1^{\lambda}); \pi \leftarrow \mathtt{NIZK.Pr}(\mathsf{ppar}_{\mathtt{NIZK}}, x, w)] - \Pr[(\mathsf{ppar}_{\mathtt{NIZK}}) \leftarrow \mathtt{NIZK.Setup}(1^{\lambda}); \pi \leftarrow \mathsf{Sim}(\mathsf{ppar}_{\mathtt{NIZK}}, x)]| = \mathsf{negl}(\lambda)$.

Zero-Knowledge Succinct Non-Interactive Argument of Knowledge. We can define an additional requirement for NIZK, namely succinctness. Succinctness means that the proof should be small and the verification should be efficient. More precisely, the size of the proof should be sublinear in the security parameter and the verification should be linear in the statement size. Such proof systems are called Zero-Knowledge Succinct Non-interactive Argument of Knowledge (ZK-SNARK) [35]. As we said before, a powerful aspect of ZK-SNARK is that the statement aims to prove the validity of some NP statement that forms a computational model (*e.g.*, an arithmetic circuit). Thus, by encoding a program in that NP statement, it is possible to prove that one have done some computation that produces some outputs, without revealing what the associated input is. This leads to generic proof systems. However, unlike more specific proof systems (*e.g.*, proof of knowledge of a discrete logarithm), ZK-SNARK are computationally very expensive.

2.6 Authenticated Key Exchange

In most situations, when two entities want to communicate remotely, they are forced to communicate over an insecure channel (*e.g.*, the Internet). An Authenticated Key Exchange (AKE) protocol allows one to create a secure channel session from scratch over an insecure channel. Intuitively, when communicating over a secure channel, both endpoints are authenticated, and thus have a guarantee about who they are communicating with and that the messages they exchange have integrity and confidentiality protection. We will now describe a formal model for AKE, taken from the work of Jager *et al.* [36] and Krawczyk *et al.* [37]. We begin by describing one of the building blocks that we will use to define AKE security, namely the Stateful Length Hiding Authenticated Encryption scheme.

Stateful Length Hiding Authenticated Encryption. An authenticated encryption scheme with associated data (AEAD) is a scheme that can be used to protect the confidentiality and integrity of a message in a symmetric setting. Furthermore, the message can be accompanied by what is known as associated data, for which only integrity and authentication protections are provided. It consists of an encryption algorithm which takes as input a key, a message and associated data in order to produce a ciphertext, and a decryption algorithm which works in the same way but takes as output the ciphertext and returns a message. Intuitively, integrity ensures that it is not possible to modify the message or associated data, and confidentiality ensures that it is not possible to learn the contents of the message. Authenticated encryption is the basic tool to provide a secure channel. In practice, however, many AKE protocols provide more than integrity and confidentiality over the communication channel. They often also allow the order in which messages were sent to be checked and the length of the message to be hidden. Therefore, we can define Stateful Length Hiding Authenticated Encryption schemes sLHAE = (sLHAE.Init, sLHAE.Enc, sLHAE.Dec). To use sLHAE schemes, one must first initialize the states $(\mathsf{st}_e, \mathsf{st}_d) \leftarrow \mathtt{sLHAE.Init}()$. Then it can encrypt a message $(c, \mathsf{st}'_e) \leftarrow \mathtt{sLHAE.Enc}(k, l, \mathsf{ad}, m, \mathsf{st}_e)$ where l is the length of the ciphertext. The ciphertext can be decrypted later: $(m, \mathsf{st}'_d) \leftarrow \mathtt{sLHAE.Dec}(k, \mathsf{ad}, c, \mathsf{st}_d)$.

Phases. We consider two-party AKE protocols consisting of two phases. The *pre-accept* phase corresponds to the phase in which the two parties authenticate each other and establish a shared key. During the *post-accept* phase, the parties can exchange data. When one party is authenticated by the other, it is said to be in an accepting state. The post-accept phase occurs when both endpoints are in an accepting state.

Sessions. The protocol runs in *sessions* between an *instance* of one endpoint and an instance of the other. We denote the i-th instance of the party P by π^i_{P}. Each P is associated with a tuple of long-term parameters $(\mathsf{sk}, \mathsf{pk})$ and each instance keeps track of the following *attributes*:

- $\pi^i_{\mathsf{P}}.\mathsf{pid}$: this is the identifier of the communication partner which must be another party $\mathsf{Q} \in \mathbb{P} \setminus \mathsf{P}$.
- $\pi^i_{\mathsf{P}}.T$: this attribute is a transcript of all messages sent and received by instance π^i_{P} in chronological order.
- $\pi^i_{\mathsf{P}}.\rho$: where $\rho \in \{\mathtt{init}, \mathtt{resp}\}$, indicating whether party P was the initiator or the responder in session π^i_{P}.
- $\pi^i_{\mathsf{P}}.\alpha$: with $\alpha \in \{\mathtt{accept}, \mathtt{reject}, \perp\}$ indicate whether the party identified by $\pi^i_{\mathsf{P}}.\mathsf{pid}$ has been authenticated (accept) or not (reject). If authentication has not yet

taken place, then $\alpha = \perp$.

- $(\pi^i_{\mathsf{P}}.\mathsf{k}_e, \pi^i_{\mathsf{P}}.\mathsf{k}_d)$: these are the (authenticated) encryption and decryption session keys of the instance such that $(k_e, k_d) \in \mathcal{K}$ where $\mathcal{K}$ is the keyspace. They can start with the special symbol $\perp$ as a value and change later when $\pi^i_{\mathsf{P}}.\alpha = \texttt{accept}$.
- $(\pi^i_{\mathsf{P}}.u, \pi^i_{\mathsf{P}}.v)$: u and v are counters initialised to zero.
- $(\pi^i_{\mathsf{P}}.\mathsf{st}_e, \pi^i_{\mathsf{P}}.\mathsf{st}_d)$: these are the states used by the sLHAE scheme. They start with the special symbol $\perp$ and are replaced with appropriate values by the $\texttt{sLHAE.Init}$ algorithm as soon as $\pi^i_{\mathsf{P}}.\alpha = \texttt{accept}$.
- $\pi^i_{\mathsf{P}}.\mathcal{C}$: this is a list of ciphertext initially empty.
- $\pi^i_{\mathsf{P}}.b$: this is a bit randomly assigned by the challenger at the start of the game.

Definition 2.6.1 (Matching conversation)**.** *We say that π^i_{P} has a matching conversation with π^j_{Q} if $\pi^j_{\mathsf{Q}}.T$ is a prefix (*i.e.*, the first few messages of the two transcripts are the same) of $\pi^i_{\mathsf{P}}.T$ and π^i_{P} has sent the last message or if $\pi^j_{\mathsf{Q}}.T = \pi^i_{\mathsf{P}}.T$ and π^j_{Q} has sent the last message.*

Oracles. We will give the adversary access to the following oracles :

$\pi^i_{\mathsf{P}} \leftarrow \mathsf{oNewSession}(\mathsf{P}, \mathsf{Q}, \texttt{role})$: The new session oracle allows a new communication session to be instantiated between party P and party Q. The instance is set up so that $\pi^i_{\mathsf{P}}.\mathsf{pid} = \mathsf{Q}$ and $\pi^i_{\mathsf{P}}.\rho = \texttt{role} \in \{\texttt{init}, \texttt{resp}\}$.

$(m^*, \perp) \leftarrow \mathsf{oSend}(\pi^i_{\mathsf{P}}, m)$: The send oracle allows the adversary to send message m to instance π^i_{P} during the pre-accept phase. If $\pi^i_{\mathsf{P}}.\alpha = \texttt{accept}$, then the oracle outputs $\perp$ otherwise the oracle responds with the message m^* according to the protocol specification. A special message $m = \texttt{start}$ can be used to start a session.

$k \leftarrow \mathsf{oReveal}(\pi^i_{\mathsf{P}})$: This oracle reveals the session key of π^i_{P}. The key is equal to $\perp$ if $\pi^i_{\mathsf{P}}.\alpha \neq \texttt{accept}$.

$\mathsf{sk} \leftarrow \mathsf{oCorrupt}(\mathsf{P})$: With this oracle, the adversary can gain access to the long-term secret key sk_j of party P. We say that party P is τ-corrupted if $\mathsf{oCorrupt}(\mathsf{P})$ is the τ-th query of an adversary $\mathcal{A}$.

$(c_b, \perp) \leftarrow \mathsf{oEncrypt}(\pi^i_{\mathsf{P}}, m_0, m_1, l, \mathsf{ad})$: This oracle replaces oSend during the post-accept phase. If $\pi^i_{\mathsf{P}}.\alpha \neq \texttt{accept}$ it outputs $\perp$. Otherwise, because the instance is in an accept state it means that $\pi^i_{\mathsf{P}}.\mathsf{st}_e \neq \perp$. Hence, the oracle can encrypt one of the two messages $(c_b, \mathsf{st}') \leftarrow \texttt{sLHAE.Enc}(\pi^i_{\mathsf{P}}.k_e, l, \mathsf{ad}, m_b, \pi^i_{\mathsf{P}}.\mathsf{st}_e)$ depending on $\pi^i_{\mathsf{P}}.b$. The oracle also increments the counter u by one, updates the state of the instance $\pi^i_{\mathsf{P}}.\mathsf{st}_e \leftarrow \mathsf{st}'$ and appends c_b to $\pi^i_{\mathsf{P}}.\mathcal{C}$. Finally it outputs c_b.

$(m, \perp) \leftarrow \mathsf{oDecrypt}(\pi^i_{\mathsf{P}}, c, \mathsf{ad})$: During the post-accept phase, the decryption oracle allows to decrypt ciphertext c. If $\pi^i_{\mathsf{P}}.\alpha \neq \texttt{accept}$ or $\pi^i_{\mathsf{P}}.b = 0$ it outputs $\perp$. Otherwise,

it runs $(m, \mathsf{st}') \leftarrow \mathtt{sLHAE.Dec}(\pi^i_{\mathsf{P}}.k_d, \mathsf{ad}, c, \pi^i_{\mathsf{P}}.\mathsf{st}_d)$. It increments v, updates $\pi^i_{\mathsf{P}}.\mathsf{st}_d \leftarrow \mathsf{st}'$ and outputs m.

Security game. We define ACCE security, introduced by Jager *et al.* [36] by means of a game. Let $G_{\mathsf{ACCE}}(\lambda, \mathsf{N_P})$ be a game between a challenger $\mathcal{G}$ and an adversary $\mathcal{A}$. We denote $\mathsf{N_P} = |\mathbb{P}|$. The challenger starts by initializing each party $\mathsf{P} \in \mathbb{P}$ with its long-term key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$. The adversary is then given access to all the public keys and to the oracles oNewSession, oSend, oReveal, oCorrupt, oEncrypt, and oDecrypt as described above.
From this game we define two advantages:

1. $\mathsf{Adv}^{\mathtt{ACCE-Auth}}_{\mathtt{AKE}}(\mathcal{A})$ is the probability that there is an instance π^i_{P} such that :
   - $\pi^i_{\mathsf{P}}.\mathsf{pid} = \mathsf{Q}$ and $\pi^i_{\mathsf{P}}.\alpha = \mathtt{accept}$ such that π^i_{P} switches to an accepting state during $\mathcal{A}$ τ_0-th query.
   - Q is τ-corrupted and $\tau_0 < \tau$
   - $\mathcal{A}$ did not query $\mathsf{oReveal}(\pi^j_{\mathsf{Q}})$ with $\pi^j_{\mathsf{Q}}.\alpha = \mathtt{accept}$ and having matching conversation with $\pi^i_{\mathsf{P}}.\mathsf{pid}$.
   - there is no instanceπ^j_{Q} with π^i_{P} having a matching conversation to π^j_{Q}.
2. The second advantage is defined as the probability that $\mathcal{A}$ produces a triplet (P, i, b') such that : $\mathsf{Adv}^{\mathtt{ACCE-SC}}_{\mathtt{AKE}}(\mathcal{A}) = |\Pr[\pi^i_{\mathsf{P}}.b = b'] - \frac{1}{2}|$ with $b' \neq \perp$ if the following conditions hold:
   - $\pi^i_{\mathsf{P}}.\mathsf{pid} = \mathsf{Q}$ and $\pi^i_{\mathsf{P}}.\alpha = \mathtt{accept}$ such that π^i_{P} switches to an accepting state during $\mathcal{A}$ τ_0-th query.
   - Q is τ-corrupted and $\tau_0 < \tau$
   - $\mathcal{A}$ did not query $\mathsf{oReveal}(\pi^i_{\mathsf{P}})$ or $\mathsf{oReveal}(\pi^j_{\mathsf{Q}})$ with π^i_{P} having a matching conversation to π^j_{Q}.

Definition 2.6.2. *We say that an* AKE *scheme is ACCE-secure if the exists a negligible function* $\mathsf{negl}()$ *such that* $\mathsf{Adv}^{\mathtt{ACCE-Auth}}_{\mathtt{AKE}}(\mathcal{A}) \leq \mathsf{negl}(\lambda)$ *and* $\mathsf{Adv}^{\mathtt{ACCE-SC}}_{\mathtt{AKE}}(\mathcal{A}) \leq \mathsf{negl}(\lambda)$.

A simple syntax. The syntax and construction of AKE schemes can be quite complex. In this thesis we will only consider such schemes as high-level black boxes. Thus, we will use a very simple syntax consisting of a key generation algorithm $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathtt{AKE.KGen}(1^\lambda)$ that generates long-term credentials, a secure channel establishment algorithm $(k, \perp) \leftarrow \mathtt{AKE.Sce}(\mathsf{P}, \mathsf{Q})$ that corresponds to the pre-accept phase (*i.e.*, mutual authentication and establishment of shared secret) described above, and an encryption $c \leftarrow \mathtt{AKE.Enc}(k, m)$ and decryption $m \leftarrow \mathtt{AKE.Dec}(k, c)$ algorithm for the post-accept phase.

2.7 Conclusion

In this chapter we have briefly introduced the cryptographic schemes we will be using throughout this thesis. We focused on the syntax and formal definitions of security. There are many ways of implementing these schemes, but the constructions we propose in this work are independent of any particular implementation.
Hash functions and signatures are the basis of all attestation schemes for authenticating measurements. Vector commitment schemes will be used to handle multiple attestation requests in one go, while zero-knowledge proofs will be used to make attestation more privacy-friendly. Finally, AKE will enable us to transfer attestations confidentially and only to authorized verifiers.

3

Deep Attestation and Layer-Linking

Contents

DEEP Attestation is the fundamental tool for providing a fully-fledged attestation service dedicated to the telco cloud. In this chapter, we introduce an efficient and scalable deep attestation scheme with strong security properties. This work focuses on the use case of a TPM Rot of Trust, but the idea can be extended to any RoT. This work was published at ACNS 2022 [2].

3.1 Introduction

In Section 1.2.2 we introduced deep attestation as a means of attesting all layers of a virtualised infrastructure. The idea is to virtualise the root of trust with a software implementation managed by the hypervisor. Each virtual component is then given access to a specific virtual RoT so that it can undergo remote attestation. However, we also want to verify that a VM and a hypervisor that are attested are actually associated (*i.e.*, the VM is running on top of the hypervisor). In Section 1.2.2, we also described two common approaches to providing layer association: single-channel deep attestation and multi-channel deep attestation. In short, the single channel approach consists of attesting the hypervisor through the VM attestation, typically the VM will request a hypervisor attestation embedding its own attestation report. On the other hand, with multi-channel attestation, the VM and hypervisor are attested completely independently. The multi-channel approach provides a very effective method of attesting an infrastructure with many VMs running on a hypervisor, but with no means of verifying the link. In contrast, single-channel attestation provides strong binding, but scales poorly because each VM attestation implies a hypervisor attestation. When using inefficient RoTs such as a TPM, single channel deep attestation becomes unusable in practice.

3.1.1 Towards authorized linked attestation

We take a middle path between single and multi-channel deep attestation to obtain layer-binding between VMs and hypervisors with reasonable efficiency. Our solution is simple, yet elegant, using standard cryptography to ensure that a hypervisor's single attestation is linkable to any number of VMs managed by it.
Cryptographically, we see this as a new primitive, which we call Authorised Linked Attestation, built in steps from increasingly stronger primitives. Each of these intermediate steps plays a dual role: on one hand, it formalises security guarantees that are of independent interest for attestation; on the other hand, it provides an intuition of the guarantees that specific cryptographic primitives can help to achieve.

The first and fundamental step in our construction is *basic attestation*. This primitive simply states whether a component is compromised or not. In our security model, this property is assumed to be guaranteed by default when demanding an attestation from a RoT. Next, authenticated attestation builds on basic attestation by associating parties with identities. The attestation must not only indicate whether the party is compromised, but also authenticate the component. We have therefore added a cryptographic component to basic attestation, which is sufficient to provide the basic functionality required for multi-channel attestation.
One step further, the linked-attestation primitive, built from authenticated attestation, will allow two different components to (a) attest their own states; (b) provide auxiliary material that makes two separate attestations linkable. While this primitive has no direct parallel in real-world attestation, we use it as a convenient way to split the security proof of our final result into two: linked-attestation will focus on proving the fact that two attestations can be securely linked; whereas authorised linked-attestation models attestation as a protocol, using fresh randomness and a secure channel with an honest attestation server.
Finally, we add a new party to the system: the attestation server, which acts as a verifier. We then combine the linked attestation primitive with a unilaterally authenticated authenticated key exchange protocol, which authenticates the attestation server and allows the attestation itself to remain confidential with respect to a Person-in-the-Middle (PitM) adversary.

3.1.2 Our contributions

Our work on deep attestation consists of three main contributions.

A cryptographic scheme. Our scheme ensures secure and efficient linked DA. The hypervisor and VMs each attest only once. However, we also embed a list of public keys (associated with the VMs managed by the hypervisor) within the hypervisor attestation, which is established by the root of trust. To authenticate the forwarded keys, they are integrated into the attestation nonce provided by the attestation server. If the hypervisor's attestation verifies, then the attestation server can link that hypervisor with the (subsequently attesting VMs) which use keys in the forwarded list. If the hypervisor's attestation fails, then the public keys cannot be trusted.

Provably secure authorized linked attestation. An important advantage of our approach is that we have a fully-formalized provable-security guarantee. We use a composition-based approach, constructing primitives that are increasingly stronger

out of weaker ones. Our goal is to ultimately obtain authorized linked attestation : a primitive which allows components to individually attest (to an authorized entity), and to have their attestations linked.
Authorized linked attestation will have three properties: *authorization* (only an authorized server can query an attestation report); *indistinguishability* (no Person-in-the-Middle adversary can know even one bit of a report exchanged during a legitimate protocol with probability significantly higher than $\frac{1}{2}$); and *linkability* (an attestation server can detect if two components are not linked).
At the basis of our construction is a yea-or-nay basic attestation scheme, which is "secure" by assumption. Its functionality is simple: the basic attestation scheme outputs a faulty attestation whenever a component is compromised, and a correct one for honest components. In other words, this basic attestation scheme is a compromise-oracle: when queried it (indirectly) produces a proof of whether a component has been tampered with or not.
Based on this assumption, we build a sequence of cryptographic mechanisms that add security against stronger adversaries. A first step is to build authenticated attestation: a scheme which allows us to authenticate the component that provides the attestation, and additionally ensures that this component's attestations always verify prior to corruption, but fail to verify as soon as compromise occurs. We can think of authenticated attestation as the minimum provided (and required) by multi-channel attestation.
Then, we consider linked attestation: a scheme that introduces the hypervisor-VM relationship described above, and permits not only the verification of individual attestations, but also (publicly) linking attestations. The final scheme is authorized linked attestation which introduces an authorized server that is the only party able to access an attestation report and which is responsible for linking attestation together.

Implementation. We provide an implementation of our authorized linked attestation scheme and create a test environment to measure the performance of the scheme. We used a regular laptop equipped with a TPM 2.0 (as a root of trust) to create this environment. We set up an architecture with one hypervisor and multiple VMs. The VMs use full virtual TPMs as a virtual root of trust. We made over 100 experiments. This showed that our solution is more efficient than the single channel approach and adds a negligible computational burden (a hash function computation) compared to traditional multi-channel DA.

3.2 Basic Attestation

We start with the basic attestation primitive BA, which is secure by assumption in our model. We discuss this assumption in more detail in Section 3.2.3. The idea of this basic building block is to capture the fact that a measurement process should be secure and be able to collect evidence that reflects the current state of the target of the attestation. Thus, if the target is compromised, this primitive will always produce an attestation that does not verify, and vice versa, for an honest party, only attestations that verify will be produced. In basic attestation, the attestation report is simply the evidence. To denote an evidence that is not valid, we use the symbol $\mathfrak{e}$. Every time basic attestation is run on a compromised component, it will return this symbol $\mathfrak{e}$. On the other hand, a valid evidence is denoted by a classic attestation report rpt.

3.2.1 Syntax

With this primitive BA we consider a single party P associated with a single bit attribute γ that indicates whether the party is honest or has been compromised. This attribute is originally set to 0 to indicate that the party is honest and it is flipped to 1 once the party has been compromised. If $\mathsf{P}.\gamma$ ever takes the value 1 it can never going back to 0. Additionally, as stated, a compromised party attestation yields the symbol $\mathfrak{e}$, while an honest one yields rpt. The syntax of BA is as follow:

$\underline{\mathtt{BA.Setup}(1^\lambda) \rightarrow \mathsf{ppar}}$: On input the security parameter λ this algorithm outputs public parameters ppar.

$\underline{\mathtt{BA.Attest}(\mathsf{ppar}) \rightarrow \mathsf{rpt}}$: The attest algorithm, given public parameter ppar, outputs an attestation report rpt such that $\mathsf{rpt} \neq \mathfrak{e}$ if $\mathsf{P}.\gamma = 0$. Otherwise, if $\mathsf{P}.\gamma = 1$ it outputs a special symbol $\mathfrak{e}$.

$\underline{\mathtt{BA.Verify}(\mathsf{ppar}, (\mathsf{rpt}, \mathfrak{e})) \rightarrow \{0,1\}}$: This algorithm takes as inputs public parameters ppar and either an attestation report rpt or an error symbol $\mathfrak{e}$.By convention, an output of 0 means the attestation fails, while if the output is 1, the attestation succeeds. We require, by construction, that for all ppar : $\mathtt{BA.Verify}(., \mathfrak{e}) = 0$.

Correctness. We demand correctness: when a non-$\mathfrak{e}$ report is generated, the latter will automatically verify. Our basic attestation component thus becomes the minimal non-cryptographic assumption that we need to make to prove our scheme secure

Prover P		**Verifier**
Setup phase: $\texttt{BA.Setup}(1^\lambda) \rightarrow \text{ppar}$		
$\texttt{BA.Attest}(\text{ppar}) \rightarrow \text{rpt}$	$\xrightarrow{\text{rpt}}$	$\texttt{BA.Verify}(\text{ppar}, \text{rpt})$ $\rightarrow 0$ if P compromised (P.$\gamma = 1$) $\rightarrow 1$ if P uncompromised (P.$\gamma = 0$)

Figure 3.1: Basic attestation description with an honestly-generated target. Notice that there is no authentication involved.

3.2.2 Security

While our primitive is secure by assumption, we can formally define its security. To do so, we first define the following oracle :

$\underline{\texttt{oBA.Attest}() \rightarrow (\text{rpt} \cup \mathfrak{e})}$: This oracles calls the $\texttt{BA.Attest}$ algorithm for the party P and returns the result to the adversary $\mathcal{A}$. The challenger $\mathcal{G}$ stores the result in a database DB.

Then we consider the following game:

$$\begin{array}{l}\underline{\text{Game } G_{\text{BA}}(\lambda)} \\ \text{ppar} \leftarrow \texttt{BA.Setup}(1^\lambda) \\ \text{P}.\gamma \leftarrow 1 \\ \underline{\mathcal{A}^{\texttt{oBA.Attest}()}(1^\lambda, \text{ppar})} \\ \mathcal{A} \text{ wins iff. } \exists\ \text{rpt} \in \text{DB} \mid \texttt{BA.Verify}(\text{ppar}, \text{rpt}) = 1\end{array}$$

Figure 3.2: Basic attestation game.

In this game, the challenger sets the compromised bit γ for party P to 1, indicating that P is compromised. The adversary can query an oracle to get attestations of this party. The challenger records the outcomes in DB. The adversary wins if there is a report in DB that is verified correctly. We denote $\text{Adv}^{\text{SEC}}_{\text{BA}}(\mathcal{A})$ as the probability of adversary $\mathcal{A}$ winning the game. In our construction, we assume that no adversary can fool a RoT into thinking a compromised component is in fact intact except with negligible probability.

3.2.3 Discussion

One may wonder at this point what our purpose might be in constructing a security model for a primitive that is by definition correct and secure. We need that security model in our reductions: we will use the attestation primitive to build stronger, linked attestation, and then we will want to make the argument that if an attacker can break the larger primitive, it will also break the smaller primitive. As the smaller primitive is secure by design, this is not possible, and hence the larger primitive is also secure. In

practice, the gap will consist of attackers that are able to fool the RoT into establishing a valid attestation report for a compromised component.

3.3 Authenticated Attestation

Basic attestation acts as a foolproof way of telling whether a device is compromised or not. However, the security it provides is very weak. For one thing, it has no authentication guarantees, so potentially one could use an attestation report that was honestly generated for an honest component to attest a compromised one. Another problem that is more subtle concerns the way components are compromised. Because the basic reports described in the previous section have no timestamp, nor specific freshness, we cannot take into account adaptive tampering. In the security game, the party generating the attestation report is either honest or compromised from the beginning. Yet, ideally we would like a primitive that ensures that a party can start out as honest (and all the reports generated at that time verify as correct), and later be compromised (and all the reports generated after that moment will fail). We can do this by deploying cryptographic solutions (namely a signature along with a nonce).

3.3.1 Syntax and Construction

The new primitive authenticated attestation will consider an environment containing up to N parties. Similarly to the BA scheme, each party keeps track of the compromise bit γ. However, parties know a pair of public and secret keys denoted, for each party P, P.pk (the public key) and P.sk (the secret key).
We construct an authenticated attestation scheme ANA out of basic authentication BA., a large set of nonces $\mathcal{N} \leftarrow \{0,1\}^{\ell}$ (with ℓ chosen as a function of the security parameter λ), and an EUF-CMA-secure signature scheme SIG = (SIG.Setup, SIG.KeyGen, SIG.Sign, SIG.Verify).

ANA.Setup$(1^{\lambda}) \rightarrow$ ppar : This algorithm run BA.Setup(1^{λ}) a number N times, outputting $\mathsf{ppar}_1, \mathsf{ppar}_2, ..., \mathsf{ppar}_{\mathsf{N}}$. Each time ppar_i is created, a handle P_i for the party is also created (it will be the party associated with the instance of BA.Attest run for those parameters). It sets ppar $\leftarrow (\mathsf{ppar}_1, \mathsf{ppar}_2, ..., \mathsf{ppar}_{\mathsf{N}}, \mathsf{N})$, and outputs this value.

ANA.KeyGen$(\mathsf{P}_i) \rightarrow (\mathsf{P}_i.\mathsf{sk}, \mathsf{P}_i.\mathsf{pk})$: The key generation algorithm keeps a counter count (starting from 0), which indicates how many times this algorithm has been run. If at the time this algorithm is queried count $<$ N , then it runs SIG.KeyGen as a

black box and outputs the resulting $(\mathsf{sk}, \mathsf{pk})$ keys. It sets $\mathsf{P}_i.\mathsf{sk} \leftarrow \mathsf{sk}$ and $\mathsf{P}_i.\mathsf{pk} \leftarrow \mathsf{pk}$. Party P_i is then initialized with these keys.

$\mathtt{ANA.Attest}(\mathsf{ppar}, \mathsf{P.sk}, \mathsf{nonce}) \rightarrow \mathsf{authRpt} \cup \mathfrak{e}$: On input the public parameters ppar, a private key P.sk of a party P (which has already been registered), and a value $\mathsf{nonce} \xleftarrow{\$} \mathcal{N}$, this algorithm first runs $\mathsf{rpt} \leftarrow \mathtt{BA.Attest}(\mathsf{ppar})$, then the algorithm signs $\sigma \leftarrow \mathtt{SIG.Sign}(\mathsf{P.sk}, \mathsf{rpt}|\mathsf{nonce})$, that is, it signs a concatenation of the nonce and the obtained attestation report. The output of this algorithm is $\mathsf{authRpt} \leftarrow (\mathsf{rpt}, \sigma)$. If the required party or key does not exist, the value $\mathfrak{e}$ is output by default. If $\mathsf{rpt} = \mathfrak{e}$, then we instantiate $\mathsf{authRpt} = \mathfrak{e}$.

$\mathtt{ANA.Verify}(\mathsf{ppar}, \mathsf{P.pk}, \mathsf{nonce}, (\mathsf{authRpt} \cup \mathfrak{e})) \rightarrow \{0, 1\}$: On input public parameters ppar, a public key P.pk of a party P, an auxiliary value $\mathsf{nonce} \in \mathcal{N}$, this algorithm first checks if the last input is $\mathfrak{e}$; if so, the algorithm outputs 0 by default. Else, the algorithm parses $\mathsf{authRpt} = (\mathsf{rpt}, \sigma)$ (with $\mathsf{rpt} \neq \mathfrak{e}$ by construction), then runs $b \leftarrow \mathtt{SIG.Verify}(\mathsf{P.pk}, \mathsf{rpt}, \sigma)$ and $d \leftarrow \mathtt{BA.Verify}(\mathsf{ppar}, \mathsf{rpt})$. The algorithm outputs $b \wedge d$ as its response. Notably, 1 is output if, and only if, the signature and the basic attestation verify concomitantly.

Prover P		**Verifier**
Setup phase: $\mathtt{ANA.Setup}(1^\lambda) \rightarrow \mathsf{ppar}$		
$\mathtt{ANA.KeyGen} \rightarrow (\mathsf{P.pk}, \mathsf{P.sk})$		
$\mathtt{ANA.Attest}(\mathsf{ppar}, \mathsf{P.sk}, \mathsf{aux}) \rightarrow (\mathsf{rpt}, \sigma)$	$\xrightarrow{\mathsf{authRpt}=(\mathsf{rpt},\sigma)}$	$\mathtt{ANA.Verify}(\mathsf{ppar}, \mathsf{P.pk}, \mathsf{aux}, (\mathsf{rpt}, \sigma))$ $\rightarrow 0$ if P compromised ($\mathsf{authRpt} = \mathfrak{e}$ or σ invalid) $\rightarrow 1$ if P uncompromised ($\mathsf{authRpt} \neq \mathfrak{e}$ and σ valid)

Figure 3.3: Authenticated attestation built upon basic attestation.

3.3.2 Security

We formalise the security of the authenticated attestation primitive and show that our construction actually guarantees requirements assuming that the cryptographic secret and computation cannot be tampered with, and that there exists a secure basic attestation scheme BA (*i.e.*, the RoT is secured).

Intuitively, the security we require for this primitive will be that a valid authenticated attestation report for a party P and fresh auxiliary information (used as nonce) is hard to forge by an adversary which knows all the public information, can register and compromise users, and query an attestation oracle that returns a valid attestation report or $\mathfrak{e}$. In particular, in a secure scheme, verification should fail if either the authentication or the attestation fails. In other words we want our primitive to guarantee that the evidence is fresh and comes from a legitimate authenticated RoT.

To provide a formal definition of a secure authenticated attestation scheme we first define the following oracles :

$\underline{\texttt{oANA.Reg}() \rightarrow (\mathsf{P}_i, \mathsf{P}_i.\mathsf{pk})}$: If $i \leq \mathsf{N}$, it runs $(\mathsf{P}_i.\mathsf{sk}, \mathsf{P}_i.\mathsf{pk}) \leftarrow \texttt{ANA.KeyGen}(\mathsf{P}_i)$. It outputs $\mathsf{P}_i.\mathsf{pk}$ to all parties and keeps $\mathsf{P}_i.\mathsf{sk}$ private, stored in the key attribute of party P_i. A handle for this party is also returned to $\mathcal{A}$.

$\underline{\texttt{oANA.Attest}(\mathsf{P}_i, \mathsf{aux}) \rightarrow \mathsf{authRpt} \cup \mathfrak{e}}$: This oracle runs $\texttt{ANA.Attest}(\mathsf{ppar}, \mathsf{P}_i.\mathsf{sk}, \mathsf{aux})$ and returns the outputs. On adversarially chosen input values P_i and aux, the oracle updates a list $\mathcal{L}_{\mathsf{rpt}} \leftarrow \mathcal{L}_{\mathsf{rpt}} \cup (\mathsf{P}_i, \mathsf{aux}, \mathsf{authRpt})$.

$\underline{\texttt{oANA.Compromise}(\mathsf{P}_i) \rightarrow \mathsf{OK}}$: This oracle allows an adversary to compromise party P_i, thus changing $\mathsf{P}_i.\gamma$ to 1.

$\underline{\texttt{oANA.Auth}(\mathsf{P}_i, \mathsf{m}) \rightarrow \sigma_{\mathsf{m}}}$: This oracle can only be queried for a party whose compromise bit is 1, and it outputs an EUF-CMA-secure signature keyed with $\mathsf{P}_i.\mathsf{sk}$ on a message m. We require that m be outside the range of any basic attestation scheme. This last oracle reflects the fact that compromised parties can access a signing function within the TPM.

We consider the following game :

$\underline{\text{Game } G_{\mathsf{AN}}(\lambda)}$
$\mathsf{ppar} \leftarrow \texttt{ANA.Setup}(1^\lambda)$
$\mathsf{N} \leftarrow 1$
$\underline{(\mathsf{P}, \mathsf{aux}, \mathsf{authRpt}) \leftarrow \mathcal{A}^{\texttt{oANA.Reg},\texttt{oANA.Attest},\texttt{oANA.Compromise},\texttt{oANA.Auth}}(1^\lambda, \mathsf{ppar})}$
$\mathcal{A}$ wins iff. $\texttt{ANA.Verify}(\mathsf{ppar}, \mathsf{P.pk}, \mathsf{aux}, \mathsf{authRpt}) = 1 \wedge (\mathsf{P}, \mathsf{aux}, \mathsf{authRpt}) \notin \mathcal{L}_{\mathsf{rpt}}$

Figure 3.4: Authenticated attestation game.

Definition 3.3.1 (Secure Authenticated Attestation)**.** *We say that the authenticated attestation scheme* ANA *is secure if for all PPT* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(.)$ *such that :* $\Pr[\mathcal{A} \text{ wins } G_{\mathsf{AN}}(\lambda)] \leq \mathsf{negl}(\lambda)$*.*

Theorem 3.3.1 (Secure Authenticated Attestation)**.** *The* ANA *scheme is secure assuming that (1) the* BA *scheme is secure (2) the size of* $\mathcal{N}$ *is large and (3) the* SIG *signature scheme is EUF-CMA secure. More formally, if the exist an adversary* $\mathcal{A}$ *that breaks the authenticated attestation game* $G_{\mathsf{AN}}(\lambda)$ *with advantage* $\mathsf{Adv}^{\mathtt{SEC}}_{\mathtt{ANA}}(\mathcal{A})$ *then there exists adversaries* $\mathcal{B}_1$ *and* $\mathcal{B}_2$ *such that :* $\mathsf{Adv}^{\mathtt{SEC}}_{\mathtt{ANA}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathtt{SEC}}_{\mathtt{BA}}(\mathcal{B}_1) + \frac{\mathsf{Adv}^{\mathtt{EUF-CMA}}_{\mathtt{SIG}}(\mathcal{B}_2)}{\mathsf{N}}$ *where* N *is the number of parties* P*.*

Proof. We give a proof by sequence of games :

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

$\mathbb{G}_1$: This game is defined as the previous one except that the challenger aborts the game if a compromised component is able to generate a valid attestation report. Suppose that there exists an adversary $\mathcal{B}_1$ that has a non-negligible advantage $\mathsf{Adv}^{\mathtt{SEC}}_{\mathtt{BA}}(\mathcal{A})$ of winning the basic attestation game $G_{\mathtt{BA}}(\lambda)$. This means that there exists a party P such that $\mathsf{P}.\gamma = 1$ (*i.e.*, P is compromised) but $\mathsf{rpt} \in \mathsf{DB}$ with $\mathtt{BA.Verify}(\mathsf{ppar}, \mathsf{rpt}) = 1$ (note that $\mathsf{rpt} = \mathfrak{e}$ potentially). By the difference lemma we have : $|\Pr[\mathcal{A} \text{ wins } \mathbb{G}_0] - \Pr[\mathcal{A} \text{ wins } \mathbb{G}_1]| \leq \mathsf{Adv}^{\mathtt{SEC}}_{\mathtt{BA}}(\mathcal{A})$, for $\mathcal{B}_1$ the adversary having the maximal advantage to break basic attestation.

$\mathbb{G}_2$: This game is defined as the previous one except that the game aborts if $\mathcal{A}$ can generate a valid signature. We show that : $|\Pr[\mathcal{A} \text{ wins } \mathbb{G}_1] - \Pr[\mathcal{A} \text{ wins } \mathbb{G}_2]| = \frac{\mathsf{Adv}^{\mathtt{EUF-CMA}}_{\mathtt{SIG}}(\mathcal{B}_2)}{\mathsf{N}}$ where $\mathsf{Adv}^{\mathtt{EUF-CMA}}_{\mathtt{SIG}}(\mathcal{B}_2)$ if the advantage in the EUF-CMA game and N the number of parties. The proof is done by reduction. Assume that $\mathcal{A}$ can generate a valid $\mathsf{authRpt}^*$ (*i.e.*, $\mathtt{ANA.Verify}(\mathsf{ppar}, \mathsf{P}^*.\mathsf{pk}, \mathsf{aux}^*, \mathsf{authRpt}^*) = 1$) with $(\mathsf{P}^*.\mathsf{pk}, \mathsf{aux}^*, \mathsf{authRpt}^*) \notin \mathcal{L}_{\mathsf{rpt}}$. We then show that there exists adversary $\mathcal{B}_2$ using $\mathcal{A}$ as a sub-routine with non-negligible advantage of winning the EUF-CMA game. Adversary $\mathcal{B}_2$ simulates the game of $\mathcal{A}$ thus acting as the challenger in the $G_{\mathsf{AN}}(\lambda)$. The behaviour of $\mathcal{B}_2$ is defined as follows :

- receive pk from its own challenger of the EUF-CMA game.
- run the $\mathtt{ANA.Setup}$ algorithm to get ppar (and also N).
- randomly select $i^* \xleftarrow{\$} \{1, ..., \mathsf{N}\}$. Two cases need to be studied depending on the i-th query of $\mathcal{A}$:
  - **case 1 :** $i \neq i^*$. When $\mathcal{A}$ calls oracle $\mathtt{oANA.Reg}()$ then $\mathcal{B}_2$ runs $\mathtt{ANA.KeyGen}(\mathsf{P}_i)$ to retrieve $(\mathsf{P}_i.\mathsf{sk}, \mathsf{P}_i.\mathsf{pk})$. $\mathcal{B}_2$ sends back $\mathsf{P}_i.\mathsf{pk}$ to $\mathcal{A}$. When $\mathcal{A}$ calls oracle $\mathtt{oANA.Attest}()$ then $\mathcal{B}_2$ runs algorithm $\mathtt{ANA.Attest}$ (which is possible since $\mathcal{B}_2$ has the corresponding secret key). $\mathcal{B}_2$ sends back to $\mathcal{A}$ the output of the algorithm. Note that in this case, the simulation is the same as the original game since $\mathcal{B}_2$ uses the same algorithm of the oracles.
  - **case 2 :** $i = i^*$. In this case, $\mathcal{B}_2$ will inject its own material to use it in its EUF-CMA game. When $\mathcal{A}$ calls oracle $\mathtt{oANA.Reg}(\mathsf{P}_{i^*})$ then $\mathcal{B}_2$ simply return pk (the one of the EUF-CMA game). Note that in this case, $\mathcal{B}_2$ does not have access to sk. Thus when $\mathcal{A}$ calls oracle $\mathtt{oANA.Attest}()$, $\mathcal{B}_2$ cannot sign the report. Instead, $\mathcal{B}_2$ runs $\mathtt{BA.Attest}(\mathsf{ppar}_{i^*})$ and send to its challenger the report it get as an answer concatenated with the nonce $\mathsf{rpt}|\mathsf{aux}$. In response, its challenger will send a signed value of it using sk (as it is possible in the EUF-CMA game) . $\mathcal{B}_2$ then forward this to $\mathcal{A}$. The view of A that is different from the original game in this

case is the output of the $\texttt{oANA.Auth}$ oracle. The latter runs algorithm $\texttt{ANA.Attest}$ which generate an attestation report $\mathsf{rpt} \leftarrow \texttt{BA.Attest}(\mathsf{ppar}_{i^*})$ and a signature $\sigma \leftarrow \texttt{SIG.Sign}(\mathsf{P}_i.\mathsf{sk}, (\mathsf{rpt}, \mathsf{aux}))$. In the simulation, $\mathcal{B}_2$ has access to algorithm $\texttt{BA.Attest}$ but the signature scheme is different. Yet, both signature schemes are EUF-CMA thus their outputs are indistinguishable (meaning that $\mathcal{A}$ cannot decide from which schemes the output comes from with non-negligible probability) since the keys have the same probability distribution. Hence, the simulation of the game by $\mathcal{B}_2$ and the real game are indistinguishable.

- When $\mathcal{A}$ return its forgery on query i, $\mathcal{B}_2$ parses $\mathsf{authRpt}_i$ as $(\mathsf{m}^*|\mathsf{aux}^*, \sigma^*)$.
- Finally $\mathcal{B}_2$ returns $(\mathsf{m}^*|\mathsf{aux}^*, \sigma^*)$ to its challenger and wins if $\mathcal{A}$ forges the i^* query (meaning that $i = i^*$).

By combining the results, we have : $\mathsf{Adv}^{\texttt{SEC}}_{\texttt{ANA}}(\mathcal{A}) = \Pr[\mathcal{A} \text{ wins } \mathbb{G}_0] \leq \mathsf{Adv}^{\texttt{SEC}}_{\texttt{BA}}(\mathcal{B}_1) + \frac{\mathsf{Adv}^{\texttt{EUF}-\texttt{CMA}}_{\texttt{SIG}}(\mathcal{B}_2)}{\mathsf{N}}$ which is negligible. □

3.4 Linked Attestation

Authenticated attestation allows the attestation of one (out of many) components, based on that component's unique secret key. If we define now parties as being either VMs or hypervisors, the notion of authenticated attestation suffices to capture the basic guarantees of multi-channel deep-attestation. However, recall that our goal is to ultimately allow parties to link their attestations (a hypervisor's attestation should, *e.g.*, be linkable to that of a number of VMs also hosted on that platform).
In this section we describe our next primitive: linked attestation. The latter takes place in an environment where several parties are registered in a linked way. This corresponds to a single platform. A first step is platform registration, by which several parties are linked on the same underlying hardware. Each entity later generates a linkable attestation, verifiable on its own and linkable with other linkable attestation reports.

3.4.1 Syntax

We now consider multiple parties as before but which can now additionally belong to various types of components (*e.g.*, VMs or hypervisors). Although our application scenario is that of linking VM and hypervisor attestations, we make our framework more generic than that. Instead of just two types of components, we consider linkable sets $\mathcal{S}_1$, $\mathcal{S}_2$, ..., $\mathcal{S}_L$, which resemble equivalence classes. These sets are defined such that

any party in one set (say $\mathsf{P}_{\mathcal{S}_1}$) can produce an attestation that is linked to attestations produced by parties in sets $\mathcal{S}_2$, ..., $\mathcal{S}_L$. We write $\mathsf{P} \diamond \mathsf{Q}$ to say that two parties are linked. The relation is reflexive ($\mathsf{P} \diamond \mathsf{P}$), symmetric (if $\mathsf{P} \diamond \mathsf{Q}$, then $\mathsf{Q} \diamond \mathsf{P}$), and transitive (if $\mathsf{P} \diamond \mathsf{Q}$ and $\mathsf{Q} \diamond \mathsf{R}$, then $\mathsf{P} \diamond \mathsf{R}$). An intuitive depiction of these sets appears in Figure 3.5.

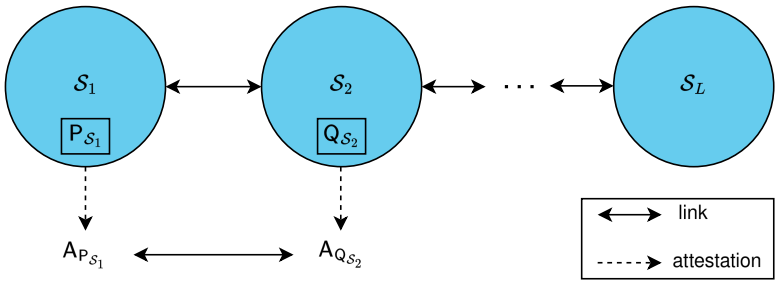


Figure 3.5: Linkable sets.

We formalize a linked-attestation scheme $\mathtt{LKA}$ as a tuple of algorithms $\mathtt{LKA} = (\mathtt{LKA.Setup}, \mathtt{LKA.Reg}, \mathtt{LKA.Attest}, \mathtt{LKA.Verify}, \mathtt{LKA.Link})$, defined for some auxiliary set $\mathcal{AUX}$. By convention, we allow the use of $\emptyset$ to indicate that any of the input or output (sub)sets might be empty.

$\mathtt{LKA.Setup}(1^\lambda) \rightarrow \mathsf{ppar}$: On input the security parameter λ, this algorithm outputs public parameters ppar. This security parameter includes the maximal number of allowed disjoint linkable sets, which we denote as L.

$\mathtt{LKA.Reg}(s_1, s_2, ..., s_L) \rightarrow \{(\mathsf{SK}_1, \mathsf{PK}_1), ..., (\mathsf{SK}_L, \mathsf{PK}_L)\}$: This algorithm keeps as state a number L of sets $\mathcal{S}_i$ originally set to $\emptyset$, and a vector of sets of public keys $\mathcal{PK}$ (also initialized to $\emptyset$). On input a number of subsets s_i $(i = 1, 2, ..., L)$, this algorithm first checks that $\forall i, j, s_i \cap \mathcal{S}_j = \emptyset$ (else the algorithm outputs $\perp$). If the relation is true, then the algorithm generates for each party $\mathsf{P}_j \in s_i$, (forall j, i) a tuple of public and private keys $\mathsf{P}_j.\mathsf{sk}, \mathsf{P}_j.\mathsf{pk}$, initializing P_j with those keys. We require the uniqueness of all the generated public keys. The subsets s_i are each added to supersets $\mathcal{S}_i$. The algorithm groups the keys of all parties $\mathsf{P}_j \in s_i$ in a pair of private/public key subsets: $(\mathsf{SK}_i, \mathsf{PK}_i)$, updating the i-th component $\mathcal{PK}[i]$ of $\mathcal{PK}$ as $\mathcal{PK}[i] \cup \mathsf{PK}_i$. All parties are given access to the public-key subsets (and more generally, to $\mathcal{PK}$).

$\mathtt{LKA.Attest}(\mathsf{ppar}, \mathcal{PK}, \mathsf{P.sk}, (s_1, ..., s_L), \mathsf{aux}) \rightarrow (\mathsf{lkRpt} \cup \mathfrak{e}, \mathsf{lkaux})$: on input public parameters ppar, the current set of public keys $\mathcal{PK}$, the private key $\mathsf{P.sk}$ of some party P, subsets $s_i \in \mathcal{S}_i$, and an auxiliary value $\mathsf{aux} \in \mathcal{AUX}$, this algorithm outputs either a linked report lkRpt or a special failure symbol $\mathfrak{e}$, and a different value lkaux (this last entry could be used to store linkage-related information).

$\underline{\texttt{LKA.Verify}(\mathsf{ppar}, \mathsf{P.pk}, (\mathsf{lkRpt} \cup \mathfrak{e}), \mathsf{aux}, \mathsf{lkaux}) \rightarrow \{0,1\}}$: On input the public parameters ppar, a public key P.pk, a linked report (or a failure symbol), as well as auxiliary values aux and lkaux, this algorithm outputs a verification bit. By convention, 0 means failure and 1 means acceptance of the attestation

$\underline{\texttt{LKA.Link}(\mathsf{ppar}, \mathcal{PK}, \Pi_1, ..., \Pi_L) \rightarrow \{0,1\}}$: On input the public parameters ppar, the set of public keys $\mathcal{PK}$, and subsets Π_i containing elements of the form $(\mathsf{P}_j.\mathsf{pk}, \mathsf{aux}, (\mathsf{lkRpt} \cup \mathfrak{e}), \mathsf{lkaux})$, this algorithm outputs 1 if the reports in all of the indicated subsets can be linked (thus also indicating the parties are linked) or 0 otherwise.

The setup algorithm outputs public parameters ppar, including the maximal number L of sets considered for linking. One can register platforms including subsets of components of each type: this algorithm generates keys for each party. A linked attestation algorithm produces a linked report lkRpt and an auxiliary linking value lkaux. Finally, the verification algorithm checks the attestation in each individual linked attestation report lkRpt and the linking algorithm outputs 1 if several linked attestations seem to belong to the same registered platform, and 0 otherwise. The global idea of the linked attestation scheme is also depicted in Figure 3.6.

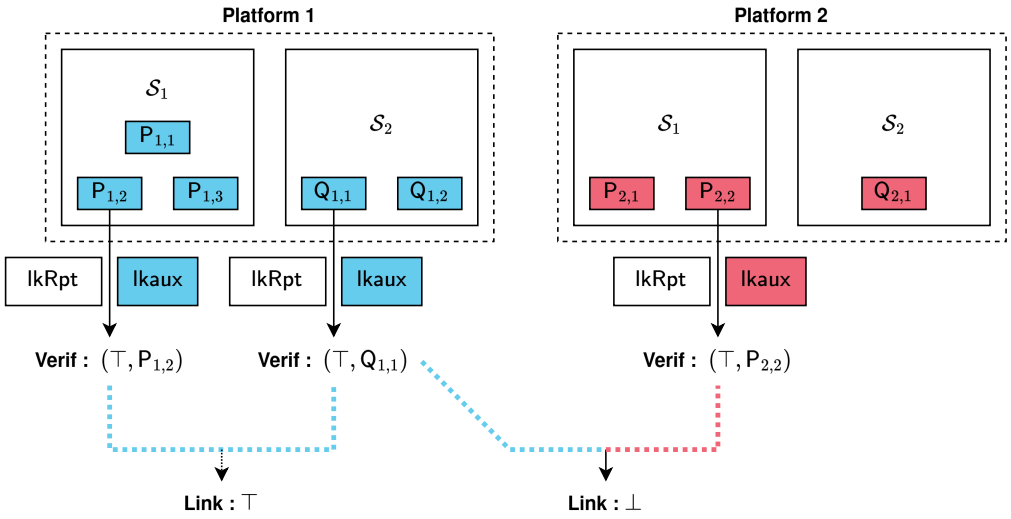


Figure 3.6: Linked attestation primitive. The dashed line indicates a platform under the same registration. In this example, both platforms are composed of two subsets (namely $\mathcal{S}_1$ and $\mathcal{S}_2$). There are a total of three attestation report verifications ($\mathsf{P}_{1,2}$, $\mathsf{Q}_{1,1}$, $\mathsf{P}_{2,2}$). The link verification outputs true when the devices are registered under the same platform and false otherwise

Correctness. The LKA scheme is built upon the ANA scheme. There are two types of component to consider, VM and hypervisor. When a component is registered on a platform, its public key is appended in a list (PK_1 for VMs, and PK_2 for the hypervisor). The public key of a VM is appended to the report in LKA.Attest and can be retrieved by

the hypervisor. The latter can link the attestation to a public key via algorithm LKA.Link. We consider two cases to verify the correctness (1) a VM (not compromised) is not registered on the platform, and (2) a component (VM or hypervisor) is compromised. For (1) the attestation will be correct since the component is not compromised, but the linking process will abort since the public key does not belong to PK_1. For (2) if a VM (or the hypervisor) is compromised then the attestation will fail since the authenticated attestation is supposed to be correct (the ANA.Attest algorithm is executed to generate the report).

3.4.2 Security

The security of linked attestation informally states that an adversary, which has Person-in-the-Middle capabilities and can compromise devices at will, cannot make it appear that two devices are linked when they are not, in fact, so.
A significant limitation on the adversary's capabilities is that compromising a device will not leak its private keys (which are assumed to be held by a TPM). However, the adversary will gain limited oracle access to those keys upon compromising the device. The limitations to those queries follow rules of access to an actual TPM.
As usual we define the security of the primitive by means of a game which in addition to a security parameter λ will also be parameterized by a set of functions F. We call those functions *permitted key-access functions*. The adversaries wins if it able to make attestation stored in $\mathcal{L}_{\mathsf{rpt}}$ for parties registered in different platforms link. However, at this point the adversary is constrained to a change-one-change-all kind of game: it cannot, for instance, append an lkaux component of its choice to an honestly-generated lkRpt, nor vice-versa. In the security game, the adversary registers platforms and can compromise some of their components. When a component is compromised, the adversary gets oracle access to a set of permitted functions of the component's private key. As a result, the impact of the security proof depends on the function space F. The more functions the adversary is able to query once it compromises a component, the more security our primitive is able to provide. However, note that we cannot give the adversary access to some functions, such as the identity function on the component's private key.
More formally we define the following oracles :

$\underline{\mathsf{oLKA.Reg}(n_1, ..., n_L) \rightarrow (\mathsf{PK}_1, ..., \mathsf{PK}_L)}$: The linked user-registration oracle creates a linked platform consisting of n_i components of the type indicated by $\mathcal{S}_i$. The challenger first instantiates a counter $\mathsf{N}_i = 0$ for all i; it also instantiates subsets s_i as a tuple of n_i handles $\mathsf{P}_{i,j}$, with $\mathsf{N}_i + 1 \leq j \leq \mathsf{N}_i + n_i$ and then runs the algorithm $\mathtt{LKA.Reg}(s_1, s_2, ..., s_L)$, instantiating the parties with their keys and outputting the

public keysets to the adversary. The subset consisting of the list of subsets is added to $\mathcal{L}_{\mathsf{Reg}}$. We note that this way of registering parties ensures by construction that no party finds itself in multiple sets, nor on multiple platforms.

oLKA.HAttest$(\mathsf{P}, (s_1, ..., s_L), \mathsf{aux}) \rightarrow (\mathsf{lkRpt} \cup \mathfrak{e}, \mathsf{lkaux}^*)$: This oracle first verifies that $\mathsf{P}.\gamma = 1$. If the condition is false (P is compromised), then this oracle outputs an error symbol $\perp$ (compromised parties must use the oracle oLKA.CAttest described below). If the condition is true, then this algorithm runs LKA.Attest$(\mathsf{ppar}, \mathcal{PK}, \mathsf{P.sk}, (s_1, ..., s_L), \mathsf{aux})$, and returns the output to the adversary. The tuple $(\mathsf{P}, (s_1, ..., s_L), \mathsf{aux}, \mathsf{lkRpt}, \mathsf{aux}^*)$ is stored in $\mathcal{L}_{\mathsf{rpt}}$.

oLKA.Compromise() : This oracle allows an adversary to compromise party P_i, thus changing $\mathsf{P}_i.\gamma$ to 1.

oLKA.CAttest$(\mathsf{P}, (s_1, ..., s_L), \mathsf{aux}, f) \rightarrow (\mathfrak{e}, \mathsf{aux}^*)$: This oracle first verifies that $\mathsf{P}.\gamma = 1$ (else $\perp$ is returned). If the condition holds, then this oracle first checks that $f \in F$ and if so, it runs f on P.sk and inputs aux to output aux^* (*i.e.*, $\mathsf{aux}^* \leftarrow f(\mathsf{P.sk}, \mathsf{aux})$). Then it runs oLKA.HAttest$(\mathsf{P}, (s_1, ..., s_L), \mathsf{aux})$ to obtain lkRpt (the second output is discarded). Note that by the security of the linked attestation primitive, we will have that $\mathsf{lkRpt} = \mathfrak{e}$. The tuple $(\mathsf{P}, (s_1, ..., s_L), \mathsf{aux}, \mathsf{lkRpt}, \mathsf{aux}^*)$ is added to $\mathcal{L}_{\mathsf{rpt}}$ and $(\mathsf{lkRpt}, \mathsf{aux}^*)$ is returned to $\mathcal{A}$.

We now consider a game $G_{\mathsf{LK}}(\lambda, F)$ between adversary $\mathcal{A}$ and challenger $\mathcal{G}$. The challenger begins by running LKA.Setup(1^λ), returning ppar to the adversary, and then it instantiates two lists: a list of parties $\mathcal{L}_{\mathsf{Reg}} \leftarrow \emptyset$ and a list of linkable attestations $\mathcal{L}_{\mathsf{rpt}} \leftarrow \emptyset$. The adversary then plays the game by using the oLKA.Reg, oLKA.HAttest, oLKA.Compromise and oLKA.CAttest oracles adaptively.

At the end of its interaction, A outputs a party P and a tuple of subsets $(\tilde{s_1}, ..., \tilde{s_L})$ with an index i^* such that $\forall i \neq i^*, s_i \leftarrow \tilde{s_i}$ and $s_{i^*} \leftarrow \tilde{s_{i^*}} \cup \{\mathsf{P}\}$. In addition the adversary outputs for every party $\mathsf{P} \in s_1 \cup \cdots \cup s_L$ (parties being indexed as $\mathsf{P}_{i,j}$) a tuple $(\mathsf{aux}, \mathsf{lkRpt}, \mathsf{aux}^*)$ such that $(., ., \mathsf{aux}, \mathsf{lkRpt}, \mathsf{aux}^*) \in \mathcal{L}_{\mathsf{rpt}}$. We note this completes output out.

We say the adversary wins if all the following conditions hold simultaneously:

- cond1 : For each s_i the parties inside this set are all registered *i.e.*, they were output by oLKA.Reg. In addition P is registered.
- cond2 : There exists at least one party $\mathsf{Q} \in s_j$ such that P and Q were issued from different oLKA.Reg queries.
- cond3 : By setting $\Pi_i \leftarrow (\mathsf{P.pk}, \mathsf{aux}, (\mathsf{lkRpt} \cup \mathfrak{e}), \mathsf{aux}^*)$ and, for $k \neq i$, for all $\mathsf{P}_{k,j} \in s_j$, $\Pi_k \leftarrow (\mathsf{P}_{k,j}.\mathsf{pk}, \mathsf{aux}, (\mathsf{lkRpt}k, j \cup \mathfrak{e}), \mathsf{aux}^*_{k,j})$, it holds that:
  LKA.Link$(\mathsf{ppar}, \mathcal{PK}, \Pi_1, ..., \Pi_L) = 1$.

In other words, the adversary wins if it is able to make attestations stored in $\mathcal{L}_{\mathsf{Reg}}$ for parties registered on different platforms (P and Q) link. Note that there are two ways that an attestation can end up in $\mathcal{L}_{\mathsf{Reg}}$: either it is issued for an honest component (and then it should hold that $\mathsf{lkRpt} \neq \mathfrak{e}$), or it is issued for a compromised party, for an adversarially-chosen evaluation of a permitted function f on a secret key (in which case $\mathsf{lkRpt} = \mathfrak{e}$). In other words, at this point a compromised component cannot just bind the output aux^* from the function evaluation oracle with a different report. The adversary will gain this ability at the next step (when the freshness aux will no longer be chosen by the adversary).

$$\frac{\text{Game } G_{\mathsf{LK}}(\lambda, F)}{\begin{array}{l}\mathsf{ppar} \leftarrow \mathtt{LKA.Setup}(1^\lambda) \\ \mathcal{L}_{\mathsf{Reg}} \leftarrow \emptyset, \mathcal{L}_{\mathsf{rpt}} \leftarrow \emptyset \\ \mathsf{out} \leftarrow \mathcal{A}^{\mathtt{oLKA.Reg},\mathtt{oLKA.HAttest},\mathtt{oLKA.Compromise},\mathtt{oLKA.CAttest}}(1^\lambda, \mathsf{ppar})\end{array}}{\mathcal{A} \text{ wins iff. } \mathsf{cond1} \wedge \mathsf{cond2} \wedge \mathsf{cond3}}$$

Figure 3.7: Linked attestation game.

Definition 3.4.1 (Secure Linked Attestation)**.** *We say that a linked attestations scheme* LKA *is secured if for all PPT $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(.)$ *such that :* $\Pr[\mathcal{A} \textit{ wins } G_{\mathsf{LK}}(\lambda, F)] \leq \mathsf{negl}(\lambda)$*.*

3.4.3 Construction

We provide a construction for platforms that have two types of components: virtual machines and their managing hypervisor. Thus, in our instantiation, $L = 2$. We use an authenticated attestation scheme (ANA.Setup, ANA.KeyGen, ANA.Attest, ANA.Verify) as a black box. The basic construction is depicted in Figure 3.8. During setup, our linked-attestation scheme first runs ANA.Setup and outputs ppar and $L = 2$. Note that by construction ANA.Setup must output a number N, denoting the maximal number of parties that can be set up. This counter will represent a global maximum to parties of all types that will exist in our security model.
Following setup, one can register a subset of VMs together with a hypervisor. The algorithm runs the key-generation algorithm ANA.KeyGen of the underlying authenticated attestation scheme for each party, independently (note that this also ensures that the total number of parties remains at most N). Finally, keys are grouped by types of parties: keys of VMs are output in a set of public keys PK_1 and the key of the hypervisor is output as PK_2.

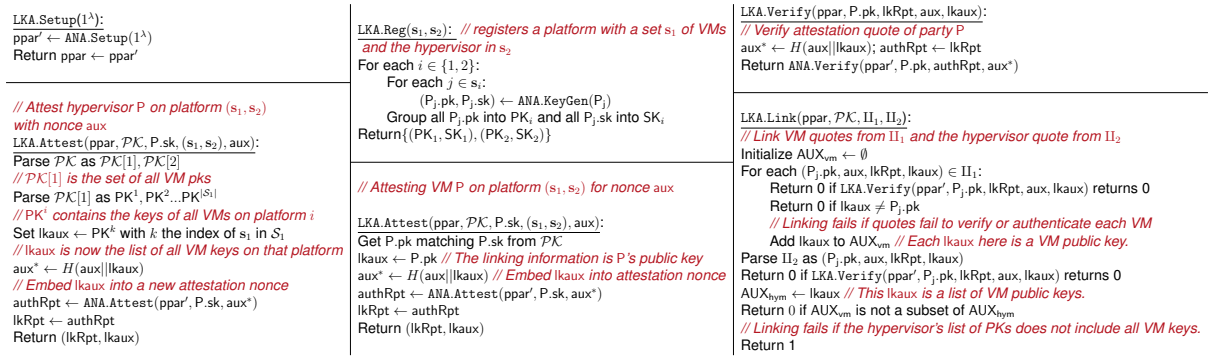


Figure 3.8: Our linked attestation scheme for platforms with 2 types of components: VMs (stored in $\mathcal{S}_1$) and hypervisors (stored in $\mathcal{S}_2$). Each type of component attests via a different LKA.Attest algorithm, the main difference between them being that the hypervisor embeds a list of public keys in its nonce.

The VMs and hypervisor generate linked attestations differently. The hypervisor first fetches the public keys of all the components registered with it on the same platform. It computes a new nonce as the hash of two concatenated values: the original auxiliary value aux and the list of the public keys. The component then runs ANA.Attest on the public parameters, this new nonce, and its private key, outputting the authenticated attestation report. By contrast, when a VM attests, it computes a new nonce from the original auxiliary value aux and (only) its own public key. The authenticated report obtained as a result is provided as the VM's linked report.
A VM (or a set of VMs) are considered to be linked to a hypervisor if, and only if, the following conditions hold simultaneously: (1) the attestations of all the purportedly-linked parties verify individually (if we run ANA.Verify it returns 1 for each individual attestation); (2) the public key that was successfully used to verify each of the VMs' attestation is part of the auxiliary value lkaux forwarded by the hypervisor.

Theorem 3.4.1 (Secure Linked Attestation)**.** *The* LKA *scheme is secure assuming that the* ANA *scheme is secure and that* H *is collision resistant. More formally, if there exists an adversary* $\mathcal{A}$ *that breaks the linked attestation game* $G_{\mathsf{LK}}(\lambda, F)$ *with advantage* $\mathsf{Adv}^{\mathtt{SEC}}_{\mathtt{LKA}}(\mathcal{A})$ *then there exists adversaries* $\mathcal{B}_1$ *and* $\mathcal{B}_2$ *such that :* $\mathsf{Adv}^{\mathtt{SEC}}_{\mathtt{LKA}}(\mathcal{A}) \leq \frac{1}{\mathsf{N}^2}(\mathsf{Adv}^{\mathsf{CR}}_{\mathsf{H}}(\mathcal{B}_1) + 2 \cdot \mathsf{Adv}^{\mathtt{SEC}}_{\mathtt{ANA}}(\mathcal{B}_2))$ *where* N *is the number of parties.*

Proof. We now prove our construction is secure with respect to the $G_{\mathsf{LK}}(\lambda, F)$ game :

$\mathbb{G}_0$: This is the original game $G_{\mathsf{LK}}(\lambda, F)$.

$\mathbb{G}_1$: We guess parties P, Q output by the adversary in the last part of its game. In other words, the challenger must draw at random two values between 1 and N, such that those values correspond to those chosen by the adversaries. We lose a factor $\frac{1}{\mathsf{N}^2}$.

$\mathbb{G}_2$: We now rule out that $\mathsf{H}(.,\mathsf{lkaux}) = (.,\mathsf{lkaux}')$ for any $\mathsf{lkaux} \neq \mathsf{lkaux}'$. Trivially, if the converse were true,we could break the collision resistance of H with equal probability and hence we lose a factor of $\mathsf{Adv}^{\mathsf{CR}}_{\mathsf{H}}(\mathcal{B}_1)$.
Note that now, since parties P and Q are registered on different platforms, since lkaux keys are unique, and since we have ruled out collisions, any honestly-generated attestations for P and Q will not link. The adversary's only hope is to forge an attestation for either one of those parties.

$\mathbb{G}_3$: At this point we rule out the fact that P's tuple $(\mathsf{aux}, \mathsf{lkRpt}, \mathsf{aux}^*)$ was in fact part of a tuple $(\mathsf{P}', ., \mathsf{aux}, \mathsf{lkRpt}, \mathsf{aux}^*) \in \mathcal{L}_{\mathsf{rpt}}$ (with $\mathsf{P} \neq \mathsf{P}'$). If that were so, we could construct an adversary against the authenticated attestation scheme (since P purports to be P′). In so doing an important oracle will be the signature oracle `oANA.Auth` added artificially in the authenticated attestation primitive; the latter will allow us to simulate `oLKA.CAttest` queries. We lose a factor $\mathsf{Adv}^{\mathtt{SEC}}_{\mathtt{ANA}}(\mathcal{A})$.

$\mathbb{G}_4$: We repeat the previous game hop for party Q.

At this point, the adversary can no longer win the game. □

3.5 Authorized Linked Attestation

The final step is to move from a primitive to a complete protocol. So far we have considered attestation as a set of algorithms executed by a verifier and a prover, but we have not considered how they exchange their outputs. In practice, remote attestation means exchanging messages over a potentially insecure channel. Our linked attestation primitive ensures that no adversary, even with access to an attestation oracle, can forge an attestation, thus having access to the message exchange within the insecure channel does not change anything. However, the verifier may not want to share its own attestation report with anyone and may want to control how it can be accessed. Therefore, we want to ensure that the actual report is only given to authorised parties, which we call attestation servers.
We will define an authorised linked attestation protocol that allows an attestation server to act as a verifier in the attestation procedures. The same server will also be the one to generate the auxiliary values required for the attestation (this provides freshness to the protocol). The server will furthermore be responsible for linking multiple attestations.

3.5.1 Syntax

In authorized linked attestation we consider a (single) attestation server S and platforms consisting of several types of components (as shown for linked attestation). The server will keep track of an evolving state, which is initially empty. However, as the server starts to attest various components, at every execution of the authorized attestation protocol, the server will output a verdict (indicating whether the component's individual attestation has failed or succeeded) and may – or may not – update its internal state. Intuitively, the state is meant to contain the linking information provided by each of the attesting components. After a number of attestations have taken place, the server might have enough information in its state to decide whether some of the components are linked or not.

Parties. As before we consider multiples attesting parties which can be of various type of components (VM or hypervisor). However we now consider an additional special party, *the server*, which has it own specific attributes.

Attesting Parties. Just as in the case of linked attestation, attesting parties can be various types of components (VM or hypervisor). Each attesting party stores a set of keys $(\mathsf{pk}, \mathsf{sk})$, as well as a compromise bit γ.

Server. The server is a special party which keeps track of the following attributes :

- $(\mathsf{pk}, \mathsf{sk})$: a tuple consisting of a public key pk (assumed to be unique and known to all other parties including the adversary) and a private key sk known only to the server. We use $\mathsf{S.pk}$ to indicate the public key of party S, and $\mathsf{S.sk}$ to indicate its private key.
- $\mathsf{S.st}$: a value called state, which stores tuples of linked attestations which may be linkable to each other.

Sessions. Parties can now interact with each other in sessions of an AKE protcol as defined in Section 2.6. The session is run by two party instances, one of the attesting party and the other, of the attestation server. Party instances keep track of the following session-specific attributes:

- $\pi_{\mathsf{P}}^{i}.\mathsf{sid}$: a session identifier, which will be useful in understanding which two party instances converse together.
- $\pi_{\mathsf{P}}^{i}.\mathsf{pidpk}$: the public key belonging to this instance's intended communication partner.
- $\pi_{\mathsf{P}}^{i}.\mathsf{T}$: a transcript of messages exchanged throughout a protocol session, in plaintext. Even if encryption is used at some point, parties append messages to their transcripts only after decrypting.

- $\pi_{\mathsf{P}}^{i}.\alpha$: this bit is originally set to 0, but can be changed to 1 if this party instance has accepted its partner as a legitimate entity to run the authorized linked attestation with.
- $\pi_{\mathsf{P}}^{i}.\mathsf{lst}$: this local state variable stores instance (and protocol) specific values, such as encryption keys, randomness, etc.

In addition server instances π_{S}^{j} keep track of the following attribute, which is the output of the immediate attestation process taking place:

- verdict: this attribute stores a bit, initially set to 0, which is flipped to 1 if the attestation server's instance has accepted the attestation received during that session.

Partner. We call two instances π_{P}^{i} and π_{Q}^{j} *partnered* if, and only if, the following conditions hold simultaneously: exactly one of P and Q is in fact the attestation server S; $\pi_{\mathsf{P}}^{i}.\mathsf{pidpk} = \mathsf{Q}.\mathsf{pk}$ and $\pi_{\mathsf{Q}}^{j}.\mathsf{pidpk} = \mathsf{P}.\mathsf{pk}$; and $\pi_{\mathsf{P}}^{i}.\mathsf{sid} = \pi_{\mathsf{Q}}^{j}.\mathsf{sid}$.

Syntax. The Authorized Linking Attestation scheme AZA is defined as the tuple of algorithms and protocols AZA = (AZA.Setup, AZA.Reg, AZA.Attest, AZA.Link) described as follows:

$\underline{\mathtt{AZA.Setup}(1^{\lambda}) \rightarrow (\mathsf{ppar}, \mathsf{S.sk}, \mathsf{S.pk}, \mathsf{S})}$: On input the security parameter λ, this algorithm outputs public parameters ppar, as well as the server handle S (such that S is equipped with newly generated keys pk, sk). The value ppar includes the maximal number of allowed disjoint linkable sets of parties, which we denote as L. The values ppar, S.pk, and S are public, S.sk remains private. The value S.pk is added as the first value in the set PK.

$\underline{\mathtt{AZA.Reg}(s_1, s_2, ..., s_L) \rightarrow \{(\mathsf{SK}_1, \mathsf{PK}_1), ..., (\mathsf{SK}_L, \mathsf{PK}_L)\}}$: this algorithm keeps as state a number L of sets $\mathcal{S}_i$ originally set to $\emptyset$, and a vector of sets of public keys $\mathcal{PK}$ (also initialized to $\emptyset$). On input a number of subsets $s_i (i = 1, 2, ..., L)$, this algorithm first checks that $\forall i, j, s_i \cap S_j = \emptyset$ (else the algorithm outputs $\perp$). If the relation is true, then the algorithm generates for each party $\mathsf{P}_j \in s_i$, (for all j, i) a tuple of public and private keys $\mathsf{P}_j.\mathsf{pk}$, $\mathsf{P}_j.\mathsf{sk}$, initializing P_j with those keys. We require the uniqueness of all the generated public keys. The subsets s_i are each added to greater sets $\mathcal{S}_i$. The algorithm groups the keys of all parties $\mathsf{P}_j \in s_i$ in a pair of private/public key subsets: $(\mathsf{PK}_i, \mathsf{SK}_i)$, updating the i-th component $\mathcal{PK}[i]$ of $\mathcal{PK}$ as $\mathcal{PK}[i] \cup \mathsf{PK}_i$. All parties are given access to the public-key subsets (and more generally, to $\mathcal{PK}$).

$\underline{\mathtt{AZA.Attest}(\mathsf{ppar}, \mathcal{PK}, \pi_{\mathsf{P}}^{i}, \pi_{\mathsf{Q}}^{j}) \rightarrow (\mathsf{verdict}, \mathsf{S.st})}$: This protocol is an interaction between two party oracles, such that exactly one of P, Q is S. The protocol yields a tuple of

values to the server: verdict and S.st (and no output for the other party). Both party oracles are assumed to update their attributes accordingly as the protocol unfolds.

$\underline{\texttt{AZA.Link}(\mathsf{ppar}, \mathcal{PK}, \mathsf{S.st}, s_1, ..., s_L) \rightarrow \{0,1\}}$: Given the public parameters and public key set, the server's current state, and a number of subsets of (purportedly-linked) parties, this algorithm outputs either 0 (the parties are not linked) or 1 (the parties are linked).

Correctness. We require two types of correctness properties. First, we require that running the protocol between two honest parties yields a verdict of 1 (accept) on the side of the attestation server. Secondly, we require that components that are linked at registration will be viewed as linked by the AZA.Link algorithm. More formally, we require that schemes AZA = (AZA.Setup, AZA.Reg, AZA.Attest, AZA.Link) be such that:

- For all $(\mathsf{ppar}, \mathsf{S.sk}, \mathsf{S.pk}, \mathsf{S}) \leftarrow \texttt{AZA.Setup}(1^\lambda)$ and for all parties $\mathsf{P} \in \mathcal{S}_i$ for some $1 \leq i \leq L$, it holds that $(\mathsf{verdict}, .) = \texttt{AZA.Attest}(\mathsf{ppar}, \mathcal{PK}, \pi_{\dot{\mathsf{P}}}, \pi_{\dot{\mathsf{S}}})$ (any legitimate party will successfully attest to the legitimate server).
- For all $(\mathsf{ppar}, ., ., \mathsf{S}) \leftarrow \texttt{AZA.Setup}(1^\lambda)$, for all subsets $\tilde{s_1}, \tilde{s_2}, ..., \tilde{s_L}$ such that there exist sets s_i for $i = 1, 2, ..., L$ such that $\tilde{s_i} \subset s_i$ and $\texttt{AZA.Reg}(s_1, s_2, ..., s_L)$ was called and did not result in $\perp$, it holds that : $\texttt{AZA.Link}(\mathsf{ppar}, \mathcal{PK}, \mathsf{S.st}, \tilde{s_1}, ..., \tilde{s_L}) = 1$ (parties that are registered together can be linked through the server's state).

3.5.2 Security

There are three fundamental properties we want AZA schemes to have: an authenticity guarantee for the attestation server (authorization); a confidentiality guarantee for the contents of the attestation (indistinguishability), and a linkability guarantee for honestly-behaving components (linking-security). The first notion, authorization, captures the fact that before reaching an accepting state, a (non-server) party must be sure that it is speaking to the legitimate server (game $G_{\mathsf{Az}}(\lambda, F)$). The second notion, indistinguishability, essentially covers Person-in-the-Middle confidentiality for the attestation protocol (game $G_{\mathsf{Ind}}(\lambda, F_{\texttt{Sign}})$). The last property, linking-security, refers to the fact that no PitM adversary with the ability to compromise components can convince an attestation server that a component is linked to another if that is not the case in reality (game $G_{\mathsf{Link}}(\lambda, F)$). Although this last property might seem similar to the security notion for our linked attestation primitive, there is one important difference between the two: in linked attestation the adversary has access to essentially two ways to generate an attestation (depending on whether the component is honest or compromised), whereas in authorized linked attestation the adversary will have more leeway in combining at-

testation material across sessions. The stronger adversary in this section will thus make for a stronger primitive in the end.
The three security games we define are parametrized by a function space F and a security parameter λ. They start with the challenger running the setup algorithm and outputting ppar as well as the handle S and its public key S.pk to the adversary. Note that this will not give the adversary black-box access to S's attributes: it simply allows the adversary to later instantiate new attestation protocol sessions for that server. The adversary will then have access to some, or all of the following oracles :

oAZA.Reg$(n_1, ..., n_L) \rightarrow (\mathsf{PK}_1, \mathsf{PK}_L)$: The authorized linked user-registration oracle creates a linked platform consisting of n_i components of the type indicated by $\mathcal{S}_i$. The challenger first instantiates subsets s_i as a tuple of n_i handles $\mathsf{P}_{i,j}$, with $1 \leq j \leq n_i$ and then runs the algorithm AZA.Reg$(s_1, s_2, ..., s_L)$, instantiating the parties with their keys and outputting the public keysets to the adversary. The subset consisting of the list of subsets is added to $\mathcal{L}_{\mathsf{Reg}}$.

oAZA.NewSession$(\mathsf{P}, \mathsf{Q}) \rightarrow \pi^i_{\mathsf{P}}$: On input the identity of a target party P and a partnering party Q, if both entities are correctly registered and exactly one of them is the server, then this oracle instantiates an instance of P whose partner will be instantiated as pidpk $=$ Q.pk. Note that in order to observe an honest session between two parties, an adversary would have to create two partnered instances, one of P and the other of Q.

oAZA.Send$(\mathsf{M}, \pi^i_{\mathsf{P}}) \rightarrow \mathsf{M}'$: This oracle simulates sending a message M to an instance π^i_{P}, and outputs the response M′i of the party instance. If the input message takes a special value M $=$ prompt and P is the initiator of the protocol (i.e., the first party to send a message), this will trigger π^i_{P} to output the first message in the protocol. We note that some messages, when sent, might trigger errors, leading to an output M′ $=\perp$. Other messages might trigger the attributes of the party (or instance) to be modified.

oAZA.RevealState$(\pi^i_{\mathsf{P}}) \rightarrow$ lst : On input a valid party instance, this oracle returns the value stored by the attribute lst of that instance.

oAZA.UseKey$(\mathsf{P}, f, \mathsf{aux}) \rightarrow \mathsf{aux}'$: On input a compromised party P (not the server), a function $f \in F$, and an auxiliary input value aux, this oracle evaluates f on P.sk and aux.

oAZA.Compromise$(\mathsf{P}) \rightarrow \mathsf{OK} \cup \perp$: On input a registered party P $\neq$ S, this oracle turns the party's compromise bit to 1 and returns OK. If P $=$ S or the party has not been registered, the output is an error symbol $\perp$.

We now proceed to describe each of the three security experiments we consider for our authorized linked attestation primitive.

Authorization Game. In the $G_{\mathsf{Az}}(\lambda, F)$, after the challenger runs the setup algorithm, the adversary $\mathcal{A}$ gets access to all the oracles described above. It ultimately stops with a stop message. We say $\mathcal{A}$ wins if, and only if, there exists an instance π_{P}^{i} such that $\mathsf{P} \neq \mathcal{S}$, for which the following conditions hold simultaneously:

- π_{P}^{i} ends in an accepting state, i.e., $\pi_{\mathsf{P}}^{i}.\alpha = 1$;
- There exists no server instance π_{S}^{j} such that π_{S}^{j} is partnered with π_{P}^{i}.

In other words, the adversary wins if it can make a registered party believe it has talked to the server when this is not the case.

Definition 3.5.1 (Authorization)**.** *We say that an authorized attestation scheme provides authorization if for all PPT $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(.)$ *such that:* $\Pr[\mathcal{A}$ *wins* $G_{\mathsf{Az}}(\lambda, F)] \leq \mathsf{negl}(\lambda)$*.*

Linking Game. In $G_{\mathsf{Link}}(\lambda, F)$, after the challenger runs the setup algorithm, the adversary $\mathcal{A}$ gets access to all the oracles above. It ends by outputting a tuple $(\mathsf{P}, s_1, ..., s_L)$ such that for all $1 \leq i \leq L$, $s_i \subset \mathcal{S}_i$ and there exists a unique i^* such that $\mathsf{P} \in \mathcal{S}_{i^*}$ and $s_{i^*} = \emptyset$. The challenger sets $\tilde{s_i} \leftarrow s_i$ for all $i \neq i^*$, and $\tilde{s_i} \leftarrow \mathsf{P}$. Then the challenger evaluates $b \leftarrow \mathtt{AZA.Link}(\mathsf{ppar}, \mathcal{PK}, \mathsf{S.st}, \tilde{s_1}, ..., \tilde{s_L})$. The adversary is said to win if, and only if the following conditions hold simultaneously:

- $b = 1$.
- There exists a party Q and an index $j^* \neq i^*$ such that $\mathsf{Q} \in \tilde{s_{j^*}}$ and P and Q were not output by the same $\mathtt{oAZA.Reg}$ query.

In other words, for this second game, the adversary has to run several sessions between (potentially compromised) parties and the (honest) server, thus bringing the server's state to a point where linkage can be verified based on that state.

Definition 3.5.2 (Linking)**.** *We say that an authorized attestation scheme provides linking if for all PPT $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(.)$ *such that :* $\Pr[\mathcal{A}$ *wins* $G_{\mathsf{Link}}(\lambda, F)] \leq \mathsf{negl}(\lambda)$*.*

Indistinguishability Game. Finally, for $G_{\mathsf{Ind}}(\lambda, F_{\mathtt{Sign}})$, once the challenger has finished the setup, it also draws a bit b at random. The adversary gets once more access to the oracles described above. It finally outputs a tuple $(\pi_{\mathsf{P}}^{i}, \mathsf{m}_0, \mathsf{m}_1)$, consisting of a party instance and two messages, such that: $|\mathsf{m}_0| = |\mathsf{m}_1|$ and $\pi_{\mathsf{P}}^{i}.\alpha = 1$. The challenger uses its knowledge of π_{P}^{i}'s state on input m_b (which is m_0 or m_1 depending on the challenger's hidden bit) to simulate outputting a message M_b which corresponds to the

next protocol message of π^i_{P} as that party would have sent it. Clearly if the protocol requires messages be sent in plaintext, $\mathsf{M}_b = \mathsf{m}_b$. The instance π^i_{P}, as well as any of its partnering instances, are closed and may no longer be used in any oracle. The adversary may subsequently continue to use oracles at will (except the instances closed above) and eventually outputs a guess $d \in \{0, 1\}$. We say the adversary wins if, and only if, the following conditions hold simultaneously:

- $d = b$.
- No oAZA.RevealState query was made for either π^i_{P}, nor for any instance π^j_{S} of the server such that π^i_{P} and π^j_{S} are partnered.

Definition 3.5.3 (Indistinguishability)**.** *We say that an authorized attestation scheme provides indistinguishability if for all PPT $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(.)$ *such that :* $\Pr[\mathcal{A}$ *wins* $G_{\mathsf{Ind}}(\lambda, F_{\mathtt{Sign}})] \leq \mathsf{negl}(\lambda)$.

3.5.3 Construction

Our construction of the AZA scheme can be seen in the Figure 3.9. We use an ACCE-secure AKE protocol, namely TLS. We also assume the existence on an underlying LKA scheme that we use for the LKA.Setup, LKA.Reg and LKA.Link in a straightforward manner. However, the LKA.Attest algorithm is no longer a primitive, but a protocol between two instances of two parties, P and Q. For simplicity of exposition, we assume that the instance of Q is the server attesting the component identified by P.
The protocol proceeds as follows. First, P and Q execute the TLS protocol, with P playing the role of the client and Q playing the role of the server. The role of the TLS protocol is two-fold: first, P authenticates the server, so that they can determine whether this party is allowed to obtain attestation data. Second, it leads to the establishment of a secure channel, such that the following messages can be passed on in a secure manner. Once the traffic key(s) established, the protocol continues as follows. First, the server uniformly randomly samples a nonce aux, which is embedded in the first message of the protocol, AttestationRequest. In response, the party P executes the LKA.Attest algorithm and the output, consisting of a lkRpt and the linkage information lkaux, is then sent to the server. The server will subsequently update his state.
In order for two components to be linked by the server successfully, the following conditions have to be met. First, the two components' attestation must be valid (their associated verdicts equals 1). Second, the two lkaux must be subsets of each other; essentially, the key that the VM used as part of its attestation must be found in the lkaux provided by the hypervisor.

$\underline{\texttt{AZA.Setup}(1^\lambda)}$:	$\underline{\texttt{AZA.Reg}(\mathrm{s}_1, \mathrm{s}_2)}$:	$\underline{\texttt{AZA.Link}(\mathsf{ppar}, \mathcal{PK}, \mathsf{S.st}, \mathrm{s}_1, \mathrm{s}_2)}$:
$\mathsf{ppar} \leftarrow \texttt{LKA.Setup}(1^\lambda)$	$\{(\mathsf{PK}_1, \mathsf{SK}_1), (\mathsf{PK}_2, \mathsf{SK}_2)\} \leftarrow \texttt{LKA.Reg}(\mathrm{s}_1, \mathrm{s}_2)$	Parse $\mathsf{S.st}$ as Π_1, Π_2
Create S	Return $\{(\mathsf{PK}_1, \mathsf{SK}_1), (\mathsf{PK}_2, \mathsf{SK}_2)\}$	Return $\texttt{LKA.Link}(\mathsf{ppar}, \mathcal{PK}, \Pi_1, \Pi_2)$
$(\mathsf{S.pk}, \mathsf{S.sk}) \leftarrow \texttt{SIG.KeyGen}(1^\lambda)$		
Return $(\mathsf{ppar}, \mathsf{S.pk}, \mathsf{S.sk}, \mathsf{S})$		

$\underline{\texttt{AZA.Attest}(\mathsf{ppar}, \mathcal{PK}, \pi_\mathsf{P}^i, \pi_\mathsf{S}^j)}$:

Component		Server
	Establish TLS channel	
$(\mathsf{lkRpt}, \mathsf{lkaux}) \leftarrow \texttt{LKA.Attest}(\mathsf{ppar}, \mathcal{PK}, \mathsf{P.sk}, (\mathrm{s}_1, \mathrm{s}_2), \mathsf{aux})$	$\xleftarrow{\mathsf{aux}}$ AttestationRequest	$\mathsf{aux} \xleftarrow{\$} \mathcal{AUX}$
	$\xrightarrow{(\mathsf{lkRpt}, \mathsf{lkaux})}$ AttestationResponse	$\mathsf{verdict} \leftarrow \texttt{LKA.Verify}(\mathsf{ppar}, \mathsf{P.pk}_j, \mathsf{lkRpt}, \mathsf{aux}, \mathsf{lkaux})$
		Add $(\mathsf{P}_j\mathsf{.pk}, \mathsf{aux}, \mathsf{lkRpt}, \mathsf{lkaux})$ to Π_i in $\mathsf{S.st}$

Figure 3.9: Our authorized linked attestation scheme for 2 types of components

We note that if the server has at some point accepted the attestation of a component (thus updating its state to add the linking information), and if later a failed attestation occurs with respect to that component, the server updates state as follows: it ignores the linking information provided in the second attestation and it removes prior linking information provided by that component.

Theorem 3.5.1 (Authorization)**.** *Our* AZA *construction provides authorization assuming that the* AKE *protocol provides server authentication. More formally, if there exists an adversary $\mathcal{A}$ that breaks the $G_{\mathsf{Az}}(\lambda, F)$ with advantage $\mathsf{Adv}^{\mathtt{Az}}_{\mathtt{AZA}}(\mathcal{A})$ then there exists an adversary $\mathcal{B}$ such that :* $\mathsf{Adv}^{\mathtt{Az}}_{\mathtt{AZA}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathtt{ACCE-Auth}}_{\mathtt{AKE}}(\mathcal{B})$*.*

Proof. (sketch) Note that in order to win this game, the adversary must make a party accept a session with the server, such that no matching server instance exists. This is against the server authentication property we assume of TLS. □

Theorem 3.5.2 (Linking)**.** *Our* AZA *construction provides linking assuming that the* AKE *is at least ACCE secure and that* LKA *is $G_{\mathsf{LK}}(\lambda, F)$ secure. More formally, is there exists an adversary $\mathcal{A}$ that breaks $G_{\mathsf{Link}}(\lambda, F)$ with advantage $\mathsf{Adv}^{\mathtt{Link}}_{\mathtt{AZA}}(\mathcal{A})$ then there exists a adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$ such that :* $\mathsf{Adv}^{\mathtt{Link}}_{\mathtt{AZA}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathtt{ACCE-SC}}_{\mathtt{AKE}}(\mathcal{B}_1) + \mathsf{Adv}^{\mathtt{SEC}}_{\mathtt{LKA}}(\mathcal{B}_2)$*.*

Proof. (sketch) A key observation for this game is that the adversary cannot impersonate a server or determine it to provide bad randomness. Instead, the adversary can compromise components and run TLS sessions on their behalf with the server, or try to obtain input from honest components instead. We distinguish between two types of adversary behaviours.

Say $\mathcal{A}$ has never queried $\texttt{oAZA.Compromise}$ for some party P. If the adversary prompts P to run a session, then $\mathcal{A}$ will not actually know anything about the messages (so it cannot misbehave on the quote, the nonce, or anything else). If $\mathcal{A}$ runs the TLS session instead of P, it will learn the channel key, but will not be able to prompt P for the

quote (since P wants to run TLS and not the attestation protocol, and since $\mathcal{A}$ cannot impersonate the server).
Say $\mathcal{A}$ queries oAZA.Compromise for some party P. Then the adversary can run TLS sessions on behalf of that party and query oAZA.UseKey in an attempt to get information on the quotes. However, in that case, the attestation of that component fails, except again if we break linked authentication security. This essentially means that the adversary has no way to maul honestly generated input to suit its purposes. □

Theorem 3.5.3 (Indistinguishability)**.** *Our* AZA *construction provides indistinguishability assuming that the* AKE *is (minimally) ACCE secure. More formally, is there exists an adversary $\mathcal{A}$ that breaks $G_{\mathsf{Ind}}(\lambda, F_{\mathtt{Sign}})$ with advantage $\mathsf{Adv}^{\mathtt{Ind}}_{\mathtt{AZA}}(\mathcal{A})$ then there exists an adversaries $\mathcal{B}$ such that : $\mathsf{Adv}^{\mathtt{Ind}}_{\mathtt{AZA}}(\mathcal{A}) \leq \frac{1}{q_{\mathsf{sess}}}\mathsf{Adv}^{\mathtt{ACCE-SC}}_{\mathtt{AKE}}(\mathcal{B})$ where q_{sess} is the number of sessions.*

Proof. (sketch) Like in the proof of authorization, the reduction here is immediate. The property of sACCE (which is already provided by TLS 1.2, whereas TLS 1.3 gives even stronger guarantees) implies that messages exchanged across the TLS channel are secure. □

3.6 Implementation

We implemented a proof of concept of our authorized linked attestation scheme. The implementation consists of three parts, a client for the hypervisor, a client for the Virtual Machines, and an attestation server written in Python 3. We do not consider the underlying NFV or cloud infrastructure, since our scheme abstracts those environments and can be used in any kind deep-attestation scenario. Therefore, any computer equipped with a TPM 2.0 (which can also be emulated) and which has virtualization capacities suffices for the purposes of our implementation.

3.6.1 Setup

In what follows we will describe that infrastructure, our implementation, and our results.

The infrastructure. We summarize our testing architecture in Figure 3.10 (note that some of our tests use more than 2 VMs, up to 55).

Hypervisor. Our *hypervisor* is a laptop running Ubuntu 20.04.3 (kernel version 5.11.0-40) with an Intel i7-10875H CPU, 32GB RAM and a STMicroelectronics ST33TPHF2XSPI

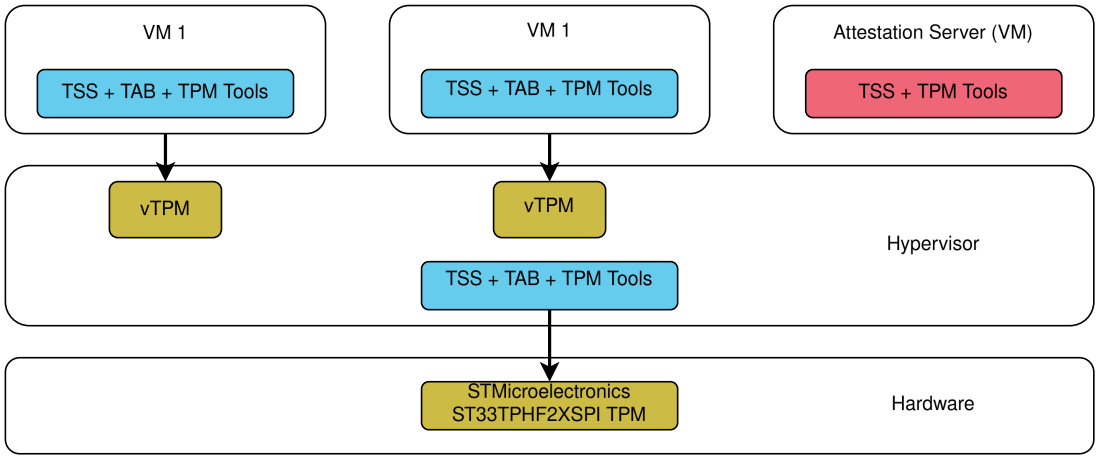


Figure 3.10: Architecture for tests.

TPM. We used KVM to turn this laptop into a hypervisor.

VM. In order to achieve a high attestation performance, we used a full *virtual TPM implementation*, using QEMU [38] with libtpms [39] version 0.7 and swtpm [40] version 0.5. All *virtual machines* are QEMU virtual machines (version 4.2.1) with 1 cores and 512 MB RAM running Fedora 35 Cloud. The VM as well as the virtual TPM instances are managed using Vagrant and Vagrant-Libvirt plugin.

Network. The hypervisor, server, and VMs communicate through a private network created with Vagrant. Thus, connection time is not considered in our tests.

TPM Communications. To communicate with the TPM we used tpm2-tss, tpm2-abrmd and tpm2-tools from the tpm2-software [41]. Note that the tpm2-tss project implements the TPM software stack (TSS), which is an API specified by the Trusted Computing Group to interact with a TPM. The tpm2-abrmd implements the access broker and resources to manage concurrent access to the TPM and manage memory of the TPM by swapping in and out of the memory as needed (hardware TPM have limited memory). tpm2-tools are a set of command-line tools based on TSS, which are used to send commands to the TPM. We used Python to wrap tpm2-tools commands.

Server. The *attestation server* is also a virtual machine, with the same characteristics as those above. This allows us to test our implementation on a single machine. We establish a secure connection between the client and the server by using Python's SSL library and then sending protocol messages as encoded json strings directly into TLS socket.

3.6.2 Results

We perform three main types of experiments on our setup.

Attestation benchmarks. The first experiment is a comparison of hypervisor attestation time and VM attestation time. Although both those processes have some (very small) amount of noise, our values faithfully show the (level of mangnitude of the) difference between attesting a component through the real (physical) TPM (hypervisor attestation) and attesting it by using a virtual TPM (VM attestation).
We ran 100 attestations for the hypervisor and 100 attestations for a virtual machine. The results exhibit significant variability so Table 3.1 presents the minimum, the maximum, mean and the median value of those 100 trials. As expected, the time required for an attestation using a hardware TPM is much higher than using a Virtual TPM.

Table 3.1: Hypervisor and VM deep attestation benchmarks

	min	median	mean	max
Hypervisor (s)	3.22	5.30	5.68	11.55
VM (s)	0.66	0.97	1.03	1.41

Scaling benchmarks. For our second and third experiments, we wanted to see how the overall runtime of our scheme evolves with the number of virtual machines that need to be attested, when the attestation is sequential (second experiment) or parallelized for the VM attestations (third experiment). In both cases, each experiment run first executed the attestation of the hypervisor, and then (sequentially or in parallel) the attestations of a varying number of VMs (up to a maximum of 55). For both experiments we did 100 runs of the experiment. The median runtime was calculated in each experiment (sequential or parallel) for each number of VMs, and the results were plotted in Figure 3.11 (the two lower curves, the middle curve is sequential runtime, while the bottom curve is parallel runtime).

Comparison to single-channel attestation. We did not implement single-channel attestation. However, since we implemented hypervisor and VM attestations, we can theoretically estimate the runtime of single-channel attestation for a varying number of VMs, which we plot in Figure 3.11 (upper curve). Indeed, a single-channel attestation process for a single VM includes a VM attestation *and* a hypervisor attestation. If we want to run it for 2 VMs, then we need to perform 2 hypervisor attestations and 2 VM attestations. This cannot be easily parallelized either, because the same TPM has to run the attestations. This yields a much higher runtime, as depicted in Figure 3.11.

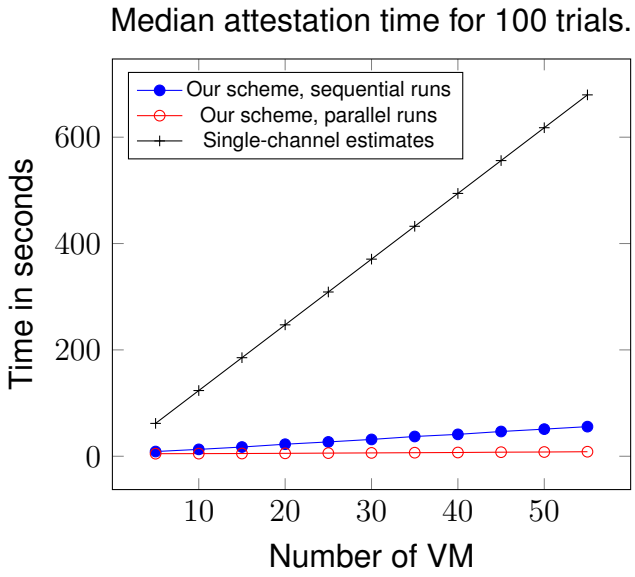


Figure 3.11: Attestation of multiple VM

Comparison to multi-channel attestation. Although our method follows basic multi-channel attestation approaches, we do add an extra computation (a hash function computation) compared to traditional multi-channel attestation. In addition, we require a little extra memory overhead for both the attestation server and for each platform, so that the additional attestation keys are stored for each VM. There is also a slight transmission overhead, since those keys are also sent upon attestation. However, the transmission overhead is negligible since it only appears for the hypervisor attestation (which occurs only once).

3.7 Conclusion

In this chapter, we have proposed layer binding in deep attestation without running into the complexity of single-channel attestation. Our construction achieves the best of both worlds, with a complexity similar to multi-channel attestation, but with the strong binding properties of single-channel attestation.
We accompany our construction with a proof-of-concept implementation that clearly demonstrates the viability and scalability of our solution, especially when VM attestations are run in parallel.
In addition, we are the first to present a full, formal treatment of our new protocol, which we call authorised linked attestation. Our construction of authorized linked attestation is modular, building on primitives with progressively stronger properties. Our underlying

assumption is a primitive called basic attestation. We show that to prove security, the attestations must be able to reflect the compromise of the component. In addition, we rely on a collision-resistant hash function, a secure EUF-CMA signature scheme, and the ACCE security of a TLS protocol.
However, our model (and scheme) does not directly account for other features of virtual infrastructures, such as privacy CAs, migrating VMs, multiple hypervisors managing the same VM, or even replacing TPMs. More importantly, this scheme does not take into account multitenancy, which, as we have seen, is the norm with NFV. This is the subject of the next chapter.

4

Attestation in multi-tenant virtualized environment

Contents

In Section 1.2.2 we reviewed several deep attestation schemes and in Chapter 3 we introduced a new protocol that aims to bring the benefits of both single and multi-channel deep attestation. However, all of these schemes consider monolithic virtualised infrastructures, where both the VMs and the hypervisor are controlled by a single entity. However, ETSI describes use cases for NFV [42] with multi-tenancy. Furthermore, deep attestation could be of interest in any cloud-like architecture where multi-tenancy is common, not just in the settings of NFV. In this chapter, we will see how to adapt our layer linking mechanism in the context of a multi-tenant infrastructure using a TPM as a root of trust. This work was published at ESORICS 2023 [3].

4.1 Introduction

For this work, we consider the typical multi-tenant architecture depicted in Figure 4.1, which is equipped with a hardware Trusted Platform Module (TPM) and spawns a virtual TPM (vTPM) for each VM it manages. VMs can be operated by tenants and one tenant can have multiple VMs. Every tenant has a dedicated verifier to perform attestation.

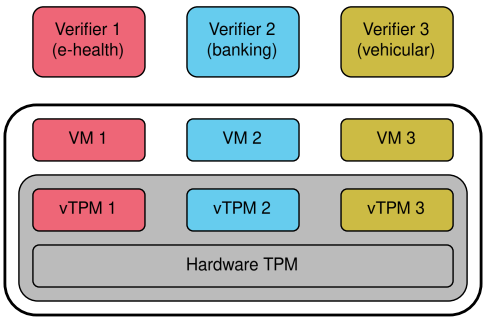


Figure 4.1: Multi-tenant architecture where each VNF belongs to a different tenant.

Considering such an infrastructure also implies a slight change in our use case. In traditional deep attestation, we have the immediate use case of an infrastructure operator wanting to monitor the state of the infrastructure using remote attestation.
We are now in a similar situation to Keylime [43], [44], a TPM remote attestation solution for cloud applications that provides system integrity monitoring. Keylime allows to deploy an agent on each target VM: the agent sends an attestation every few seconds to a cloud verifier managed by a specific tenant.

Note 4.1.1: Why TPM?

TPMs are widespread chips but with limited functionalities and computing power. Hence it is more difficult to meet the challenges of multi-tenancy with a TPM than with a more flexible RoT (and this is actually the case; our solution can be much simpler with other RoT). However the ETSI propose in some specifications the use of TPM in its specification document. In Chapter 5, we will free ourselves from this constraint. We are also not considering the possibility of modifying the TPM. Although this is technically possible, it would require updating the specifications and all TPM, which is a lengthy process.

In such multi-tenant environments, existing protocols do not achieve some of the desired properties. For example, multi-channel deep attestation enables scaling by reducing the number of hypervisor attestations to a single attestation. Attestation requests are accompanied by a nonce in order to ensure the freshness of the attestation: if a single entity controls all the VMs it can verify the freshness of the single attestation, but with multiple tenants, each use a different nonce, thus implying the need for multiple attestations. In addition, in a multi-tenant context, the privacy concern due to attestation arises both in terms of inter-tenant privacy (i.e. privacy between tenants) and host privacy (i.e. privacy of the infrastructure). In particular, TPM attestation reveals the full configuration of the attested machine; in a multi-tenant environment, the host may wish to keep its configuration secret from some of the tenants.

Note 4.1.2: Why not Keylime?

Our use case is very similar to that of Keylime, a RedHat-backed project that works efficiently even on very large cloud infrastructures [45]. Unfortunately, Keylime is limited to VM attestation (rather than linked hypervisor/VM attestation). It also provides neither tenant privacy, nor hypervisor configuration privacy.

4.1.1 Our contributions

Our work makes a triple contribution.

A new protocol. We propose a primitive called privacy-preserving multi-tenant attestation (PP-MTA), which provides attestation, but also layer-binding and privacy: *Inter-tenant privacy* (no tenant can learn if other tenants share the same platforms as the user's own VMs) and *Configuration hiding* (the hypervisor attestation convinces a

tenant that the hypervisor is well-configured without revealing the configuration). These strong properties are achieved with no modification to the TPM, and rely on ZK-SNARK, vector-commitment schemes, and secure-channel establishment.

Formal analysis. We formally model and prove the security and privacy of our protocol. We *extend* the layer-binding properties defined in Chapter 3 [2] to a multi-tenant environment, and *add* definitions for *inter-tenant privacy* and *configuration-hiding*. We formally quantify the privacy of our protocol. The security of our scheme relies on standard ACCE-secure channels, secure vector commitments, and zero-knowledge succinct non-interactive arguments of knowledge (ZK-SNARK), but also two new properties: partner-hiding authenticated key-exchange (AKE) and collision-resistant vector commitments.

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

4.1.2 Comparison with Authorized Linked Attestation

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

4.2 Technical Overview

In this section we give an overview of our solution for DA in multi-tenant environments. Recall that in a use case such as Keylime, we want to provide linkable attestation in multi-tenant environments such that the resulting solution is practical (scalable and efficient), secure, and provides strong privacy for both the tenants and the provider of the physical infrastructure, and does not require modifications to the TPM.

Solution outline. A naïve application of our scheme presented in Chapter 3 [2] to multi-tenant scenarios provides layer-linking but not privacy. The result also scales poorly.

Hypervisor-configuration hiding requires that attestation quotes, signed by a physical root of trust (TPM), only prove the validity of the PCR-measurement, without revealing it. The naïve approach of the TPM computing a zero-knowledge proof would require TPM modifications. This is precisely what we want to avoid. In our solution, this is achieved by the use of ZK-SNARK.
Moreover, the protocol in Chapter 3 is designed for environments where a single entity owns the infrastructure and all the VMs. Naïvely reusing that solution in multi-tenant environments creates a significant bottleneck when multiple tenants request attestations simultaneously; this could be resolved by *batching* [46] hypervisor attestations for multiple tenants. Batching is challenging to achieve simultaneously with inter-tenant privacy and layer-linking. As layer-linking requires VM attestations to be linked to their managing hypervisor's attestation, the latter has to include some binding information to the VMs it is managing. Yet, in multi-tenant environments such VMs might belong to distinct users; each user must only verify the binding of its own VMs to the hypervisor. To bridge that gap, we use vector-commitment schemes to store (in a hidden form) linking information to VMs hosted on the hypervisor. This allows each tenant to open some specific positions in that commitment, learning nothing about other positions.

Our solution. We introduce several new elements to the layer-linking DA solution of Chapter 3. A trivial, but necessary modification is at setup: unlike authorized linked attestation, we need to account for the *ownership*, by a tenant, of a VM. In our infrastructure, tenants will have to use long-term credentials in order to register new VMs. As VM attestations are independently requested by VM-owners, we can simply use typical multiple-channel attestation for this step. Our key contribution, however, is a novel hypervisor-attestation method which is scalable (as it allows for simultaneous attestation requests to be batched), linkable to VM attestations, and guaranteeing inter-tenant privacy and hypervisor-configuration hiding.
The hypervisor may receive one or more attestation requests from one or more tenants. Requests are buffered if the TPM is busy. As soon as the TPM is free, the hypervisor makes an attestation request for the aggregated queries, using a special nonce computed as follows.
For each tenant, the hypervisor retrieves linking information and concatenates it with that tenant's nonces (per each VM). Then it assigns a random position in a vector commitment to each tenant, places the concatenated linking-values for the tenant at that position, and commits to the resulting vector[1]. This is the nonce used in the

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

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

4.3 Model

Our security model applies to the virtualization architecture shown in Figure 4.1. Tenants associated with unique identities $\mathcal{T}$ can register a number of virtual machines VM on a hypervisor $\mathcal{H}$.
Each hypervisor $\mathcal{H}$ has a physical root of trust represented by a TPM TPM. We assume each physical machine (with a unique hypervisor $\mathcal{H}$) upper-bounds the number of tenants $N_{\mathcal{T}}$ it can host, and the number of VMs N_{VM} each tenant can have on $\mathcal{H}$. Such bounds do exist in practice, usually driven by physical constraints. For the sake of legibility, we assume *universal* bounds (for all hypervisors), rather than *local*, hypervisor-specific ones.
We call the list of tuples of PCR measurements and accepted values during hypervisor attestation the *configuration* of the hypervisor. We assume the existence of a set (of more than one element) $\mathcal{CONF}$ of possible configurations for each hypervisor. In the quote, the current configuration is represented as the hash of the list of PCRs.

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 [47] which can compromise the software of any VM (co-resident) or hypervisor (system level). However, we rule out hardware adversaries, including side-channel attacks.

Security/privacy notions. We formally define the privacy properties required for our protocol and the adversary model in Sections 4.3.2 and 4.3.3. 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 section 4.3.4.

4.3.1 Syntax

We formally define a new primitive called privacy-preserving multi-tenant attestation (PP-MTA). It consists of 9 PPT algorithms: PP-MTA= ($\mathtt{MTA.Setup}$, $\mathtt{MTA.HSetup}$, $\mathtt{MTA.TKGen}$, $\mathtt{MTA.VMReg}$, $\mathtt{MTA.HAtt}$, $\mathtt{MTA.VMAtt}$, $\mathtt{MTA.VerHAtt}$, $\mathtt{MTA.VerVMAtt}$, $\mathtt{MTA.Link}$) as follows:

- $\mathtt{MTA.Setup}(1^\lambda) \to \{\mathsf{ppar}, \mathsf{spar}\}$: On input a security parameter λ, this algorithm outputs public parameters ppar (including the bounds $N_\mathcal{T}$, N_{VM}, and valid configuration-set $\mathcal{CONF}$), and private parameters spar (which may be instantiated to $\perp$ if not useful). The public parameters are input implicitly for every subsequent algorithm.
- $\mathtt{MTA.HSetup}(\mathsf{ppar}) \to \{\mathcal{H}.\mathsf{pk}, \mathcal{H}.\mathsf{sk}, \mathsf{AK.pk}, \mathsf{AK.sk}, \mathcal{H}.\mathsf{Conf}, \mathcal{H}.\mathsf{state}\}$: 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.
- $\mathtt{MTA.TKGen}(\mathsf{ppar}) \to \{\mathcal{T}.\mathsf{pk}, \mathcal{T}.\mathsf{sk}\}$: This algorithm generates public and private keys for a single tenant $\mathcal{T}$. All parties have access to all the public keys. Only the tenant has access to its private key.
- $\mathtt{MTA.VMReg}(\mathcal{H}, \mathcal{T}.\mathsf{sk}, \mathsf{VMdesc}) \to \{(VM, \mathsf{VAK.pk}), \mathsf{VAK.sk}, \mathcal{H}.\mathsf{state}\} \cup \perp$: This algorithm performs the registration, by tenant $\mathcal{T}$, of a VM of description VMdesc on the machine with hypervisor $\mathcal{H}$. If the tenant's request exceeds either the hypervisor's capacity to host new tenants $N_\mathcal{T}$, or its capacity for VMs for this tenant N_{VM}, then

the algorithm returns $\perp$. Otherwise, the hypervisor creates the required VM, for which it returns a handle VM, as well as a tuple of public/private parameters, corresponding to the *attestation* keypair for that VM, as stored by the vTPM: $(\mathsf{VAK.pk}, \mathsf{VAK.sk})$. The algorithm also requires mutual authentication of the tenant and the hypervisor, enabling $\mathcal{H}$ to update its state $\mathcal{H}.\mathsf{state}$. If the authentication fails, the algorithm returns $\perp$. Otherwise it returns to the tenant the handle VM and the public keys and $\mathsf{VAK.pk}$.

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

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

$\mathtt{MTA.VMAtt}\langle VM(\mathsf{VAK.sk}), \mathcal{T}(\mathcal{T}.\mathsf{sk}, \mathsf{nonce})\rangle \rightarrow \{\mathsf{ATT}_{VM}\} \cup \perp$: 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}.

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

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

4.3.2 Inter-tenant Privacy

Intuitively, inter-tenant privacy ensures that a malicious, but legitimate tenant cannot tell whether or not VMs from other tenants are running on the same hypervisor as its own VMs. In the real-world, tenants might be well-aware that they are sharing resources with other tenants. However, by considering a much stronger privacy notion, we ensure that this (and potentially other) information is not leaked by the attestation. This makes

our protocol usable in all situations, not just those in which some leakage is acceptable. The definition we provide (and prove for our protocol) only guarantees that leakage is avoided at the protocol-level: the adversary may have alternative means of knowing about VMs co-hosted on the same hardware.

Formally, inter-tenant privacy is defined as a game (depicted in Figure 4.2) between a challenger $\mathcal{G}$ and an adversary $\mathcal{A}$. The challenger runs the setup algorithm $\texttt{MTA.Setup}(1^\lambda)$. It then sets up a hypervisor $\mathcal{H}$ by running $\texttt{MTA.HSetup}(\text{ppar})$. $\mathcal{G}$ initiates $\mathcal{L}_H := \emptyset$ and $\mathcal{L}_C := \emptyset$. The challenger 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:

- $\underline{\text{oHonTReg}_b(\{\text{VMDesc}_i\}_{i=1}^{\ell})}$: this oracle depends on bit b. Given as input a set of VM descriptions VMDesc_i, this oracle internally runs the key-generation algorithm $\texttt{MTA.TKGen}(\text{ppar})$, receiving either $\perp$ (too many tenants) or a handle $\mathcal{T}$ and keys $\mathcal{T}.\text{pk}, \mathcal{T}.\text{sk}$. 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 oHonTReg did not output $\perp$: if $b = 1$, the oracle runs $\texttt{MTA.VMReg}(\mathcal{H}, \mathcal{T}.\text{sk}, \text{VMDesc}_i)$ for each VM in the input set, obtaining handles VM, keys $\text{VAK.pk}, \text{VAK.sk}$, and an updated hypervisor state $\mathcal{H}.\text{state}$, containing tuples of the form ($\mathcal{T}$, VM_i, VAK.sk_i, VAK.pk_i, REAL) for each VM. If $b = 0$, then the VMs are not truly created: instead, the oracle generates random values VAK.pk_i for each $i = 1, \ldots, \ell$, and handles VM_i, updating the hypervisor state with tuples of the form $(\mathcal{T}, VM_i, \text{VAK.pk}_i, \text{FAKE})$. Finally, the oracle outputs the following values to the adversary: $\mathcal{T}, \{VM_i\}_{i=1}^{\ell}$ as well as keys: $\mathcal{T}.\text{pk}, \{\text{VAK.pk}_i\}_{i=1}^{\ell}$. If $\ell > N_{VM}$, the output of MTA.VMReg will be $\perp$, forwarded to the adversary instead of the VM information. The adversary can, in parallel, use MTA.TKGen algorithm to register malicious tenants: these will be added by the challenger to $\mathcal{L}_C$.
- $\underline{\text{oVMReg}(\mathcal{T}, \text{VMDesc})}$: given as input a (registered) tenant $\mathcal{T} \in \mathcal{L}_H$ and a VM with description VMDesc, this oracle internally runs $\texttt{MTA.VMReg}(\mathcal{H}, \mathcal{T}.\text{sk}, \text{VMDesc})$. If the bound N_{VM} has still not been reached for tenant $\mathcal{T}$ then the algorithm outputs $(VM, \text{VAK.pk})$ as in the previous oracle. The hypervisor state $\mathcal{H}.\text{state}$ is updated. A malicious tenant can always register a new VM by running the MTA.VMReg algorithm directly.
- $\underline{\text{oHAttest}(\mathcal{T})}$: given as input a registered tenant $\mathcal{T} \in \mathcal{L}_H$, this oracle simulates running MTA.HAtt between $\mathcal{T}$ and the hypervisor $\mathcal{H}$. The adversary gains a transcript τ_{HAtt} of the protocol run (or $\perp$ if *e.g.*, $\mathcal{H}$ does not exist or if $\mathcal{T}$ has no VMs registered on $\mathcal{H}$). If the VMs created for this tenant were fake (the bit b picked by the challenger

is 0), the hypervisor attestation is done over the current configuration and the VMs currently existing on the machine.

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

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

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

Figure 4.2: The inter-tenant privacy game.

Definition 4.3.1 (Inter-tenant privacy)**.** *A PP-MTA scheme* PP-MTA*= (*`MTA.Setup`*,* `MTA.HSetup`*,* `MTA.TKGen`*,* `MTA.VMReg`*,* `MTA.HAtt`*,* `MTA.VMAtt`*,* `MTA.VerHAtt`*,* `MTA.VerVMAtt`*,* `MTA.Link`*) is* $(N_{\mathcal{T}}, N_{VM}, \epsilon)$-inter-tenant private *if, and only if, for every probabilistic polynomial adversary* $\mathcal{A}$*, the following holds:*

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

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

4.3.3 Hypervisor-Configuration Hiding

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

In the hypervisor configuration-hiding game (figure 4.3), the adversary gets access to the following oracle:

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

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

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

Figure 4.3: The configuration-privacy game.

Definition 4.3.2 (Configuration privacy)**.** *A PP-MTA scheme* PP-MTA= *(*`MTA.Setup`*,* `MTA.HSetup`*,* `MTA.TKGen`*,* `MTA.VMReg`*,* `MTA.HAtt`*,* `MTA.VMAtt`*,* `MTA.VerHAtt`*,* `MTA.VerVMAtt`*,* `MTA.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 $\frac{1}{|\mathcal{CONF}|}$).

4.3.4 Linkability Security

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

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

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

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

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

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 4.3.3 (Linkability security)**.** *A PP-MTA scheme* PP-MTA= *(*$\mathtt{MTA.Setup}$*,* $\mathtt{MTA.HSetup}$*,* $\mathtt{MTA.TKGen}$*,* $\mathtt{MTA.VMReg}$*,* $\mathtt{MTA.HAtt}$*,* $\mathtt{MTA.VMAtt}$*,* $\mathtt{MTA.VerHAtt}$*,* $\mathtt{MTA.VerVMAtt}$*,* $\mathtt{MTA.Link}$*) is* $(q_{att}, N_{\mathsf{P}}, \epsilon)$-linkable *if, and only if, for every probabilistic polynomial adversary* $\mathcal{A}$*, the following holds:*

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

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

4.4 Construction

In this section we instantiate the PP-MTA primitive using a signature scheme $(\mathtt{SIG.KeyGen}, \mathtt{SIG.Sign}, \mathtt{SIG.Verify})$, a collision-resistant hash function H, a vector commitment scheme ($\mathtt{VCO.Setup}$, $\mathtt{VCO.Com}$, $\mathtt{VCO.Open}$, $\mathtt{VCO.Ver}$), a ZK-SNARK scheme ($\mathtt{SNK.Setup}$, $\mathtt{SNK.Prove}$, $\mathtt{SNK.Ver}$), and a secure-channel establishment protocol : ($\mathtt{AKE.KGen}$, $\mathtt{AKE.Sce}$, $\mathtt{AKE.Enc}$, $\mathtt{AKE.Dec}$).

4.4.1 Setup

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

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

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

$$(\mathsf{CRS}, \tau) \leftarrow \mathtt{SNK.Setup}(R)$$

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

$$\texttt{SNK}\{(\mathsf{rpt}, \sigma) : \texttt{SIG.Verify}(\mathsf{AK.pk}, \mathsf{rpt}, \sigma, c) == 1 \ \wedge \ \mathsf{rpt}.\mathcal{H}.\mathsf{Conf} \in \mathcal{CONF}\}$$

we fix the statement :

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

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

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

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

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

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 $\{\mathcal{T}.\mathsf{pk}, \mathcal{T}.\mathsf{sk}\} \leftarrow \texttt{MTA.TKGen}(\mathsf{ppar})$ which are, in fact, AKE keys:

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

4.4.2 Registration

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

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

As depicted in Figure 4.4, when a registration request is made, $\mathcal{H}$ and $\mathcal{T}$ run $\texttt{AKE.Sce}$ (using their long-term credentials) to establish a secure channel, over which they can communicate in a confidential and authentic manner. Over this secure channel, $\mathcal{T}$ requests the registration of a VM with description VMdesc. The hypervisor verifies it can still allow tenants to register a new VM (w.r.t.N_{VM}). If this is not so, the algorithm aborts. Otherwise, the hypervisor generates attestation (signature) keys for the newly-registered VM: $(\mathsf{VAK.sk}, \mathsf{VAK.pk}) \leftarrow \texttt{SIG.KeyGen}(\mathsf{ppar})$. The keys $\mathsf{VAK.sk}$ and $\mathcal{T}.\mathsf{pk}$ will be stored

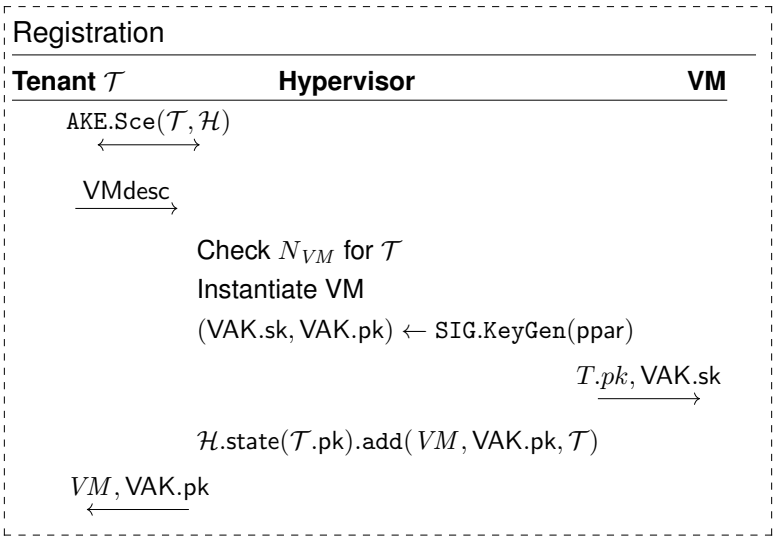


Figure 4.4: Registration, the tenant (successfully) registers a new VM of description VMdesc.

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

4.4.3 Hypervisor Attestation

We instantiate this algorithm as $\{\text{ATT}_{\mathcal{H},\mathcal{T}}\} \leftarrow \text{MTA.HAtt}\langle\mathcal{T}(\mathcal{T}.\text{sk}, \text{nonce}_{\mathcal{T}}), \mathcal{H}(\mathcal{H}.\text{sk}, \text{AK.sk}, \mathcal{H}.\text{state}, \mathcal{H}.\text{Conf})\rangle$. The idea is to attest (in a configuration-hiding way) the hypervisor and also to embed in that attestation elements that characterise each managed VM. During hypervisor attestation, the latter retrieves the public attestation-key stored on each of the vTPMs it manages[2]. Those values are concatenated with the nonce and hashed to obtain a new nonce.

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

Preparation of vector commitment. Multiple tenants (authenticated over a secure channel) may request attestations simultaneously, each providing a nonce to the hypervisor. The hypervisor randomly associates each tenant with an index $i \in \{1, \dots N_{\mathcal{T}}\}$. It then retrieves the VAK.pk of all the VMs registered by the tenant(s) requesting an attestation and concatenates, for each tenant, the nonce that tenant provided and the VAK.pk

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

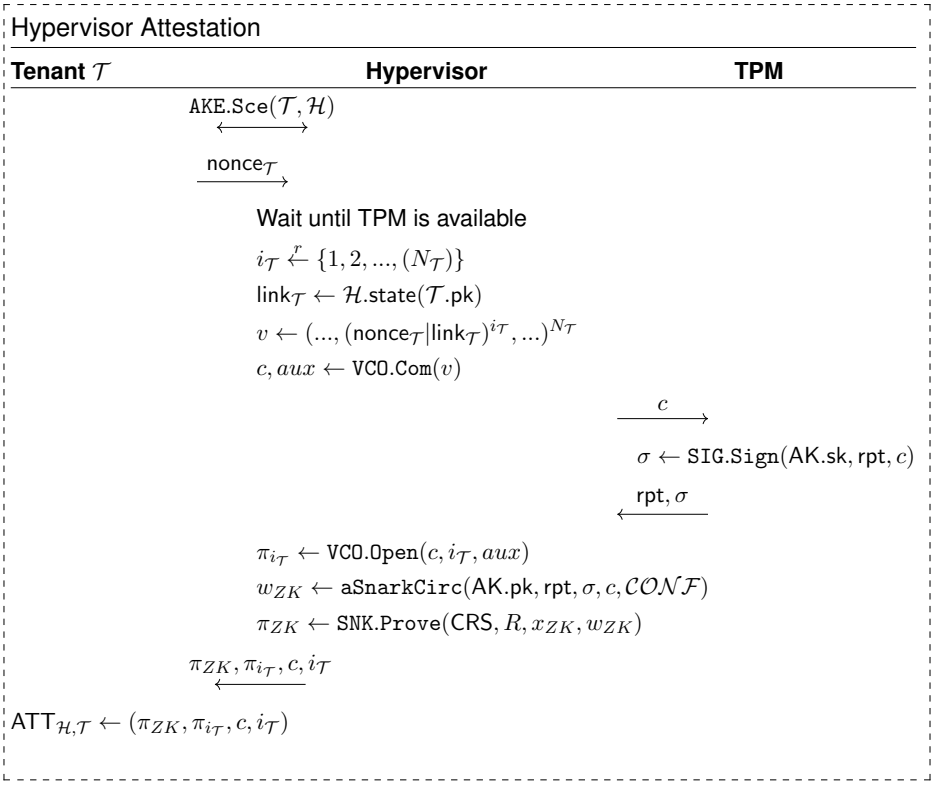


Figure 4.5: Hypervisor Attestation, only the i-th tenant is represented but we can see the aggregation of all the nonces through the commitment which allows for a single TPM operation.

of all the VMs the latter owns. The list of VAK.pk per tenant constitutes its linking information (link in Figure 4.5). If less than $N_{\mathcal{T}}$ tenants request an attestation, empty positions in the vector are filled with random values, and the commitment is always of constant size. The vector commitment must be hiding, as tenants should learn no information about positions they will (later on) be unauthorized to open.
Our scaling approach provides scalability. Say that two or more tenants request hypervisor attestations while the TPM is busy. Without nonce-aggregation, those requests would be treated separately. Instead, aggregation allows the hypervisor to generate a single attestation that can be provided to all the tenants and *still* hide everything except the content relevant to the tenant itself.

Hypervisor attestation. The next step is to obtain an attestation report from the TPM. This communication is on the physical device (hidden from tenants). The hypervisor submits to the TPM the commitment c *in lieu of* an attestation nonce. The TPM computes an attestation report rpt and a signature σ on it with the private attestation key AK.sk associated to the hypervisor.

Proof of attestation. The hypervisor, having received the report and signature com-

putes a proof of ownership of a valid attestation. Note that the attestation report (and corresponding signature) reveal the configuration of the hypervisor, which we want to hide from the tenants. Thus, the hypervisor proves that it has a valid attestation from the TPM for a configuration within the set $\mathcal{CONF}$, with respect to nonce c, *i.e.*, it needs to compute:
$\texttt{SNK}\{(\mathsf{rpt}, \sigma) : \texttt{SIG.Verify}(\mathsf{AK.pk}, \mathsf{rpt}, \sigma, c) == 1 \wedge \mathsf{rpt}.\mathcal{H}.\mathsf{Conf} \in \mathcal{CONF}\}$
We can compile this computation into an arithmetic circuit. Then, a ZK-SNARK will allow the hypervisor to prove it has run this algorithm for some public set $\mathcal{CONF}$, the nonce c, with respect to AK.pk, and that the algorithm output 1, all this without revealing the report rpt nor the signature σ.

Algorithm 1 The snark circuit

```
procedure aSnarkCirc(AK.pk, rpt, σ, c, CONF)
    if SIG.Verify(AK.pk, rpt, σ, c) == 1 and rpt.H.Conf ∈ CONF then
        return 1
    else
        return 0
    end if
end procedure
```

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

4.4.4 VM Attestation

This algorithm $\{\mathsf{ATT}_{VM}\} \leftarrow \texttt{MTA.VMAtt}(VM(\mathsf{VAK.sk}, \mathsf{VAK.pk}), \mathcal{T}(\mathcal{T}.\mathsf{sk}, \mathsf{nonce}))$ (Figure 4.6) generates an attestation report that only the tenant owning the VM can actually see and verify. The tenant and VM run an AKE protocol to establish a secure channel, over which $\mathcal{T}$ requests an attestation and forwards a nonce. The VM retrieves the linking information (VAK.pk) and concatenates it with the nonce to obtain a value later hashed to lkaux. Then, the VM requests a signed report for lkaux and forwards the response to the tenant over the secure channel.

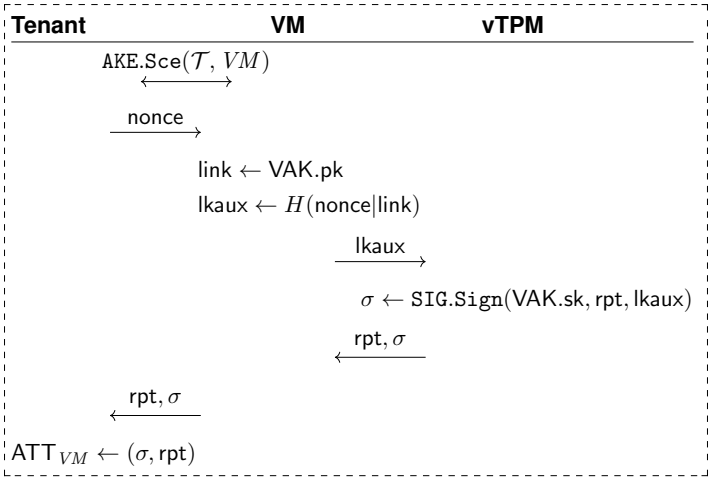


Figure 4.6: VM Attestation

4.4.5 Verification

Hypervisor attestation verification. We instantiate the algorithm $\{0,1\} \leftarrow$ MTA.VerHAtt(ATT$_{\mathcal{H},\mathcal{T}}$, nonce$_{\mathcal{T}}$, link$_{\mathcal{T}}$) as depicted in Figure 4.7. The tenant opens the commitment c at the relevant index,then checks that it opens to the concatenation of the nonce nonce$_{\mathcal{T}}$ and linking information link$_{\mathcal{T}}$ (if this fails, the algorithm outputs 0). Then the tenant verifies the SNK proof and outputs 1 if both verifications succeed.

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

MTA.VerHAtt(ATT$_{\mathcal{H},\mathcal{T}}$, nonce$_{\mathcal{T}}$, link$_{\mathcal{T}}$) $\rightarrow \{0,1\}$:

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

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

SNK.Ver(CRS, R, x_{ZK}, π_{ZK})

If all verification pass output 1, otherwise 0

(a) Hypervisor

MTA.VerVMAtt(ATT$_{VM}$, nonce, link) $\rightarrow \{0,1\}$:

Parse ATT$_{VM}$ as (rpt, σ)

lkaux ← H(nonce|link)

SIG.Verify(VAK.pk, rpt, σ, lkaux)

If all verifications work output 1, otherwise 0

(b) VM

Figure 4.7: Verification

4.4.6 Linking attestations

The link algorithm $\{0,1\} \leftarrow$ MTA.Link(ATT$_{\mathcal{H},\mathcal{T}}$, nonce$_{\mathcal{T}}$, link$_{\mathcal{T}}$, ATT$_{VM}$, nonce, 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 reports 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.

$\underline{\texttt{MTA.Link}(\mathsf{ATT}_{\mathcal{H},\mathcal{T}}, \mathsf{nonce}_{\mathcal{T}}, \mathsf{link}_{\mathcal{T}}, \mathsf{ATT}_{VM}, \mathsf{nonce}, \mathsf{link}) \rightarrow \{0,1\}:}$
$\texttt{MTA.VerHAtt}(\mathsf{ATT}_{\mathcal{H},\mathcal{T}}, \mathsf{nonce}_{\mathcal{T}}, \mathsf{link}_{\mathcal{T}})$
$\texttt{MTA.VerVMAtt}(\mathsf{ATT}_{VM}, \mathsf{nonce}, \mathsf{link})$
Check $\mathsf{link} \in \mathsf{link}_{\mathcal{T}}$
If all verifications work output 1, otherwise 0

4.5 Security Analysis

In this section we show that our construction guarantees inter-tenant privacy, configuration hiding, and layer-linking. To prove that our scheme provides inter-tenant privacy we first introduce a special property of AKE schemes, namely Parter-hiding, and shows that TLS 1.3 guarantees in section 4.5.1. In section 4.5.4 we also define a new property for vector commitment in order to prove the linking security of our construction.

4.5.1 Partner-hiding AKE

Our solution requires secure channels, constructed from AKE schemes. Indeed, consider the VM attestation algorithm from Section 4.4. The tenant and VM use a mutually-authenticated AKE protocol 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. The protocol runs in *sessions*. Each P is associated with a tuple of long-term parameters $(\mathsf{sk}, \mathsf{pk})$ and each instance

keeps track of the following *attributes*:

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

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

We define Partner-Hiding in terms of an adversary $\mathcal{A}$ that is a Person-in-the-Middle. The security game will, in a nutshell, guarantee that an adversary is unable to tell the difference between an interaction with a real, uncorrupted party, and an interaction with a simulator (which only has access to the security parameter, but not to any of the private keys generated in the game).

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

- $\pi_{\mathsf{P}}^{i} \leftarrow \mathsf{oNewSession}(\mathsf{P}, \mathsf{Q}, \mathsf{role})$: the (honest) session creation oracle will initiate a new instance π_{P}^{i} with partner identifier $\pi_{\mathsf{P}}^{i}.\mathsf{pid} = \mathsf{Q}$, such that π_{P}^{i} plays the role designated by role (either initiator or responder) in its session.
- $m^* \leftarrow \mathsf{oSend}(\pi_{\mathsf{P}}^{i}, m)$: the (honest) sending oracle models sending message m to an already-existent instance π_{P}^{i}. It is expected that π_{P}^{i} 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).
- $k \leftarrow \mathsf{oReveal}(\pi_{\mathsf{P}}^{i})$: the revelation oracle allows the adversary to learn already-established session keys k.
- $\pi_{\mathsf{P}}^{i} \leftarrow \mathsf{oNewSession}_{b,\mathsf{role}}(\mathsf{P}, \mathsf{Q})$: this is the left-or-right version of the 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 P, with partner identifier Q, such that P will have a role as either the initiator or the responder of the session. On the i-th call to the oracle $\mathsf{oNewSession}_{b}(\mathsf{P}, *)$, the created instance will be indexed as π_{P}^{i}. The oracle forwards the handle π_{P}^{i} to the adversary. If $b = 1$, the oracle call is forwarded to a

simulator Sim, which is only given the security parameter, but no party information.

- $m^* \leftarrow \mathsf{oSend}_b(\pi^i_{\mathsf{P}}, m)$: this left-or-right version of the sending oracle allows the message m to be either forwarded to π^i_{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 $\mathsf{P} \in \mathbb{P}$. The adversary receives all the public keys, whereas the challenger keeps track of all the private keys. The simulator will be given no information at all, apart from the security parameter.
There are two phases to the game. In the learning phase, the adversary will use the honest session-creation and sending oracles, as well as the session-key revelation oracle, in order to observe honest sessions and interact with the honest parties.
In the second phase of the game, the adversary gains access only to the left-or-right instance-creation and sending oracles. We distinguish between the two following notions:

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

Finally, the adversary will be allowed a final learning phase, identical to the first one. When the adversary is ready to end the game, it will output a bit d, which will be its guess for the bit b used by the challenger during the challenge phase.
It should be noted that at the transition to each new phase, all ongoing sessions are aborted.

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

Figure 4.8: The initiator-hiding game.

Definition 4.5.1 (Initiator-Hiding security)**.** *Consider an authenticated key-exchange protocol* AKE. *This protocol is* (N_{P}, $q_{\mathsf{oNewSession}}$, $q_{\mathsf{oNewSession}_{b,\mathsf{Role}}}$, ϵ)-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* $\mathsf{Adv}^{\mathsf{IHide}}_{\mathsf{AKE}}(\mathcal{A}) := \left|\Pr[\mathcal{A} \text{ wins } G_{\mathsf{InitHide}}(\lambda, N_{\mathsf{P}})] - \frac{1}{2}\right|$*, 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 4.5.1. *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 (sketch). We first note that previous work [48] 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 π^i_{P} (even if the simulator himself does not actually know the instance is supposed to belong to 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 [3].

For our proof, we make the following game hops:

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

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

$\mathbb{G}_2$: the same as $\mathbb{G}_1$, except that we abort if in any two of the sessions created, the hash over the Client and Server Hellos coincide. As per $\mathbb{G}_1$, at least one input is unique in each session, notably the client's input. As a result, the two games are identical except if the content signed in the CertificateVerify message features a 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. The guess is correct up to $\frac{1}{N_{\mathsf{P}}}$. 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 $N_{\mathsf{P}} - 1$ parties, but not for the one party that we denote 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 $\mathsf{P} \neq \mathsf{P}^*$ which was created by an $\mathsf{oNewSession}_{b,\mathsf{Initiator}}(\mathsf{P}, \mathsf{P}^*)$ query (*i.e.*, P^* is the expected partner identifier of

[3] 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.

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 $\text{oNewSession}_{b,\text{Initiator}}$ 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 $\frac{1}{2}$.

□

4.5.2 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 tenants 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 4.5.1 (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:*

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

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 (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 $\mathtt{MTA.HAtt}$ algorithm, rather than the oHAttest oracle), the attestation report is replaced by an error symbol $\perp$. The adversary can only distinguish between the two if the adversary manages to impersonate an honest tenant. The reduction will first guess which tenant the attacker will target (losing a factor $N_{\mathcal{T}}$), hence giving us a loss equalling the second term of the bound above.
- $\mathbb{G}_3$: Identical to $\mathbb{G}_2$, except that, at the first oHAttest query from the adversary, the challenger replaces the correct attestation report 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 report is sent.
- $\mathbb{G}_4 \;\rightarrow \mathbb{G}_{2+q_{\mathsf{oHAttest}}}$ In each game $\mathbb{G}_{2+i}$, for $i \in \{2, \ldots, q_{\mathsf{oHAttest}}\}$, we proceed as in $\mathbb{G}_3$ for the i-th oHAttest query. At each time we lose a term $\mathsf{Adv}^{\mathtt{ACCE-SC}}_{\mathtt{AKE}}(\mathcal{B}_3)$.
- $\mathbb{G}_{3+q_{\mathsf{oHAttest}}}$: 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 $\mathtt{MTA.HAtt}$ algorithm rather

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

$\mathbb{G}_{4+q_{\text{oHAttest}}} \rightarrow \mathbb{G}_{2+q_{\text{oHAttest}}+q_{\text{HAttest}^*}}$: Proceed to modify, in game $\mathbb{G}_{2+q_{\text{oHAttest}}+i}$ the vector commitment for the i-th hypervisor attestation demand, for $i \in \{2, 3, \ldots q_{\text{HAttest}^*}\}$ 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 bound.

□

Limitations to our guarantee. Our security model and proof hold 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.

4.5.3 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 machine 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 hypervisor attestation (which is generated by the physical TPM). The hypervisor receives a signed report from the TPM (this communication takes part within the machine itself), then forwards a *proof* that it is in possession of a signed report, which is consistent with a configuration $\mathcal{H}.\mathsf{Conf} \in \mathcal{CONF}$. In particular, the

hypervisor computes (and later sends) $\mathtt{SNK}\{(\mathsf{rpt}, \sigma) : \mathtt{SIG.Verify}(\mathsf{AK.sk}, \mathsf{rpt}, \sigma, c) == 1 \wedge \mathsf{rpt}.\mathcal{H}.\mathsf{Conf} \in \mathcal{CONF}\}$.
Our proof is straight-forward: by the zero-knowledge property of the ZK-SNARK, no information is revealed about rpt 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 4.5.2 (Configuration privacy)**.** *Let* PP-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* PP-MTA *with advantage* $\mathsf{Adv}^{\mathsf{CPriv}}_{\mathsf{PP\text{-}MTA}}(\mathcal{A})$*, then there exists an adversary $\mathcal{B}$ such that:* $\mathsf{Adv}^{\mathsf{CPriv}}_{\mathsf{PP\text{-}MTA}}(\mathcal{A}) \leq q_{\mathsf{oHAttest}} \cdot \mathsf{Adv}^{\mathsf{ZK}}_{\mathsf{NIZK}}(\mathcal{A})$ *where q_{oHAttest} is the number of queries $\mathcal{A}$ makes to the hypervisor-attestation algorithm.*

Proof sketch. We use a hybrid argument, replacing in each game hop the true attestation rpt 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 $\mathsf{Adv}^{\mathsf{ZK}}_{\mathsf{NIZK}}(\mathcal{B})$. 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. □

4.5.4 Collision Resistant Vector Commitment

The goal of a malicious hypervisor could be to find a collision by keeping the values of a given tenant with its correct index but modifying the other positions to find a collision with a previous committed vector. Another attack could also occurs if the nonce of a given tenant matched a previous nonce. In that case, the hypervisor could simply send the old attestation with the same proof of membership in the vector. Finally, the malicious hypervisor could also compute a collision which occurs with probability $\frac{q}{2^{n+1}}$ for q attestation queries and vector commitment of size n.
We thus define the collision resistance property for vector commitment which will allow us 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, $G_{\mathsf{CR}}(\lambda)$ (see Figure 4.9), to define the collision

resistance of vector commitment. This game is similar to the collision resistance of hash functions: the challenger computes the setup algorithm to send the public parameters to $\mathcal{A}$ which outputs two (different) vectors. We say that $\mathcal{A}$ wins the game if and only if the commitments are equal.

Game $G_{\mathsf{CR}}(\lambda)$
$\{\mathsf{ppar}\} \leftarrow \mathtt{VCO.Setup}(1^\lambda, n)$
$(v, v') \leftarrow \mathcal{A}(\mathsf{ppar})$
$\mathcal{A}$ **wins** iff.: $\exists i$ such that $v[i] \neq v'[i]$ and
$\mathtt{VCO.Com}(v) = \mathtt{VCO.Com}(v')$

Figure 4.9: The collision resistance game for vector commitments.

Definition 4.5.2 (VCO collision-resistance)**.** *We say that* VCO *is* $(\lambda, n)-$collision resistant *if for all adversary* $\mathcal{A}$*, the probability of winning game* $G_{\mathsf{CR}}(\lambda)$ *is negligible.*

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

Lemma 4.5.2. *A* VCO *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 VCO and show that we can construct B, using $\mathcal{A}$ as a subroutine, winning the collision resistance of H.
If such an adversay $\mathcal{A}$ exists, then 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. □

4.5.5 Linkability Security

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

Theorem 4.5.3 (Linkability security)**.** *Let* PP-MTA *be a multi-tenant attestation scheme; this scheme provides linkability security if: the hash function* H *and* VCO *are collision resistant, the* SNK *is sound, and the signature scheme* SIG=*(*SIG.KeyGen,SIG.Sign,SIG.Verify*) 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:

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

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 considers 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.*, `MTA.Link` returns 1) is to forge an attestation from the hypervisor.

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

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

$\mathbb{G}_1$: The challenger guesses parties P and Q outputted by $\mathcal{A}$. There are two platforms composed each of one hypervisor and N_{VM} VMs. The probability of a correct 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 $\mathsf{Adv}^{\mathsf{Coll}}_{\mathsf{VCO}}(\mathcal{B}_2)$.
At this point, the only way for the adversary to win the game is to forge an attestation for P or Q. The next games attempt to rule out the fact that $\mathcal{A}$ outputs $\mathsf{ATT}_{\mathsf{P}}$ (or $\mathsf{ATT}_{\mathsf{Q}}$) as an attestation that corresponds to a value stored in $\mathcal{L}_{Att}$. The games $\mathbb{G}_4$, $\mathbb{G}_5$ and $\mathbb{G}_6$ ensure that 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 $N_{VM} \cdot \mathsf{Adv}^{\mathsf{VCBind}}_{\mathsf{VCO}}(\mathcal{B}_3)$.

$\mathbb{G}_5$: 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 $\mathsf{Adv}^{\mathsf{ZK}}_{\mathsf{NIZK}}(\mathcal{B}_4)$.

$\mathbb{G}_6$: We ensure that the adversary cannot forge a signature of the report, 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 Q. At this point the adversary cannot win the game.

□

4.6 Implementation

We provide a proof-of-concept implementation of the scheme described in Section 4.4 in Python, with some parts related to the ZK-SNARK written in Rust. We used this implementation to design benchmarks and evaluate the performance of the scheme. Specifically, we focused on the performance of the VM attestation and hypervisor attestation compared to a traditional attestation. We also demonstrate the scaling properties of our scheme experimentally.

4.6.1 Implementation and experiment details

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

TPM libraries. To communicate with the (physical or virtual) TPM, we used the software provided by tpm2-software community [41], relying on the 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 [49] combined with a basic hash-based commitment scheme.

SNARK. We used bellperson [50] which implements a preprocessing circuit-specific crs snark [51] 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 [52]. a library to compute arbitrary-precision arithmetic operations inside SNARKs.

SNARK Circuit design. The circuit computes an RSA PKCS#1 v1.5 signature verification and set membership verification. We implemented the RSA verification for a 2048-bit modulus and a fixed exponent of 0x10001. Moreover, the circuit is designed for a nonce of length 32. Our implementation also assumes that the TPM will use sha256 as its hash algorithm. Hence, the size of the attestation report is fixed and thus the number of con-

straints in the circuit only depends on the size of the configuration set. The CRS will only need to be recomputed for a different size set (if using a circuit-specific-CRS SNARK).

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

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

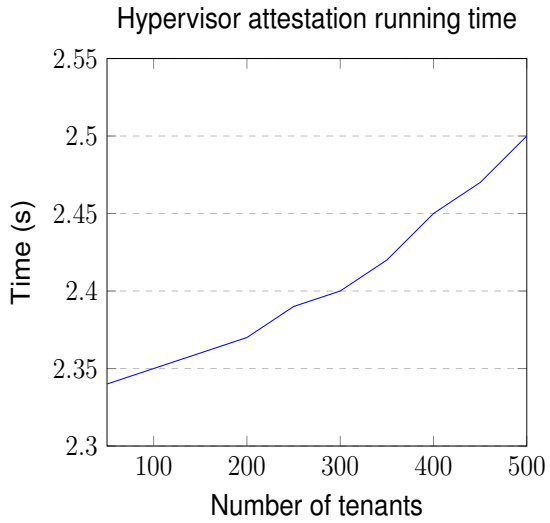
Experimental method. The Rust part of the code was measured using the Criterion library [53] which run the function to benchmark 100 times after a warm-up phase, then compute statistics over these samples. For the Python code we implemented a custom decorator, which works in a similar fashion and provides statistics over these runs.

4.6.2 Benchmark results

We now present our results. First we compare a traditional attestation with our privacy preserving hypervisor attestation and we show how our scheme scales with the number of tenants. This experiment was carried out using a set of possible configuration of size 128. Hence we provide measurements of the SNARK computation for different set sizes to show this aspect impacts the performances of our construction. Finally we provide some results concerning the linking procedure proving that layer linking is achieved for a nearly negligible cost.

Attestation and scaling. Table 4.10b compares a traditional attestation and our hypervisor attestation. We can see that the ZK-SNARK adds a significant overhead but the total time is still low enough (2.4s) for practical use. This is especially true with a high number of tenants requests (like in 5G and beyond). With a classic sequential processing of the requests it would take minutes to answer all the tenants whereas in our case despite having a relatively high base computing time, the attestation scales very well as depicted on Figure 4.10a.

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



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

Attestation

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

Verification

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

(b) Time to perform attestation

Figure 4.10: Benchmarks

our SNARK circuit. Table 4.1 shows how the setup, prover, and verifier algorithm performances change with set size.

Table 4.1: Performance variations of the ZK-SNARK in the configuration-set size. Median value over 100 samples.

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

Linking. Table 4.2 presents our measurements of the time required for the linking of a VM and a hypervisor attestation report. Note that the measurements below do not correspond to the full algorithm presented in section 4.4.6 but only include the verification of the linking information. The linking information inside the hypervisor attestation depends on the number of VM owned by the tenant, which impacts the overall performance. Even in spite of such variations, our scheme provides very fast, easy linking.

Table 4.2: Time to perform linking depending on the number of VM hosted by the hypervisor.

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

4.7 Conclusion

In this work, we have proposed a scalable and efficient TPM attestation scheme for multi-tenant environments. Our scheme does not require any modification of the TPM or unrealistic trust assumptions (e.g., attestation proxy). It provides strong privacy for both tenants and the hypervisor, and guarantees layer binding.
Our scheme achieves privacy by relying on vector commitment and ZK-SNARK. The latter primitive incurs a relatively high overhead, but it remains stable even with a drastically high number of attestation requests (which is the case in multi-tenant environments such as 5G) without requiring any TPM modification. Furthermore, if in some cases the configuration hiding property is not needed but we still have a high number of attestation requests due to the modularity of our construction, our scheme can still work efficiently by simply omitting the ZK-SNARK module.
Finally, note that this scheme could be simpler with a more powerful RoT. For example, consider the configuration privacy property. We have to rely on the expensive ZK-SNARK because the TPM cannot run an arbitrary algorithm of our choice on the attestation data. If the TPM were able to do this, configuration privacy could be ensured with a commit and prove method. Efficient algorithms exist for this commit-and-prove method, similar to those employed in the next chapter.

5

Collective Attestation in NFV

Contents

So far we have envisioned NFV as being similar to a cloud. A provider hosts VNFs owned and operated by various tenants, and working independently. While this approach corresponds to real use cases of NFV, operators can also combine multiple VNFs to provide a complete network service with an entry point and an end point as shown in Chapter 1. Therefore, instead of looking at individual VNFs, we want to verify the state of an entire network service by performing the deep attestation of each VNF. A trivial solution is to simply attest them sequentially. However, we can do better with collective attestation, as introduced in Section 1.2.4. In this chapter, we show how to adapt techniques from IoT collective attestation to render them suitable for the deep attestation of network services. This work has been accepted at NCA 2024 [4].

5.1 Introduction

The context of this work is illustrated in Figure 5.1. We consider a forwarding graph (FG) consisting of VNFs owned by different entities, which are colour-coded: blue VNFs are owned by a blue tenant, green VNFs belong to the green tenant, and so on. The VNF-FG has an entry point (shown on the left) and an end point (shown on the right). In addition to the owners of the various VNFs, an additional party represented in Figure 5.1 is an operator who manages the VNF-FG (and may also own VNFs and parts of the NFVI). We consider a fixed network topology ensuring network stability during each attestation. Although this assumption is unrealistic for IoT swarms, it is reasonable in the NFV use case, where the VNF-FG must remain online during communication. We also assume that each VNF and hypervisor are equipped with a root of trust.

In this context, the operator may request the attestation of the entire swarm either in order to verify the state of the VNF-FG, or as part of a service provided to users of the forwarding graph. The challenge is to prove that the VNFs of the graph are in a valid state, but in a way that preserves the privacy of the exact configuration of each VNF and hypervisor.

To be sure that a forwarding graph is in a valid state, it is important to attest each VNF, but also the underlying hypervisor, hence the need for deep attestation. However, deep attestation may not be sufficient, as we also need to ensure that each VNF is running on the correct hypervisor and has not been moved, hence the need for linking DA.

Finally, as in any multi-tenant environment, we want to preserve the privacy of the VNF configuration.

Clearly, a naïve solution to providing the attestation of a VNF forwarding graph with the properties we want is to sequentially attest each VNF using a privacy-preserving

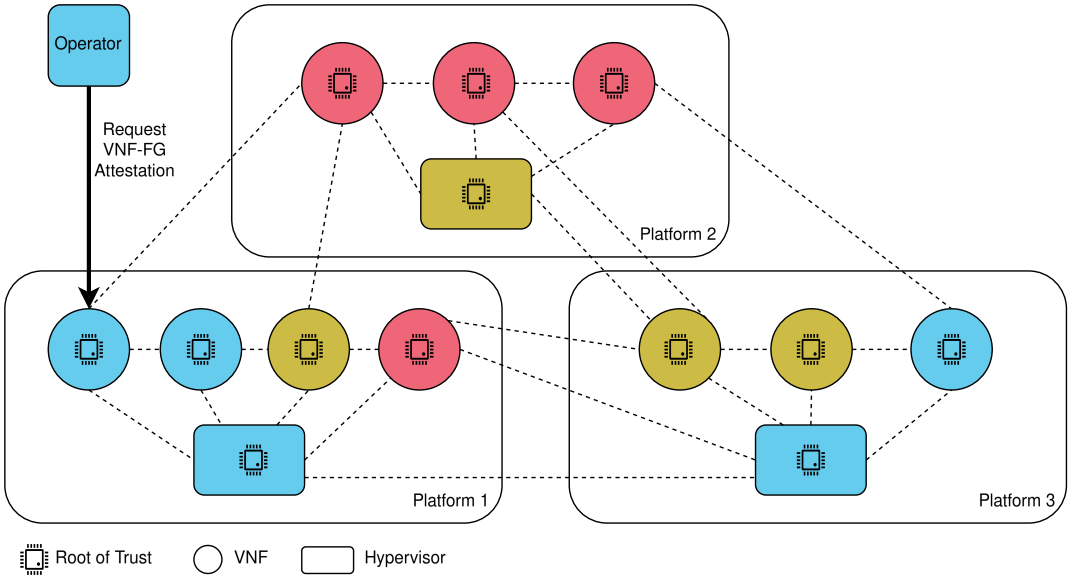


Figure 5.1: A multi-tenant and multi-domain VNF-FG.

deep attestation scheme. However, this is inefficient in terms of attestation computation time, attestation verification time, and bandwidth usage. Our goal is to do better by using collective remote attestation methods while remaining privacy-preserving and enabling deep attestation with linking.

Our Contribution. To meet these challenges we designed a new protocol **D-cRA**, which allows to verify both the *valid state* of each component in a VNF-FG (VNFs and hypervisors), and the *layer-linking* between VNFs and hypervisors. In addition, D-cRA guarantees an essential *privacy property*: the hiding of specific state details (*i.e.*, the configuration) of VNFs and hypervisors in the VNF-FG. All that will be leaked upon attestation is that the configuration belongs to a *set* of valid configurations, typically chosen by the node itself, the tenant, the operator, or the vertical, or even all the entities together. Configuration hiding is a business must in multi-tenant environments, and also reinforces Internet openness as recently pointed out by the IETF: " *Allowing clients to use a variety of software as long as it is protocol-compliant is an essential part of the IETF development process and the openness of the Internet.*" [54].
Our protocol relies on the use of spanning trees, which allow nodes to transmit attestation requests and collect attestations over a VNF-FG. Each node corresponds to a component (*i.e.*, VNF or hypervisor), and it must attest to its parent nodes. The latter verify the attestations and cumulate the results of those verifications into binary aggregates over all the child nodes. If all the child-node attestations are valid, then the parent's aggregate is set to 1; else, it will be set to 0. The use of intermediate

and unforgeable verification of attestations and propagation through binary aggregates makes large group attestations efficient.
An important aspect of our scheme, is that node attestations do not actually feature the node's current state, but rather a proof, in zero-knowledge, that the state belongs to a public set of valid states. This *configuration-hiding* property is guaranteed in our protocol by the use of zero-knowledge set membership proofs. Another important contribution of this work is formalizing the privacy and linking properties that collective remote attestation must achieve and rigorously proving that D-cRA reaches them.
Finally, we empirically evaluate the performance of D-cRA using an implementation using OMNeT++ [55]. The results show that a VNF-FG attestation time is logarithmic in the number of involved nodes and the attestation of thousands of nodes does not exceed $200ms$. Thus, our privacy-preserving layer-linking collective-attestation solution achieves strong privacy while remaining highly competitive in practice.

5.2 Technical Overview

Like the first IoT cRA protocols [22], [23], in our protocol D-cRA, we use a spanning tree to efficiently attest the nodes of a VNF-FG. Each forwarding-graph will be associated with a spanning tree that has, at its origin, the point of entry of the VNF-FG (also called a root node). However, contrarily to previous collective remote attestation approaches (which can efficiently attest VNFs but omit hypervisors), we consider not only VNFs but also their associated hypervisors[1]. Our decision to include hypervisors in the VNF-FG model emanates from our proposed goal of guaranteeing layer-linking: a strong property, which ensures that attestation not only provides proof as to the lack of tampering on nodes such as hypervisors and VNFs, but additionally links VNFs to the hypervisors managing them. This decision (including hypervisors in the VNF-FGs) does not simply imply a complication in terms of increasing the size of the forwarding-graph. Indeed, in NFV structures, hypervisors may manage a number of VNFs and this relationship will need to be translated to the graph's structure. Should hypervisors be direct parent-nodes of the managed VNFs? This might be inefficient, as the structure of the graph will impact the optimal spanning-tree size and layout. The result is then a strong structural guarantee of the VNF-FG, which guarantees the deep attestation of the underlying infrastructure.

[1]Notice that in the ETSI definition of a VNF-FG, the term *node* corresponds to a VNF. In the rest of this chapter we will use the term node to refer to either a VNFs or a hypervisors. We also assume that both VNF and hypervisor are equipped with a Root of Trust (RoT) that can be physical or virtualized.

Note 5.2.1: Why Tree?

Tree based cRA use a tree to distribute the attestation and to aggregate the results upon return. This simple and very efficient method (still the most efficient approach as of today), only works with a static network. In our case, we consider static network for the time of attestation. In fact, in NFV the VNF-FG remain unchanged beyond the interval of attestation: as described in Chapter 1, a VNF-FG is associated with a graph descriptor (stored in the MANO) that fixes the topology of such a graph in the long run. This allows us to consider a complex one-time setup, where VNFs and hypervisors are initialised with no need for a dynamic join or update protocol. Though counter intuitive, this way of using VFN-FG does not go against the dynamic nature of NFV; indeed, the operator can always add or remove services by adding and removing VNF-FG.

Keys, configurations, linking. Similarly to spanning tree based IoT collective attestations, we assume that, before attestation can take place, each node of the VNF-FG is initialized in a trusted way by the operator during an offline phase. Subsequently, attestation can take place during potentially many online phases. During setup, each node generates its cryptographic attestation credentials $(\mathsf{sk}, \mathsf{pk})$ (the secret key is kept into the RoT). In addition to non privacy-preserving schemes, we explicitly consider the notion of node *configuration*, which essentially specifies the exact setup of the VNF or hypervisor, and which the owner of the VNF or hypervisor might want to keep private from neighbouring, potentially semi-trusted nodes. Thus, in addition to the keypair meant for attestation, each node shares a *set* $\mathsf{confset}$ of potential configurations rather than the single configuration being actually adopted in practice. Thus, contrary to the common, non privacy-preserving approach of [22], [23], we require attestation in VNF-FGs to guarantee *configuration-hiding*, a privacy property similar to the one we introduced in Chapter 4. Finally, our scheme will require attestations to be linkable, like in our two previous schemes. As a result, during setup, each node generates and transmits its linking information link, which in the case of a VNF will be a single public key, and in the case of a hypervisor will be a set of public keys corresponding to the VNFs public keys associated with it. This will enable our scheme to guarantee layer-linking, a property essential in virtualized infrastructures like multi-tenant VNF-FGs, but which is not relevant in an IoT context.

VNF-FG attestation. The online phase of our approach is the actual collective remote attestation protocol, which can be triggered on demand by the operator that uses the VNF-FG as an underlying support for its services. The operator triggers the computation

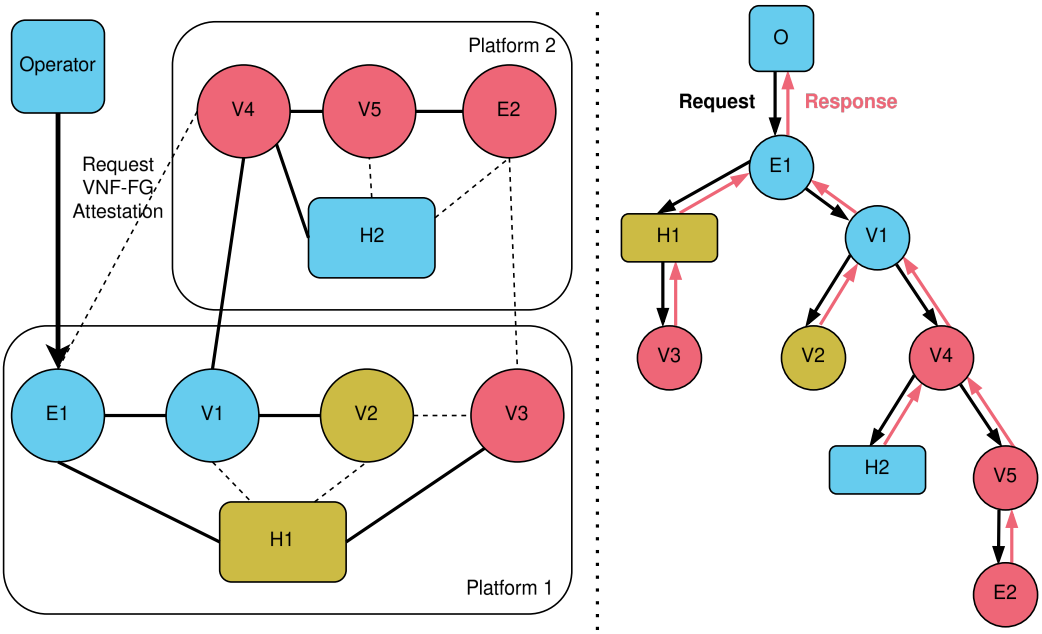


Figure 5.2: VNF-FG to an attestation tree.

of the spanning tree rooted at the entry point of the VNF-FG, over all the forwarding-graph nodes. Then, the operator can broadcast an attestation request starting from the root (VNF-FG entry point), which is forwarded by each node to its child nodes. The response will be forwarded in reverse order, from child to parent nodes and all the way back to the root. This is illustrated in Figure 5.2.

A basic approach towards remote attestation would enable nodes receiving an attestation request from a parent node to compute an attestation of the node's current configuration. If, moreover, layer-linking is required, then that can be achieved by including linking information by the method we provided in Chapter 3. However, here we also consider the *privacy* of node-configurations.

Thus, in our protocol, when a node receives the request req (which includes an attestation nonce), the node's Root of Trust retrieves the node's configuration. The RoT computes and sends an attestation (to the parent node), consisting of 3 elements: (1) a hiding commitment c of the retrieved configuration; (2) a zero-knowledge proof π that c is the commitment of a configuration among the set of valid configuration $\mathsf{confset}$, without revealing the configuration itself; (3) a signature σ over the commitment, the nonce (received in the request), and the node linking information.

When a node receives an attestation report, it proceeds as follows: it first retrieves the set of configurations and the linking information associated with the node which sent the attestation. Then it checks the proof π and signature σ under the expected linking

information and nonce. When a node has received and verified all the attestations from its child nodes, it proceeds to the aggregation of its verifications, in a SEDA-like way [22], which consists of simply setting a single bit to 1 if all the received attestations successfully verify and 0 otherwise. Finally, it computes and sends a signature over the aggregate and the nonce (received in the request req), as well as its own attestation (computed as described earlier), to the parent node. This process is repeated up the to the root node (the entry point of the VNF-FG).
To summarize, our scheme D-cRA enables (i) to efficiently compute a collective attestation by using a spanning tree and binary aggregation approaches, (ii) to verify the layer linking properties for VNFs and hypervisors due to sharing linking information at the offline phase and embedding it within attestations, and (iii) to prove the validity of a node configuration without sharing it thanks to set membership proofs.

5.3 Model

In this section, we formalize our scheme D-cRA and define the security and privacy properties that we want to achieve. Table 5.1 provides a summary of the used notations.

5.3.1 Syntax

We define a formal syntax for our privacy preserving collective remote attestation scheme D-cRA consisting of 10 probabilistic polynomial-time (PPT) algorithms. We first describe the algorithms of the *offline phase*, which are used to essentially set up the VNF-FG nodes. We subsequently focus on the algorithms making up the actual attestation protocol, which are run in an *online phase*.
Both in the syntax of our primitive and for the security model, we restrict the choice of forwarding graph to a set Γ, which essentially rules out disconnected graphs as well as those containing parallel links between nodes.

Offline Phase This phase consists of four algorithms:

$\underline{\texttt{cRA.Setup}(1^\lambda, \mathcal{G}) \rightarrow (\mathsf{ppar}, \mathcal{G}, \mathcal{T}_\mathcal{G})}$: The setup algorithm takes as input the security parameter λ (in unary), as well as a graph $\mathcal{G} \in \Gamma$. It outputs public parameters ppar, as well as the graph that was given in input and a spanning tree $\mathcal{T}_\mathcal{G}$ for that graph. The graph taken in input succinctly abbreviates the structure of the VNF-FG.

$\underline{\texttt{cRA.InitVerifier}(\mathsf{ppar}) \rightarrow (\mathsf{sk}_{\mathsf{ver}}, \mathsf{pk}_{\mathsf{ver}})}$: This algorithm initializes the verifier (the operator in our case) with a keypair consisting of the private key $\mathsf{sk}_{\mathsf{ver}}$ and corresponding

Table 5.1: Notation.

Generic Parameters	
λ	Security parameter
ppar	Public parameters
$\mathcal{G}$	Graph forming the VNF-FG
Γ	Set of possible graphs admitted by scheme
$\mathcal{T}_{\mathcal{G}}$	Set of all the possible spanning for the graph $\mathcal{G}$
n	Number of nodes in a graph $\mathcal{G}$
Verifier Parameters	
ver	Verifier handle
$\mathsf{sk}_{\mathsf{ver}}$	Verifier secret key
$\mathsf{pk}_{\mathsf{ver}}$	Verifier public key
$\mathsf{stg}_{\mathsf{ver}}$	Verifier storage
Node Parameters	
node	Node handle
sk	Node secret key
pk	Node public key
conf	Node current configuration
confset	Node list of possible valid configuration
link	Node's current linking information
stg	Storage of the node stored in the RoT
$\mathsf{pk}_{\mathsf{neib}}$	A neighbour's public key
$\mathsf{confset}_{\mathsf{neib}}$	A neighbour's set of configurations
$\mathsf{link}_{\mathsf{neib}}$	A neighbour's expected linking information
Attestation Parameters	
req	Attestation request
$\mathsf{rand}_{\mathsf{req}}$	Request nonce
σ_{req}	Request signature
a	Attestation report
c	Attestation commitment
σ_{a}	Attestation signature
π_{a}	Attestation ZKP
agg	Aggregate report
$\mathsf{res}_{\mathsf{agg}}$	Binary result of aggregation
σ_{agg}	Aggregate signature
Root Node Parameters	
$\mathsf{pk}_{\mathsf{root}}$	Root node public key
$\mathsf{confset}_{\mathsf{root}}$	Root node set of valid configurations
$\mathsf{link}_{\mathsf{root}}$	Root node linking information
$\mathsf{a}_{\mathsf{root}}$	Root node attestation
$\mathsf{agg}_{\mathsf{root}}$	Root node aggregate

public key $\mathsf{pk}_{\mathsf{ver}}$.

$\underline{\mathtt{cRA.InitNode}(\mathsf{ppar}) \rightarrow (\mathsf{pk}, \mathsf{sk}, \mathsf{confset}, \mathsf{link})}$: This initialization algorithm can be used to initialize any nodes of graph $\mathcal{G}$ (by setting it up as a VNF or hypervisor). The algorithm creates a keypair for the node and sets its configurations' set and linking information.

$\underline{\mathtt{cRA.InitStg}(\mathsf{ppar}, (\mathsf{node} \cup \mathsf{ver}), (\mathsf{pk}_{\mathsf{neib}}, \mathsf{confset}_{\mathsf{neib}}, \mathsf{link}_{\mathsf{neib}})_{\mathsf{neib} \in \mathcal{N}[\mathsf{node}, \mathcal{G}]}, \mathsf{pk}_{\mathsf{ver}}) \rightarrow \mathsf{stg}}$: The $\mathtt{cRA.InitStg}$ is a potentially interactive algorithm, which allows a node node in the graph $\mathcal{G}$ to be initialized with public parameters associated with their neighbours. We denote by $\mathcal{N}[\mathsf{node}, \mathcal{G}]$ the set of neighbours of the node $\mathsf{node} \in \mathcal{G}$. These public values are used for the verification of future attestations. The verifier also initializes its storage by getting information of the entry point node in the spanning tree.

Online Phase The attestation process is run through the online interaction of the following 6 algorithms :

$\underline{\mathtt{cRA.ReqGen}(\mathsf{sk}_{\mathsf{ver}}, \mathcal{T}_{\mathcal{G}}) \rightarrow \mathsf{req}}$: This algorithm is used by the verifier (*i.e.*, the operator) to create an attestation request. It also takes in input the tree description $\mathcal{T}_{\mathcal{G}}$ published as part of the public parameters, and outputs a request req, which must imperatively include an attestation nonce $\mathsf{rand}_{\mathsf{req}}$.

$\underline{\mathtt{cRA.Attest}(\mathsf{stg}, \mathsf{sk}, \mathsf{conf}, \mathsf{confset}, \mathsf{link}, \mathsf{req}) \rightarrow \mathsf{a} \cup \perp}$: This is the attestation algorithm, which will either abort, producing an error (if the attestation request req does not internally verify) or will eventually produce, for a nonce $\mathsf{rand}_{\mathsf{req}}$ included in an attestation request req, an attestation report a for a specific node (associated with the signature key sk). More precisely, if the request verifies, req is propagated further to the next nodes. In addition, the algorithm takes in input a private key sk, the node's current configuration conf, a set of valid configurations $\mathsf{confset}$ (it holds that $\mathsf{conf} \in \mathsf{confset}$), and some linking information link, as well as the nonce $\mathsf{rand}_{\mathsf{req}}$. The output is an attestation report a. In anticipation of our construction, notice that this attestation report will provide strong authentication and layer-linking properties.

$\underline{\mathtt{cRA.Verify}(\mathsf{stg}, \mathsf{a}, \mathsf{req}) \rightarrow \{0, 1\} \cup \perp}$: The attestation verification allows a node to verify an attestation for a request req which includes the nonce $\mathsf{rand}_{\mathsf{req}}$. The verification algorithm is stateful and the verifying node uses its internal storage stg[2] during verification to recover all the information needed for the verification process. If this algorithm outputs a result equal to 1, the verifying node will conclude that the node acting as the prover (and providing a) is in a *valid* state (*i.e.*, the node has a

[2] In the construction we give a concrete example of stg

valid configuration and correct linking information). By contrast, when the result is 0, the prover node is assumed to have been *tampered with* (*i.e.*, the node has an invalid configuration or improper linking information). The algorithm can also output $\perp$ if a node doesn't answer to its parent before a certain timeout.

$\texttt{cRA.Aggregate}(\mathsf{stg}, \mathsf{sk}, (\mathsf{a}_1, ..., \mathsf{a}_i), (\mathsf{agg}_1, ..., \mathsf{agg}_j), \mathsf{req}) \rightarrow \mathsf{agg}'$: This algorithm is used to aggregate a number of attestation reports a_i and/or a number of intermediate aggregates agg_j for a common request req including a nonce $\mathsf{rand}_{\mathsf{req}}$. It can aggregate both attestation and already aggregated results.

$\texttt{cRA.AggVerify}(\mathsf{stg}, \mathsf{agg}, \mathsf{req}) \rightarrow \{0, 1\} \cup \perp$: This algorithm is used to verify if an aggregate is valid for a nonce request req and a node's storage stg. If the aggregate is valid (the result of the verification is 1), then every node that has attested so far is assumed to have been in a valid state. Otherwise, it is assumed that at least one of the nodes contributing to the aggregate has failed to attest. Similarly to the attestation verification algorithm, the aggregate verification can output $\perp$ if a node which is expected to send an aggregate fails to do so before a given timeout.

$\texttt{cRA.Link}(\mathsf{stg}_{\mathsf{ver}}, \mathsf{a}_{\mathsf{root}}, \mathsf{agg}_{\mathsf{root}}, \mathsf{req}) \rightarrow \{0, 1\}$: The linking algorithm is the final algorithm of the online phase, run by the verifier. It takes in input an aggregate $\mathsf{agg}_{\mathsf{root}}$ and an attestation $\mathsf{a}_{\mathsf{root}}$ sent from the root node, along with the attestation request req (containing the nonce $\mathsf{rand}_{\mathsf{req}}$) and the storage of the verifier $\mathsf{stg}_{\mathsf{ver}}$. If the linking algorithm outputs 1, every node in the tree is assumed to be in a valid state *and correctly linked to the expected hypervisor/VNFs.*

Remark 5.3.1. *Notice that in practice the description of the VNF-FG comes from the NFV MANO (NFV MANagement and Orchestration), and thus the spanning tree of the graph will be computed honestly, outside the direct control of the verifier (and of potential adversaries). This will be reflected in our subsequent security games.*

5.3.2 Properties

In this section we define the properties that collective remote attestation should guarantee. We use our work of Chapter 3, which captured the basic provable-security properties of deep attestation, and employ a game-based provable-security approach, in which the adversary $\mathcal{A}$ plays a security game against a challenger $\mathcal{C}$. The attacker is given access to oracles which capture special capabilities such as the corruption of a VNF, or the choice of a convenient configurations for a particular node. The adversary can also effectively use any algorithms for which it knows the correct input values.

Unforgeability

Unforgeability is the basic property expected for an attestation scheme, namely that no adversary should be able to forge a valid attestation for a node which is in an invalid state. We extend this idea for a VNF-FG. For an unforgeable collective attestation scheme no adversary should be able to provide a valid collective attestation that was not legitimately generated by an honest node. In that case we assume that every node in the VNF-FG is equipped with a RoT assumed to be incorruptible.

Oracles. We define the following oracles:

- $\underline{\mathsf{oCorrupt}(\mathsf{node}_i) \rightarrow \mathsf{OK}}$: This oracle allows the adversary to corrupt node node_i. Corrupt nodes allow the adversary software access to that node, but no access to the secret key or the state stored in the RoT.
- $\underline{\mathsf{oReqGen}(\mathcal{G}) \rightarrow \mathsf{req}}$: This oracle takes in input a graph description $\mathcal{G}$, it retrieves the spanning tree $\mathcal{T}_\mathcal{G}$ generated upon setting up the graph $\mathcal{G}$, then runs as a black-box the algorithm $\mathtt{cRA.ReqGen}(\mathsf{sk}_{\mathsf{ver}}, \mathcal{T}_\mathcal{G})$. The resulting request req is forwarded to the adversary.
- $\underline{\mathsf{oAttest}(\mathsf{node}_i, \mathsf{req}) \rightarrow \mathsf{a}}$: The adversary can use this oracle to obtain an attestation generated legitimately for node node_i, using the nonce $\mathsf{rand}_{\mathsf{req}}$ from the request req. This oracle runs, as a black box, the algorithm $\mathtt{cRA.Attest}(\mathsf{sk}_i, \mathsf{conf}_i, \mathsf{confset}_i, \mathsf{link}_i, \mathsf{rand}_{\mathsf{req}})$ in order to produce the attestation.
- $\underline{\mathsf{oAggreg}(\mathsf{node}_i, (\mathsf{a}_1, ..., \mathsf{a}_n), (\mathsf{agg}_1, ..., \mathsf{agg}_m), \mathsf{req}) \rightarrow \mathsf{agg} \cup \perp}$: The aggregation oracle can be queried to get an aggregation of attestations $(\mathsf{a}_1, ..., \mathsf{a}_n)$ and/or potential aggregates $(\mathsf{agg}_1, ..., \mathsf{agg}_m)$ under nonce $\mathsf{rand}_{\mathsf{req}}$ from request req. The aggregation is performed by node node_i, by running, as a black box, the algorithm $\mathtt{cRA.Aggregate}(\mathsf{sk}_i, \mathsf{stg}_i, (\mathsf{a}_1, ..., \mathsf{a}_n), (\mathsf{agg}_1, ..., \mathsf{agg}_m), \mathsf{rand})$. Note that, in our scheme, only certain nodes can verify, and thus aggregate attestations. If the input node is not one of those nodes, the result provided by this oracle is $\perp$.

Security game. We define the game $G_{\mathsf{UF}}(\lambda)$ below.

Definition 5.3.1 (Unforgeability)**.** *We say that* D-cRA *is* (n, ϵ)*-unforgeable if for any graph* $\mathcal{G} \in \Gamma$ *on at most* n *nodes chosen at setup, and for all PPT* $\mathcal{A}$ *playing the unforgeability game, it holds that:*

$$\mathsf{Adv}^{\mathsf{UF}}_{\mathtt{cRA}}(\mathcal{A}) = \Pr[\mathcal{A} \textit{ wins } G_{\mathsf{UF}}(\lambda)] \leq \epsilon$$

The value $\mathsf{Adv}^{\mathsf{UF}}_{\mathtt{cRA}}(\mathcal{A})$ *is called the* advantage *of the adversary in winning the unforgeability game.*

Game $G_{\mathsf{UF}}(\lambda)$

$(\mathsf{ppar}, \mathcal{G}, \mathcal{T}_{\mathcal{G}}) \leftarrow \mathtt{cRA.Setup}(1^{\lambda}, \mathcal{G})$

$(\mathsf{sk}_{\mathsf{ver}}, \mathsf{pk}_{\mathsf{ver}}) \leftarrow \mathtt{cRA.InitVerifier}(\mathsf{ppar})$

$\mathsf{stg}_{\mathsf{ver}} \leftarrow \mathtt{cRA.InitStg}(\mathsf{ppar}, \mathsf{ver}, (\mathsf{pk}_{\mathsf{root}}, \mathsf{confset}_{\mathsf{root}}, \mathsf{link}_{\mathsf{root}}), \varnothing)$

$\forall i \in [n] : (\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{confset}_i, \mathsf{link}_i) \leftarrow \mathtt{cRA.InitNode}(\mathsf{ppar})$

$\mathsf{stg}_i \leftarrow \mathtt{cRA.InitStg}(\mathsf{ppar}, \mathsf{node}_i, (\mathsf{pk}_{\mathsf{neib}_i}, \mathsf{confset}_{\mathsf{neib}_i}, \mathsf{link}_{\mathsf{neib}_i})_{\mathsf{nb}_{\mathsf{neib}}}, \mathsf{pk}_{\mathsf{ver}})$

$(\mathsf{agg}, \mathsf{a}, \mathsf{node}_i, \mathsf{req}) \leftarrow \mathcal{A}^{\mathsf{oCorrupt}(.),\mathsf{oReqGen}(.),\mathsf{oAttest}(.),\mathsf{oAggreg}(.)}(1^{\lambda}, \{\mathsf{pk}_i, \mathsf{confset}_i, \mathsf{link}_i\}_{i\in[n]})$

$\mathcal{A}$ **wins** iff.

- $\mathtt{cRA.AggVerify}(\mathsf{stg}_i, \mathsf{agg}, \mathsf{req}) = 1$ or $\mathtt{cRA.Verify}(\mathsf{stg}_i, \mathsf{a}, \mathsf{req}) = 1$.
- AND the oAttest and oAggreg oracles have not been queried using the pair $(\mathsf{node}_i, \mathsf{rand}_{\mathsf{req}})$ (with $(\mathsf{rand}_{\mathsf{req}}, \sigma) \leftarrow \mathsf{req}$).

Figure 5.3: The cRA unforgeability game

Linking

In Chapter 3 we define linking as a game, in which an adversary must succeed in convincing a verifier that a VM is managed by a given hypervisor, even though this is not the case. Here we are aiming for a similar property, adapted to the context of collective attestation.

In tree-based collective attestation, components such as hypervisors and VMs can be distant nodes within the tree. We could enforce local verification of linking, enabling each node to verify the validity of linking of other components. Yet, this would expose linking information beyond the span of components that are directly related, and require the transmission of verification elements from each node to all other nodes. Instead, we prefer to make an implicit local partial verification of linking information through the attestation verification algorithm and provide an explicit result only at the end of the attestation process for all the VNF-FG.

The adversary's goal is to persuade the verifier at the root of the spanning tree in the VNF of the linking of one specific VNF and hypervisor, which are not in reality associated with each other. Formally, like is the unforgeability property, we let the challenger set up the VNF-FG; thus, every node is initialized with the expected linking information, outside of the adversary's control. We then give the adversary access to an oracle which allows it to modify the configuration of the VNF-FG. In addition the adversary is given active access to the oAttest, oReqGen, and oAggreg oracles defined in Section 5.3.2.

The adversary wins if it is able to provide an attestation and an aggregate which verify as linkable by the cRA.Link algorithm, in spite of the modification of at least one linking information (translated into a change in the structure of the original VNF-FG).

In order to distinguish calls to the oAttest and oAggreg oracles for one graph topology or

another, the challenger keeps track of a database of legitimately generated requests, obtained by calls to oReqGen. This database, which we denote as $\mathcal{D}_{\mathsf{oReqGen}}$ consists of entries of the type $(\mathcal{G}, \mathsf{req})$, which essentially list the graph $\mathcal{G}$ given in input to oReqGen and the resulting request req, which includes a nonce $\mathsf{rand}_{\mathsf{req}}$.

Oracles. In addition to the oAttest, oReqGen and oAggreg oracles we introduce the oModG oracle :

- $\underline{\mathsf{oModG}(\mathcal{G}, (\mathsf{sk}, \mathsf{pk}, \mathsf{stg}, \mathsf{confset}, \mathsf{link})_n)}$: This oracle takes as input a graph description $\mathcal{G} \in \Gamma$. It can be used to modify the VNF-FG topology according to this graph description. Such modifications include moving a VNF to a different location within the graph, removing a VNF, but also adding a VNF. The adversary has full control over added VNFs and specifies their information as it wants as input to this oracle. The oracle runs the cRA.InitStg for the new VNFs and only for them, all other VNF state remain unchanged.

Security game. We define the $G_{\mathsf{LK}}(\lambda)$ as follows :

Game $G_{\mathsf{LK}}(\lambda)$

$(\mathsf{ppar}, \mathcal{G}, \mathcal{T}_{\mathcal{G}}) \leftarrow \mathtt{cRA.Setup}(1^\lambda, \mathcal{G})$
$(\mathsf{sk}_{\mathsf{ver}}, \mathsf{pk}_{\mathsf{ver}}) \leftarrow \mathtt{cRA.InitVerifier}(\mathsf{ppar})$
$\mathsf{stg}_{\mathsf{ver}} \leftarrow \mathtt{cRA.InitStg}(\mathsf{ppar}, \mathsf{ver}, (\mathsf{pk}_{\mathsf{root}}, \mathsf{confset}_{\mathsf{root}}, \mathsf{link}_{\mathsf{root}}), \varnothing)$
$\forall i \in [n] : (\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{confset}_i, \mathsf{link}_i) \leftarrow \mathtt{cRA.InitNode}(\mathsf{ppar})$
$\mathsf{stg}_i \leftarrow \mathtt{cRA.InitStg}(\mathsf{ppar}, \mathsf{node}_i, (\mathsf{pk}_{\mathsf{neib}_i}, \mathsf{confset}_{\mathsf{neib}_i}, \mathsf{link}_{\mathsf{neib}_i})_{\mathsf{nb}_{\mathsf{neib}}}, \mathsf{pk}_{\mathsf{ver}})$
$(\mathsf{agg}_{\mathsf{root}}, \mathsf{a}_{\mathsf{root}}, \mathsf{req}) \leftarrow \mathcal{A}^{\mathsf{oModG}(.),\mathsf{oReqGen}(.),\mathsf{oAttest}(.),\mathsf{oAggreg}(.)}(1^\lambda, \{\mathsf{pk}_i, \mathsf{confset}_i, \mathsf{link}_i\}_{i\in[n]})$

$\mathcal{A}$ **wins** iff.
- $\mathtt{cRA.Link}(\mathsf{stg}_{\mathsf{ver}}, \mathsf{a}_{\mathsf{root}}, \mathsf{agg}_{\mathsf{root}}, \mathsf{req}) = 1$.
- AND there exists no nonce $\mathsf{rand}_{\mathsf{req}}$ and no request req such that: $\mathsf{rand}_{\mathsf{req}} \in \mathsf{req}$
- AND $(\mathcal{G}, \mathsf{req}) \in \mathcal{D}_{\mathsf{oReqGen}}$ with $\mathcal{G}$ being the graph input at the start of the game,
- AND $\mathsf{a}_{\mathsf{root}}$ and $\mathsf{agg}_{\mathsf{root}}$ issued by oAttest and oAggreg on input $\mathsf{rand}_{\mathsf{req}}$.

Figure 5.4: The linking game

We define $\mathsf{Adv}^{\mathsf{LK}}_{\mathtt{cRA}}(\mathcal{A})$ to be the probability of adversary $\mathcal{A}$ to win the game the $G_{\mathsf{LK}}(\lambda)$.

Definition 5.3.2 (Linkability)**.** *We say that* D-cRA *is* (n, ϵ)*-linkable if for all original graphs* $\mathcal{G} \in \Gamma$ *employed at setup, and any PPT* $\mathcal{A}$ *playing the linking game for a modified graph* $\mathcal{G}$ *on at most* n *nodes, it holds that:*

$$\mathsf{Adv}^{\mathsf{LK}}_{\mathtt{cRA}}(\mathcal{A}) = \Pr[\mathcal{A} \textit{ wins } G_{\mathsf{LK}}(\lambda)] \leq \epsilon$$

The value $\mathsf{Adv}^{\mathsf{LK}}_{\mathtt{cRA}}(\mathcal{A})$ *is called the* advantage *of the adversary in winning the linking game.*

Configuration Hiding

The core idea behind this property is to guarantee that even if all the nodes (VNFs and hypervisors) of a VNF-FG are compromised except for one honest node, that honest node's configuration is indistinguishable to an adversary from all the other configurations in that node's (publicly known) configuration set. We give $\mathcal{A}$ access to the oracle $\mathsf{oChooseConf}_b$, presented below.

Oracle. Let $\mathsf{oChooseConf}_b$ be the following oracle :

- $\underline{\mathsf{oChooseConf}_b(\mathsf{node}_i, \mathsf{conf}_0, \mathsf{conf}_1, \mathsf{confset}) \rightarrow (\mathsf{pk}_i, \mathsf{link}_i) \cup \perp}$: This oracle can only be called once. On input two configurations conf_0 and conf_1 as well as a configuration set $\mathsf{confset}$ and a bit b, this oracle checks that both configurations are in the set of valid configuration $\mathsf{confset}$ ($\mathsf{conf}_0 \in \mathsf{confset}$ and $\mathsf{conf}_1 \in \mathsf{confset}$). If the verification fails, the oracle outputs $\perp$. Otherwise it initializes the configuration of the component node_i as conf_b, sets the configuration set of that node to $\mathsf{confset}$ and then simulates all the other steps of the $\mathtt{cRA.InitNode}$ algorithm to obtain a public and private key for that node, as well as any linking information. The adversary is given the public key pk and the linking information link.

Security game. Let i^* denote the index of the node initialized by querying $\mathsf{oChooseConf}_b$. We define the $G_{\mathsf{CHide}}(\lambda)$ game :

$$
\begin{array}{l}
\underline{\text{Game } G_{\mathsf{CHide}}(\lambda)} \\
\mathcal{G} \leftarrow \mathcal{A} \\
(\mathsf{ppar}, \mathcal{G}, \mathcal{T}_{\mathcal{G}}) \leftarrow \mathtt{cRA.Setup}(1^\lambda, \mathcal{G}) \\
b \xleftarrow{\$} \{0,1\} \\
\mathsf{pk}_{\mathsf{ver}} \cup \{\mathsf{node}_i, \mathsf{pk}_i, \mathsf{confset}_i, \mathsf{link}_i\}_{i\in[n], i\neq i^*} \leftarrow \mathcal{A}^{\mathsf{oChooseConf}_b(.)}(1^\lambda, \mathcal{G}, \mathsf{ppar}, \mathcal{T}_{\mathcal{G}}) \\
\underline{d \leftarrow \mathcal{A}^{\mathsf{oAttest}(.),\mathsf{oAggreg}(.)}(1^\lambda, \mathsf{ppar}, \mathcal{G}, \mathsf{pk}_{\mathsf{ver}} \cup \{\mathsf{node}_i, \mathsf{pk}_i, \mathsf{confset}_i, \mathsf{link}_i\}_{i\in[n], i\neq i^*})} \\
\mathcal{A} \textbf{ wins iff. } b = d
\end{array}
$$

Figure 5.5: The configuration-hiding game

Definition 5.3.3 (Configuration Hiding)**.** *We say that* D-cRA *is ϵ-configuration-hiding if for any PPT $\mathcal{A}$ playing the configuration-hiding game, it holds that:*

$$\mathsf{Adv}^{\mathsf{CHide}}_{\mathtt{cRA}}(\mathcal{A}) := |\Pr[\mathcal{A} \textit{ wins } G_{\mathsf{CHide}}(\lambda)] - \frac{1}{2}| \leq \epsilon$$

The value $\mathsf{Adv}^{\mathsf{CHide}}_{\mathtt{cRA}}(\mathcal{A})$ *is called the* advantage *of the adversary in winning the configuration-hiding game.*

5.4 Construction

In this section we instantiate our scheme D-cRA using a signature scheme $\mathtt{SIG} = (\mathtt{SIG.Setup}, \mathtt{SIG.KeyGen}, \mathtt{SIG.Sign}, \mathtt{SIG.Verify})$, a commitment scheme $\mathtt{COM}$ = ($\mathtt{COM.Commit}$, $\mathtt{COM.Verify}$) and a non-interactive zero-knowledge proof of knowledge NIZK = ($\mathtt{NIZK.Setup}$, $\mathtt{NIZK.Pr}$, $\mathtt{NIZK.Ver}$). As in Section 5.3.1, we structure the presentation of our instantiation in two phases: the online and the offline phase.

Offline Phase The offline phase corresponds to the initialization of the VNF-FG with all the parameters needed for the attestation process itself. The verifier and each node (VNFs as well as the hypervisors) are initialized with a signature key-pair, their expected sets of configurations and linking information. Then they share such public information with their neighbours. This is done by running the $\mathtt{cRA.InitNode}$ and $\mathtt{cRA.InitStg}$ algorithms for each node (the description of each algorithm is provided below in detail). After this setup, every node (VNFs and hypervisors) has its storage initialized with all the information they need to be able to verify their neighbours attestation. We start by describing the content of this storage and give more information about what linking information is:

$\underline{\mathsf{stg} = (\mathsf{pk}_{\mathsf{ver}}, (\mathsf{pk}_{\mathsf{neib}}, \mathsf{confset}_{\mathsf{neib}}, \mathsf{link}_{\mathsf{neib}})_{\mathsf{nb}_{\mathsf{neib}}})}$: The storage of every node in the VNF-FG includes: the verifier public key $\mathsf{pk}_{\mathsf{ver}}$ (used when attestation requests are forwarded, in order to ensure that nodes only respond to requests legitimately produced by the verifier), and a table of triples $(\mathsf{pk}_{\mathsf{neib}}, \mathsf{confset}_{\mathsf{neib}}, \mathsf{link}_{\mathsf{neib}})$ (each entry corresponding to a neighbour and featuring the latter's public key $\mathsf{pk}_{\mathsf{neib}}$, its set of possible configurations $\mathsf{confset}_{\mathsf{neib}}$, and expected linking information $\mathsf{link}_{\mathsf{neib}}$). In addition to such values, the verifier also contains an entry detailing information related to the root node (*i.e.*, the entry node of the VNF-FG).

$\underline{\mathsf{link}:}$ The linking information of each node differs depending on whether that node is a VNF or a hypervisor. In the case of a VNF, the linking information is the public attestation (signature) key of that VNF (the same as the key generated by the $\mathtt{cRA.InitNode}$ algorithm). For a hypervisor, the linking information is the set of public attestation keys of all the VNFs the hypervisor is currently managing. For a compact storage of linking information, the (collision-resistant) hash of the public keys could be used. This technique is especially useful in the case of the hypervisor, in order to ensure that the size of the linking information remains constant in the number of managed VNFs.

Remark 5.4.1. *In our security analysis, we will assume this setup to be done in a trusted way, both in terms of generating the setup values and communicating them to the neighbouring nodes. The latter of these assumptions would count as strong in the case of forwarding graphs consisting of IoT devices, since the latter are highly mobile, and therefore new forwarding graphs will form in ad-hoc manners at short intervals. By contrast, in our use case (Virtual Network Functions owned by various infrastructure providers), VNF-FGs are inherently stable and less likely to change over time. As a result, an assumption of a trusted setup is less strong and can be achieved in practice by the use of mutually-authenticated secure channels.*

Remark 5.4.2. *We furthermore stress that the storage* stg *is stored into the RoT and cannot be tampered with in practice.*

We proceed to detail the offline-phase algorithms:

$\mathtt{cRA.Setup}(1^\lambda, \mathcal{G}) \rightarrow (\mathsf{ppar}, \mathcal{G}, \mathcal{T}_\mathcal{G})$: The setup algorithm runs the individual setup algorithms of the signature $\mathsf{ppar}_{\mathtt{SIG}} \leftarrow \mathtt{SIG.Setup}(1^\lambda)$ and the commitment schemes $\mathsf{ppar}_{\mathtt{COM}} \leftarrow \mathtt{COM.Setup}(1^\lambda)$. Given the graph description $\mathcal{G}$ of the VNF-FG it computes a spanning tree $\mathcal{T}_\mathcal{G}$[3]. Finally the algorithm sets $\mathsf{ppar} \leftarrow (1^\lambda, \mathsf{ppar}_{\mathtt{SIG}}, \mathsf{ppar}_{\mathtt{COM}})$ and outputs $(\mathsf{ppar}, \mathcal{G}, \mathcal{T}_\mathcal{G})$. All subsequent algorithms take ppar in input, even if the latter is not specifically included in the algorithm parameters.

$\mathtt{cRA.InitVerifier}(\mathsf{ppar}) \rightarrow (\mathsf{sk}_{\mathsf{ver}}, \mathsf{pk}_{\mathsf{ver}})$: This algorithm initializes the verifier by creating a signature keypair : $(\mathsf{sk}_{\mathsf{ver}}, \mathsf{pk}_{\mathsf{ver}}) \leftarrow \mathtt{SIG.KeyGen}(\mathsf{ppar}_{\mathtt{SIG}})$.

$\mathtt{cRA.InitNode}(\mathsf{ppar}) \rightarrow (\mathsf{pk}, \mathsf{sk}, \mathsf{confset}, \mathsf{link})$: This algorithm run for every node in the VNF-FG (both VNFs and the hypervisors). It then sets the linking information according to the description we gave above, as well as the set of possible valid configurations $\mathsf{confset}$. Then it generates a signature key-pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathtt{SIG.KeyGen}(\mathsf{ppar}_{\mathtt{SIG}})$ (these keys will be stored within the RoT of the device) and run the setup algorithm of the NIZK proof of knowledge, $\mathsf{ppar}_{\mathtt{NIZK}} \leftarrow \mathtt{NIZK.Setup}(1^\lambda)$.

$\mathtt{cRA.InitStg}(\mathsf{ppar}, (\mathsf{node} \cup \mathsf{ver}), (\mathsf{pk}_{\mathsf{neib}}, \mathsf{confset}_{\mathsf{neib}}, \mathsf{link}_{\mathsf{neib}})_{\mathsf{neib} \in \mathcal{N}[\mathsf{node}, \mathcal{G}]}, \mathsf{pk}_{\mathsf{ver}}) \rightarrow \mathsf{stg}$: Once every node (including the verifier) is initialized, important public and linking information has to be propagated to pertinent other nodes. This information makes up the storage of the node. Every node receives triplets $(\mathsf{pk}_{\mathsf{neib}}, \mathsf{confset}_{\mathsf{neib}}, \mathsf{link}_{\mathsf{neib}})$ from each of its neighbours, and stores them into a storage stg as a table. Also included in the storage is the public key of the verifier. In its own turn, the verifier initializes its storage with the information of the root node (namely $\mathsf{pk}_{\mathsf{root}}$, $\mathsf{confset}_{\mathsf{root}}$ and $\mathsf{link}_{\mathsf{root}}$).

[3] We reiterate that the spanning tree is computed, in practice, in a distributed and trusted way outside of the control of the adversary.

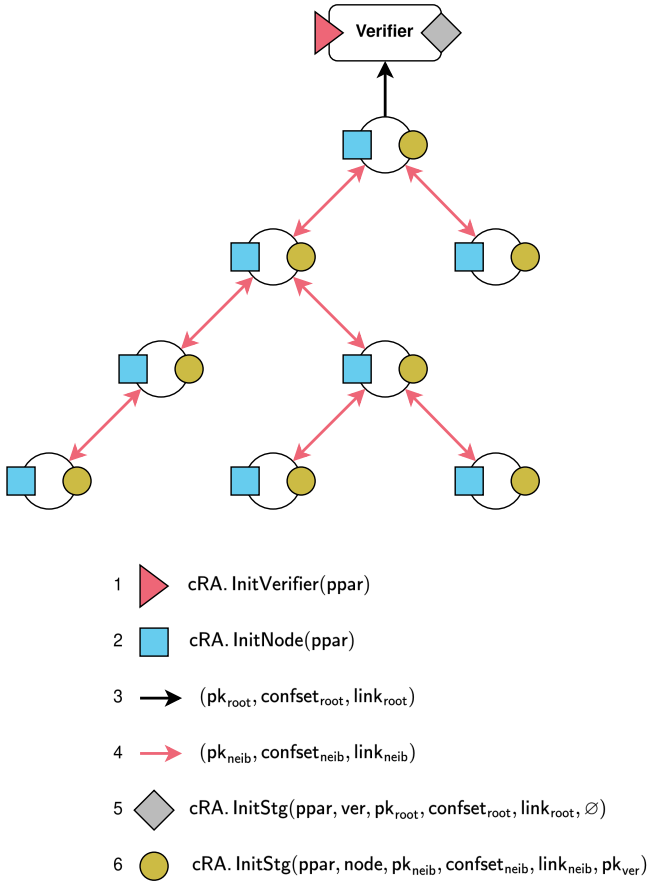


Figure 5.6: The offline phase after setup.

Online Phase The setup phase of the scheme takes place offline and – in our use case – will be relatively infrequent. Once the VNF-FG and each of its composing nodes are set up, one can proceed to performing attestation, in an online phase.
Each attestation is produced in response to an attestation request, that was generated by the verifier (*i.e.*, the operator) through the use of the $\texttt{cRA.ReqGen}$ algorithm. The request contains an attestation nonce $\texttt{rand}_{\text{req}}$, and is authenticated. Subsequently, nodes can verify the authentication.
Once the verifier generates the attestation request, the latter is propagated through the computed spanning tree $\mathcal{T}_{\mathcal{G}}$. When the request reaches the leaf nodes of the spanning tree, the latter retrieve their current configuration $\texttt{conf}$, run the $\texttt{cRA.Attest}$ algorithm, and send that result to the parent-nodes. Upon receiving authenticated attestations from their children, parents verify them by using the $\texttt{cRA.Verify}$ algorithm, then aggregate those values by using $\texttt{cRA.Aggregate}$, sending the aggregate further up the tree. Note that even a valid attestation or aggregate will be deemed invalid if it is computed for the wrong

nonce (*i.e.*, not the nonce recovered from the attestation request by the verifying node). Formally, we define attestations and aggregates as follows.

$\underline{\mathsf{a} = (\pi_\mathsf{a}, \sigma_\mathsf{a}, \mathsf{c})}$: An attestation consists of: (1) A commitment c of the true configuration of the attested node[4]; (2) a proof π_a that the commitment is made with respect to a configuration value included in a set of valid configurations $\mathsf{confset}$ (the attestation itself does not contain the value of $\mathsf{confset}$, which is in fact contained in the storage value included in the Root of Trust of neighbouring nodes); (3) a signature σ_a over the commitment value c, the linking information link, and a nonce $\mathsf{rand}_\mathsf{req}$ – retrieved from the attestation request. The random character of $\mathsf{rand}_\mathsf{req}$ is crucial in preventing replays of old attestations.

$\underline{\mathsf{agg} = (\mathsf{res}_\mathsf{agg}, \sigma_\mathsf{agg})}$: An aggregate contains: (1) A binary value $\mathsf{res}_\mathsf{agg}$ which indicates if all the prior attestations/aggregates verify as valid (as soon as one aggregate or attestation verification fails, the binary result returned by the verifier node will be 0); (2) A signature σ_agg over the result $\mathsf{res}_\mathsf{agg}$ with the nonce $\mathsf{rand}_\mathsf{req}$ retrieved by that node from the propagated attestation request.

The successive verification, attestation, aggregation, and forwarding of the attestations is done in increments, with parents that receive aggregate values additionally verifying the aggregates by using the `cRA.AggVerify` algorithm.
Apart from the attestation and/or aggregate verification, the verifier will moreover check the correct linking between the nodes of the graph. This is done through the `cRA.Link` algorithm, which states whether the entire VNF-FG is in the expected state with VNF running on the expected hypervisor or if at least one node is in an invalid state/has invalid linking. An overview of how the online-phase algorithms are used is provided in Figure 5.7.

Remark 5.4.3. *Note that some online steps are parallelizable, while others are not. For instance, nodes can generate attestations as soon as they receive the verifier's authenticated attestation request (from which they can extract and use the attestation nonce). However, parent nodes must first wait for the child nodes to produce their attestations (and potential aggregates) before aggregating their own attestations to the chain and sending them on. This dependency between the nodes implies the practical requirement of a timeout*[5]*. If a node does not send its attestation within the time limit, the node responsible for verifying that node's attestation/aggregate must assume that*

[4]Note that the commitment is hiding and thus no information can be learned about the configuration

[5]Timeouts can be exploited by adversaries in several ways, including in order to deny service to honest entities. While in our current work DoS attacks are out of scope, understanding such limitations for VNF-FG schemes is viewed as essential future work.

the node is in an invalid state.

Remark 5.4.4. *We note that the attestation-computation, aggregation, and verification steps of all of the algorithms deployed during the online phase by a node of the VNF-FG are executed from the Root of Trust (RoT). The only operation that is not assumed to be trusted is the transmission of that information, allowing an adversary to potentially delay, jam, or modify communication.*

We now describe the $\texttt{cRA.ReqGen}$, $\texttt{cRA.ReqVer}$, $\texttt{cRA.Link}$, $\texttt{cRA.Attest}$, $\texttt{cRA.Verify}$, $\texttt{cRA.Aggregate}$ and $\texttt{cRA.AggVerify}$ algorithms :

$\underline{\texttt{cRA.ReqGen}(\mathsf{sk}_{\mathsf{ver}}, \mathcal{T}_{\mathcal{G}}) \rightarrow \mathsf{req}}$: On input the verifier secret-key $\mathsf{sk}_{\mathsf{ver}}$ this algorithm chooses a random number $\mathsf{rand}_{\mathsf{req}} \xleftarrow{\$} \{0,1\}^{\lambda}$. It computes a signature over that random value $\sigma_{\mathsf{req}} \leftarrow \texttt{SIG.Sign}(\mathsf{sk}_{\mathsf{ver}}, \mathsf{rand}_{\mathsf{req}})$, then sends an attestation request req consisting of the concatenation $(\mathsf{rand}_{\mathsf{req}}, \sigma_{\mathsf{req}})$ to the root node (entry point of the VNF-FG).

$\underline{\texttt{cRA.Attest}(\mathsf{stg}, \mathsf{sk}, \mathsf{conf}, \mathsf{confset}, \mathsf{link}, \mathsf{req}) \rightarrow \mathsf{a} \cup \perp}$: attestation takes in input a node's storage value stg, the node's private key sk, the node's current configuration conf, configuration set $\mathsf{confset}$, and linking information link, as well as an attestation request req. Parsing req as $(\mathsf{rand}_{\mathsf{req}}, \sigma_{\mathsf{req}})$, the node uses the verifier's public key value $\mathsf{pk}_{\mathsf{ver}}$ (included in each node's storage) to verify the signature $\texttt{SIG.Verify}(\mathsf{pk}_{\mathsf{ver}}, \mathsf{rand}_{\mathsf{req}}, \sigma_{\mathsf{req}})$. If the signature is invalid, the algorithm outputs $\perp$ otherwise the node can produce an attestation. First, the node computes a commitment of its current configuration $\mathsf{c} \leftarrow \texttt{COM.Commit}(\mathsf{conf}, r)$ with some randomness r. Then it computes the proof $\pi_{\mathsf{a}} \leftarrow \texttt{NIZK.Pr}(\{(\mathsf{conf}, r) : \mathsf{c} \leftarrow \texttt{COM.Commit}(\mathsf{conf}, r) \wedge \mathsf{conf} \in \mathsf{confset}\})$ which shows that c is a valid commitment for a value conf contained in $\mathsf{confset}$ without revealing the committed value. The node also signs c $\sigma_{\mathsf{a}} \leftarrow \texttt{SIG.Sign}(\mathsf{sk}, \mathsf{c}|\mathsf{link}|\mathsf{rand}_{\mathsf{req}})$, by using the node's private key sk, stored in the RoT. The signature ensures that the committed value was generated within the Root of Trust with a fresh commitment-nonce. Note that the signature is computed over the node's own linking information (which is the VNF's public key for a VNF, and a set of public keys of each of its linked VNFs for a hypervisor). Finally it sets $\mathsf{a} = (\pi_{\mathsf{a}}, \sigma_{\mathsf{a}}, \mathsf{c})$ and outputs a.

$\underline{\texttt{cRA.Verify}(\mathsf{stg}, \mathsf{a}, \mathsf{req}) \rightarrow \{0,1\} \cup \perp}$: given in input a securely-stored node storage stg, an attestation a, and the current attestation request req, the verification algorithm proceeds to verify that both the proof and the signature included in the attestation are valid. To do so, the node parses the attestation a as $(\pi_{\mathsf{a}}, \sigma_{\mathsf{a}}, \mathsf{c})$ and the request req as $(\mathsf{rand}_{\mathsf{req}}, \sigma_{\mathsf{req}})$. There are two cases: either the attestation stems from a node that is an actual neighbour of the verifying node, or not. In the second case, the

node will not have the public key $\mathsf{pk}_{\mathsf{neib}}$ allowing it to verify the attestation – and will eventually return $\perp$. In the former case, the verifying node retrieves from its storage stg the public key $\mathsf{pk}_{\mathsf{neib}}$ of the attested node, the expected configuration $\mathsf{confset}_{\mathsf{neib}}$, and the expected linking-information $\mathsf{link}_{\mathsf{neib}}$, and checks that $\mathtt{NIZK.Ver}(\pi_{\mathsf{a}}) = 1$ (*i.e.*, the node checks that the commitment c contains a configuration which is in the set $\mathsf{confset}_{\mathsf{neib}}$). It also verifies that $\mathtt{SIG.Verify}(\mathsf{pk}_{\mathsf{neib}}, \mathsf{c}\,|\mathsf{link}_{\mathsf{neib}}|\mathsf{rand}_{\mathsf{req}}, \sigma_{\mathsf{a}}) = 1$. If every check passes, then the algorithm outputs 1, otherwise it outputs 0.

$\mathtt{cRA.Aggregate}(\mathsf{stg}, \mathsf{sk}, (\mathsf{a}_1, ..., \mathsf{a}_i), (\mathsf{agg}_1, ..., \mathsf{agg}_j), \mathsf{req}) \rightarrow \mathsf{agg}'$: The aggregation algorithm takes in input a variable number of attestations a and/or a variable number of aggregate values agg under an attestation request req, as well as the node's storage stg and private key sk. The algorithm parses req as $(\mathsf{rand}_{\mathsf{req}}, \sigma_{\mathsf{req}})$, then verifies every attestation $\mathtt{cRA.Verify}(\mathsf{stg}, \mathsf{a}, \mathsf{req})$ and every aggregate $\mathtt{cRA.AggVerify}(\mathsf{stg}, \mathsf{agg}, \mathsf{req})$. If all the verifications return 1 (valid), the algorithm sets $\mathsf{res}'_{\mathsf{agg}} = 1$, otherwise it sets $\mathsf{res}'_{\mathsf{agg}} = 0$. Finally the algorithm computes a signature over the result $\sigma'_{\mathsf{agg}} = \mathtt{SIG.Sign}(\mathsf{sk}, \mathsf{res}'_{\mathsf{agg}}|\mathsf{rand}_{\mathsf{req}})$ and outputs $\mathsf{agg}' = (\mathsf{res}'_{\mathsf{agg}}, \sigma'_{\mathsf{agg}})$.

$\mathtt{cRA.AggVerify}(\mathsf{stg}, \mathsf{agg}, \mathsf{req}) \rightarrow \{0, 1\} \cup \perp$: This algorithm parses the aggregate attestation agg as $(\mathsf{res}_{\mathsf{agg}}, \sigma_{\mathsf{agg}})$ and the request req as $(\mathsf{rand}_{\mathsf{req}}, \sigma_{\mathsf{req}})$. There are, once more, two cases: either the aggregate is received from a neighbouring node, or it is not. In the latter case, the verifying node will not have the information required to verify the aggregate, and it will output $\perp$. Else, if the aggregate is received from a neighbour, the verifying node retrieves the public key $\mathsf{pk}_{\mathsf{neib}}$ associated with the aggregate from its storage and verifies the signature over the result $\mathtt{SIG.Verify}(\mathsf{pk}_{\mathsf{neib}}, \mathsf{res}_{\mathsf{agg}}|\mathsf{rand}_{\mathsf{req}}, \sigma_{\mathsf{agg}})$. If the signature is valid it outputs 1 otherwise it outputs 0.

$\mathtt{cRA.Link}(\mathsf{stg}_{\mathsf{ver}}, \mathsf{a}_{\mathsf{root}}, \mathsf{agg}_{\mathsf{root}}, \mathsf{req}) \rightarrow \{0, 1\}$: The linking algorithm is run by the verifier using its storage $\mathsf{stg}_{\mathsf{ver}}$ given the attestation $\mathsf{a}_{\mathsf{root}}$ and aggregate $\mathsf{agg}_{\mathsf{root}}$ of the root node. It checks both under the root node public key : $\mathtt{cRA.Verify}(\mathsf{stg}_{\mathsf{ver}}, \mathsf{a}_{\mathsf{root}}, \mathsf{req})$ and $\mathtt{cRA.AggVerify}(\mathsf{stg}_{\mathsf{ver}}, \mathsf{agg}_{\mathsf{root}}, \mathsf{req})$. If both checks succeed, it outputs 1 (and concludes that the attestation confirms the expected linking of VNFs and hypervisors) otherwise 0.

Remark 5.4.5. *In our construction we assume nodes know the location of their neighbours and thus know which public key should be used to verify an attestation or an aggregate. If the verification requires for a node to iterate through all the public keys of all of its neighbours to check if any correctly verify, this could be used by an adversary for a DoS attack. In practice, one can require nodes to communicate over a mutually-authenticated (secure) channel.*

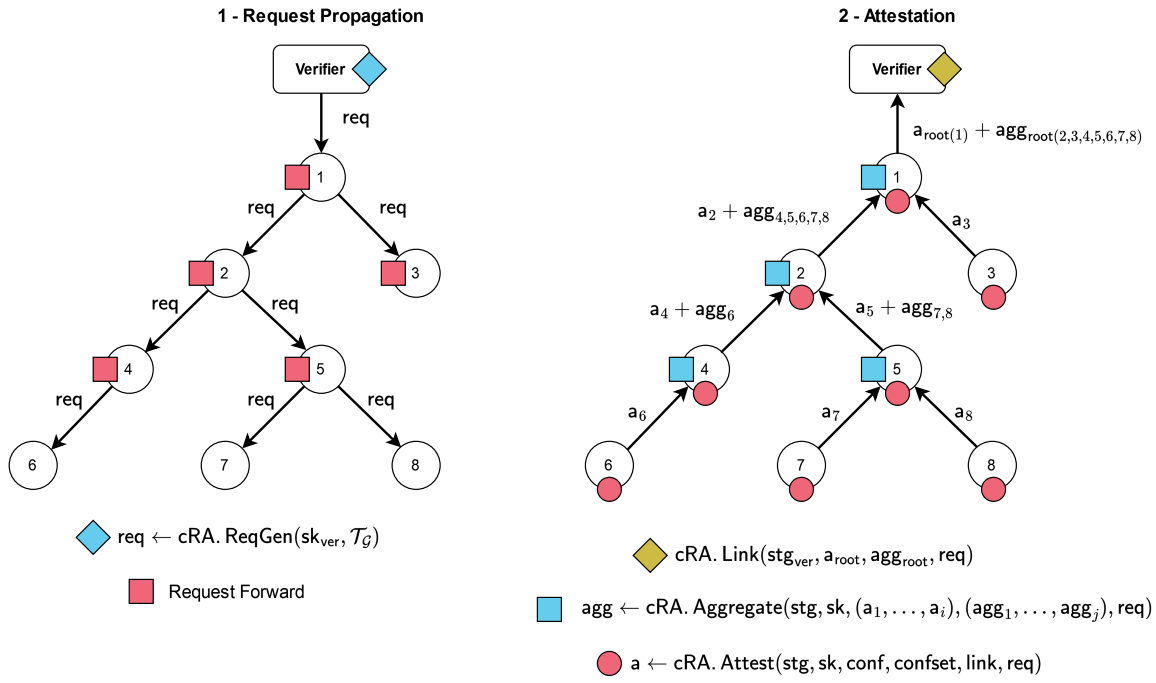


Figure 5.7: The online phase.

5.5 Security Analysis

In this section we prove that the construction presented in Section 5.4 guarantees the properties defined in Section 5.3.

5.5.1 Configuration Hiding

Theorem 5.5.1. *Let* cRA *be the privacy-preserving collective remote attestation scheme described in Section 5.4, using a signature scheme* SIG = (SIG.Setup, SIG.KeyGen, SIG.Sign, SIG.Verify)*, a commitment scheme* COM(COM.Commit, COM.Verify) *and a non-inter-active zero-knowledge proof of knowledge* NIZK = (NIZK.Setup, NIZK.Pr, NIZK.Ver)*. If the commitment scheme is computationally hiding and the NIZK proof is composable zero-knowledge in the sense of [56], then* D-cRA *is configuration-hiding. More specifically, assume that there exists an adversary* $\mathcal{A}$ *winning the configuration-hiding game* ($G_{\mathsf{CHide}}(\lambda)$) *against a graph on* n *nodes with advantage* $\mathsf{Adv}^{\mathsf{CHide}}_{\mathtt{cRA}}(\mathcal{A})$*. Then there exist adversaries (reductions)* $\mathcal{R}_1$ *against the computational-hiding property of* COM *and* $\mathcal{R}_2$ *against the zero-knowledge property of* NIZK *winning with advantages* $\mathsf{Adv}^{\mathsf{Hide}}_{\mathtt{COM}}(\mathcal{R}_1)$ *and*

respectively $\mathsf{Adv}^{\mathsf{CompZK}}_{\mathsf{NIZK}}(\mathcal{R}_2)$*, such that:*

$$\mathsf{Adv}^{\mathsf{CHide}}_{\mathtt{cRA}}(\mathcal{A}) \leq n \cdot (\mathsf{Adv}^{\mathsf{Hide}}_{\mathtt{COM}}(\mathcal{R}_1) + \mathsf{Adv}^{\mathsf{CompZK}}_{\mathsf{NIZK}}(\mathcal{R}_2)).$$

Proof. Recall that in the configuration-hiding game, the adversary's goal is to learn which of two chosen configurations is used to initialize a particular, target node. The adversary in this case controls both the topology of the VNF-FG and most of the nodes (with the exception of the target node), from which it can request attestations and aggregates.

The proof proceeds with only 3 intermediate game hops.

$\mathbb{G}_0$**:** The original configuration-hiding game.

$\mathbb{G}_1$**:** In this intermediate game, the challenger chooses randomly an integer $i^* \in \{1, \ldots, n\}$ and outputs it privately at the beginning of the game. If the adversary queries node_i to the oChooseConf oracle, with $i \neq i^*$, then the challenger outputs $\perp$ (the game fails). Else, the game is run normally. Clearly, since the challenger has a probability of $\frac{1}{n}$ to guess the adversary's target node, it holds that:

$$\mathsf{Adv}^{\mathsf{CHide}}_{\mathbb{G}_0}(\mathcal{A}) \leq n \cdot \mathsf{Adv}^{\mathsf{CHide}}_{\mathbb{G}_1}(\mathcal{A}) \ .$$

$\mathbb{G}_2$**:** We modify $\mathbb{G}_1$ In $\mathbb{G}_2$, the challenger changes the way it answers oAttest queries to the target node node_{i^*}. Specifically, instead of generating the challenger consistently replaces the configuration value conf_b with that of conf_0 regardless of the bit b.This game is indistinguishable from the previous game from the point of view of the attacker, since we assumed the commitment scheme to be computationally hiding. Indeed, using a distinguisher between $\mathbb{G}_1$ and $\mathbb{G}_2$ we can construct a reduction $\mathcal{R}_1$ which aims to break the hiding property of the commitment scheme. Clearly, the only difference between the two schemes appears for $b = 1$, when the adversary is provided a commitment over conf_0 rather than a commitment over conf_1. If the distinguisher between $\mathbb{G}_1$ and $\mathbb{G}_2$ guesses which game it is playing, then the reduction is also able to distinguish whether the commitment is made over conf_0 or conf_1. Hence,

$$|\mathsf{Adv}^{\mathsf{CHide}}_{\mathbb{G}_1}(\mathcal{A}) - \mathsf{Adv}^{\mathsf{CHide}}_{\mathbb{G}_2}(\mathcal{A})| \leq \mathsf{Adv}^{\mathsf{Hide}}_{\mathtt{COM}}(\mathcal{R}_1).$$

$\mathbb{G}_3$**:** In this game hop, the challenger modifies once more its response to oAttest queries. It employs the simulator of the NIZK proof of knowledge in order to produce the proofs required in the attestation a produced by the challenge node node_{i^*}. This

is indeed possible for composable zero-knowledge, since the simulator requires in input only the statement and the public parameters required for verification – which our scheme provides. If an adversary can distinguish between $\mathbb{G}_2$ and $\mathbb{G}_3$, then it can distinguish between a real proof (in $\mathbb{G}_2$) and a simulated proof (in $\mathbb{G}_3$), and therefore:

$$|\mathsf{Adv}^{\mathsf{CHide}}_{\mathbb{G}_2}(\mathcal{A}) - \mathsf{Adv}^{\mathsf{CHide}}_{\mathbb{G}_3}(\mathcal{A})| \leq \mathsf{Adv}^{\mathsf{CompZK}}_{\mathsf{NIZK}}(\mathcal{R}_2).$$

In $\mathbb{G}_3$, all the values that were computed for the challenge configuration conf_b are either identical in both cases (for the commitment), or are simulated without using the exact configuration in input (NIZK proof of knowledge). Therefore the adversary's winning advantage in $\mathbb{G}_3$ is 0. This provides the required bound. □

5.5.2 Unforgeability

Theorem 5.5.2. *Let* cRA *be the privacy-preserving collective remote attestation scheme described in Section 5.4, using a signature scheme* SIG = (SIG.Setup, SIG.KeyGen, SIG.Sign, SIG.Verify)*, a commitment scheme* COM(COM.Commit, COM.Verify) *and a non-inter-active zero-knowledge proof of knowledge* NIZK = (NIZK.Setup, NIZK.Pr, NIZK.Ver)*. If the signature scheme is EUF-CMA, then* D-cRA *is unforgeable. More specifically, assume that there exists an adversary $\mathcal{A}$ winning the unforgeability game ($G_{\mathsf{UF}}(\lambda)$) against a graph on n nodes with advantage* $\mathsf{Adv}^{\mathsf{UF}}_{\mathtt{cRA}}(\mathcal{A})$*. Then there exists an adversary (reduction) $\mathcal{R}$ against the EUF-CMA of the signature scheme* SIG *winning with advantage* $\mathsf{Adv}^{\mathsf{EUF-CMA}}_{\mathtt{SIG}}(\mathcal{R})$*, such that:*

$$\mathsf{Adv}^{\mathsf{UF}}_{\mathtt{cRA}}(\mathcal{A}) \leq n \cdot \mathsf{Adv}^{\mathsf{EUF-CMA}}_{\mathtt{SIG}}(\mathcal{R}).$$

Proof. To understand why unforgeability, recall that, in the security game, the adversary will be required to forge a valid attestation or aggregate on behalf of a node node_i in the graph, without obtaining the aggregate or attestation from a direct oracle query. Recall moreover that an attestation is a triple $\mathsf{a} = (\pi_{\mathsf{a}}, \sigma_{\mathsf{a}}, \mathsf{c})$, in which the second element, σ_{a} is a signature over 3 values: the commitment c, the linking information link, and the random value $\mathsf{rand}_{\mathsf{req}}$ in the request req. An aggregate is a tuple consisting of an aggregated verification bit $\mathsf{res}_{\mathsf{agg}}$ and a signature σ_{agg} over that result and the request nonce $\mathsf{rand}_{\mathsf{req}}$. While the adversary is able to corrupt any node in software, it has no access to the node's Root of Trust (and therefore its private keys). Thus, an adversary can only win the $G_{\mathsf{UF}}(\lambda)$ game if it produces a forgery on one of those two signatures.

The proof is relatively straight-forward, with a single intermediary game-hop.

$\mathbb{G}_0$**:** The original unforgeability game $G_{\mathsf{UF}}(\lambda)$.

$\mathbb{G}_1$**:** In this game, the challenger chooses randomly an integer $i^* \in \{1, \ldots, n\}$ and outputs it privately at the beginning of the game. If the adversary's final output out is of the type $(\mathsf{node}_i, \cdot, \cdot)$ with $i \neq i^*$, then the challenger outputs $\perp$ (the game fails). Else, the game is run normally. Clearly, since the challenger has a probability of $\frac{1}{n}$ to guess the adversary's target node, it holds that:

$$\mathsf{Adv}^{\mathsf{UF}}_{\mathbb{G}_0}(\mathcal{A}) \leq n \cdot \mathsf{Adv}^{\mathsf{UF}}_{\mathbb{G}_1}(\mathcal{A}) \ .$$

We now compute the adversary's success probability in game $\mathbb{G}_1$. Specifically, we construct a reduction $\mathcal{R}$ against the EUF-CMA security of the signature scheme which wins whenever adversary $\mathcal{A}$ wins $\mathbb{G}_1$.

The simulation is straight-forward. The reduction $\mathcal{R}$ receives from its EUF-CMA challenger the target public key pk. The reduction has access to a signing oracle, which takes in input messages m and outputs signatures of the type $\sigma \leftarrow \mathtt{SIG.Sign}(\mathsf{sk}, m)$. The goal of $\mathcal{R}$ is to eventually output a tuple (m^*, σ^*) such that $\mathtt{SIG.Verify}(\mathsf{pk}, m, \sigma) = 1$ and m^* was never queried to the signing oracle.

Once it receives the target public key pk, the reduction $\mathcal{R}$, playing the part of the challenger in $\mathbb{G}_1$, generates all the private and public parameters of all the parties in the modified $G_{\mathsf{UF}}(\lambda)$ of $\mathbb{G}_1$. The sole exception is during the setup of node node_{i^*}, in which the challenger instantiates all the node's private, public, configuration, and linking parameters, *except for that node's attestation key*. Instead, for node node_{i^*}, the reduction injects the public key pk received from its challenger.

For the remainder of the game, the reduction simulates $\mathbb{G}_1$ straightforwardly, except for $\mathsf{oAttest}$ and $\mathsf{oAggreg}$ queries. In the case of $\mathsf{oAttest}(\mathsf{node}_{i^*}, \mathsf{req})$ queries, the reduction computes the commitment value c and NIZK proofs of knowledge faithfully, but then has to produce a valid signature for the concatenations of the concatenation of c, link_{i^*}, and the nonce rand extracted from req. In order to do so , it sends the concatenation of those messages as a message M to its signature oracle and forwards that response as a signature, together with c and the proof π to $\mathcal{A}$. The procedure is the same for $\mathsf{oAggreg}$. Finally $\mathcal{A}$ outputs a value out which can be parsed as $(\mathsf{agg}, \mathsf{a}, \mathsf{node}_i, \mathsf{req})$. The reduction verifies the signatures σ_{a} in a and σ_{agg} in the aggregate agg. If at least one of these signatures verifies, and it was not produced by the $\mathsf{oAttest}$ or $\mathsf{oAggreg}$ oracles, then $\mathcal{R}$ outputs that verifying, fresh signature to its challenger. Else, it outputs a random value as a forgery.

For the analysis, note that the reduction simulates the game perfectly for $\mathcal{A}$. If the latter wins, then it is able to output either a verifying and fresh σ_{a} or σ_{agg}. In that case, however, $\mathcal{R}$ also wins.
As a result $\mathsf{Adv}^{\mathsf{UF}}_{\mathbb{G}_1}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathtt{SIG}}(\mathcal{R})$, giving the required bound. □

5.5.3 Linking

Theorem 5.5.3. *Let* cRA *be the privacy-preserving collective remote attestation scheme described in Section 5.4, using a signature scheme* SIG = (SIG.Setup, SIG.KeyGen, SIG.Sign, SIG.Verify)*, a commitment scheme* $\mathtt{COM}(\mathtt{COM.Commit}, \mathtt{COM.Verify})$ *and a non-inter-active zero-knowledge proof of knowledge* $\mathsf{NIZK} = (\mathtt{NIZK.Setup}, \mathtt{NIZK.Pr}, \mathtt{NIZK.Ver})$*. If the signature scheme is EUF-CMA, then* D-cRA *is unforgeable. More specifically, assume that there exists an adversary* $\mathcal{A}$ *winning the unforgeability game (*$G_{\mathsf{LK}}(\lambda)$*) against a graph on* n *nodes with advantage* $\mathsf{Adv}^{\mathsf{LK}}_{\mathtt{cRA}}(\mathcal{A})$*. Then there exist adversaries (reductions)* $\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3$ *against the EUF-CMA of the signature scheme* SIG *winning with advantage* $\mathsf{Adv}^{\mathsf{EUF-CMA}}_{\mathtt{SIG}}(\mathcal{R}_1)$*, and respectively* $\mathsf{Adv}^{\mathsf{EUF-CMA}}_{\mathtt{SIG}}(\mathcal{R}_2)$ *and* $\mathsf{Adv}^{\mathsf{EUF-CMA}}_{\mathtt{SIG}}(\mathcal{R}_3)$*, such that:*

$$\mathsf{Adv}^{\mathsf{LK}}_{\mathtt{cRA}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{EUF-CMA}}_{\mathtt{SIG}}(\mathcal{R}_1) + n \cdot \mathsf{Adv}^{\mathsf{EUF-CMA}}_{\mathtt{SIG}}(\mathcal{R}_2) + \mathsf{Adv}^{\mathsf{EUF-CMA}}_{\mathtt{SIG}}(\mathcal{R}_3).$$

Proof. To understand why this theorem holds, recall that linking is meant to ensure that graph nodes (representing VNFs and hypervisors) are linked in specific ways in order to ensure specific functionalities of the VNF-FG. This often-contractual obligation is violated if, following adversarial intervention, some VNFs are no longer managed by the expected hypervisor, or a node (VNF or hypervisor) is suppressed. Thus, in the linking game, the adversary is allowed to modify the original, honestly-chosen graph to a different, convenient configuration. The adversary's goal is to ensure that the VNF-FG still produce a root attestation and aggregate that seem to be correctly linked with respect to the original, unmodified graph (which models the expected configuration).
Intuitively, the proof of this statement relies on several constructional decisions of our schemes. A crucial role is played here by the apparently-simplistic structure of propagated attestation requests (which are sent across the spanning tree of the graph by the verifier). Two crucial elements play a part here: the fresh randomness selected by the verifier at each attestation, and the signature of the request. This ensures that on the one hand, the adversary cannot replay old attestations, obtained for the original graph, and pass them off as being attestations obtained for the new graph; and on the other hand, that an adversary cannot forge attestation requests at will for convenient nonces.

Note moreover that in the security game, we rule out an adversarial victory for a nonce that was already used for the old graph, as long as the root attestation and aggregates are yielded by oAttest or oAggreg. This saves us the trouble of an additional game hop in which we assume that honestly-generated randomness is always unique. In reality, we would, however, have to ensure that nonces are sizeable, in order to guarantee that no collusion occurs.

The next crucial construction element we employ is the linking information obtained and set up in a trusted way, which is additionally authenticated during the transmission of both attestations and aggregates. This ensures that graph modifications are identified by nodes along the spanning tree of the graph.

More formally, we prove the statement through the following sequence of game hops.

$\mathbb{G}_0$**:** The original linking game $G_{\mathsf{LK}}(\lambda)$.

$\mathbb{G}_1$**:** We modify the original game. In $\mathbb{G}_1$, if the adversary queries oAttest or oAggreg with an input req which was not issued by oReqGen, the challenger automatically outputs $\perp$ for those two oracles. We claim that this game is indistinguishable from $\mathbb{G}_0$. Indeed, in order for the adversary to be able to tell the difference between the games, it has to forge a *legitimate, fresh* attestation request. If this is the case, then we can construct an EUF-CMA reduction $\mathcal{R}_1$ against the signature scheme employed by the verifier. The reduction injects the target public key pk received from the challenger in its simulation of the verifier. The challenger generates the rest of the graph/node parameters honestly and therefore can perfectly simulate any graph modification, attestation, and aggregation requests. For its oReqGen responses, the reduction chooses fresh randomness and queries it to the signature oracle, outputting the response to the adversary. Eventually, the adversary must output a forgery, which the reduction forwards to its challenger. Clearly the simulation is perfect and whenever $\mathcal{A}$ distinguishes between the two games, the reduction also wins. This yields the bound:

$$|\mathsf{Adv}^{\mathsf{Link}}_{\mathbb{G}_0}(\mathcal{A}) - \mathsf{Adv}^{\mathsf{Link}}_{\mathbb{G}_1}(\mathcal{A})| \leq \mathsf{Adv}^{\mathsf{EUF-CMA}}_{\mathtt{SIG}}(\mathcal{R}_1).$$

$\mathbb{G}_2$**:** We modify the game once more. In this intermediate game, the challenger aborts if the adversary uses, in input to oAttest or oAggreg any attestation or aggregate that was not output by a call to oAttest or oAggreg for the same nonce. This includes the attestation/aggregate used in the adversary's ultimate output. Clearly, this is equivalent to the successful forging of an attestation or aggregate. The equivalence between $\mathbb{G}_1$ and $\mathbb{G}_2$ can be proved through a 2-step reduction, akin to

the unforgeability game: first the challenger needs to guess on behalf of which node the forgery is made (yielding a security loss of a factor of $\frac{1}{n}$), and then constructing a reduction against the unforgeability of that node's signature scheme. This yields,

$$|\mathsf{Adv}^{\mathsf{Link}}_{\mathbb{G}_1}(\mathcal{A}) - \mathsf{Adv}^{\mathsf{Link}}_{\mathbb{G}_2}(\mathcal{A})| \leq n \cdot \mathsf{Adv}^{\mathsf{EUF-CMA}}_{\mathtt{SIG}}(\mathcal{R}_2).$$

We now analyse the adversary's winning probability (which is the same as its advantage in the case of the $G_{\mathsf{LK}}(\lambda)$ security notion). There are two cases.

NO NEW GRAPH In this case, the adversary never modifies a graph via a oModG oracle query. In this case, the adversary's only attestations and aggregates stem from attestation requests for graph $\mathcal{G}$. As per the last hop the adversary's output must stem from an oAttest or oAggreg query for the root node, for a valid request (as per our first game hop). Hence, in this game, there are two options:

- Either the adversary includes a bogus value for the request, not obtained from oReqGen, in which case the adversary fails, except for a forgery. We can construct in that case the adversary $\mathcal{R}_3$ and simulate the game as in the first game hop.
- Or the adversary includes an honestly-obtained value for req, in which case the adversary's win is not counted as legitimate (all three conditions of failure are true).

AT LEAST ONE NEW GRAPH In this case, the adversary queries oModG at least once. We divide our analysis in the following sub-cases:

- None of the subsequently-modified graphs $\mathcal{G}'$ input to oModG differ from the original graph $\mathcal{G}$. This case is equivalent to the no-new-graph case.
- At least one graph modification is true (the input graph $\mathcal{G}' \neq \mathcal{G}$). In this case, the adversary now owns attestations and aggregates for the new graph as well as for the old graph. As per the winning conditions, and game $\mathbb{G}_2$, the adversary can only win for a request req such that $(\mathcal{G}, \mathsf{req}) \notin \mathcal{D}_{\mathsf{oReqGen}}$. But, in this case, since no attestation or aggregate can be forged, the linking information included in the signatures has to be faithful to whichever graph $\mathcal{G}^*$ for which $(\mathcal{G}^*, \mathsf{req}) \in \mathcal{D}_{\mathsf{oReqGen}}$. In that case, the adversary fails as the new linking will be different from the current one.

This concludes the proof with the bound provided above. □

5.6 Implementation

We implemented the total of 10 offline and online algorithms of D-cRA in C++ and simulated D-cRA over a group of nodes (up to 2.5×10^5 nodes) using OMNet++ [55]. OMNet++ is a network simulator that can take in input a tree configuration (*i.e.*, the number of nodes and the arity) then runs a specific protocol, in our case D-cRA, on that infrastructure. This setup for an easy performance-evaluation for D-cRA over various tree configurations (as illustrated in Figure 5.8).

PoC platform details. Our tests and benchmarks were carried out on a standard laptop running Ubuntu 20.04.5 with an Intel i7-10875H CPU (16 cores) and 32GB RAM. The benchmarks were run using the Cmdenv environment of OMNeT++ 6.0.1.

Cryptographic details. We implemented the zero-knowledge set membership proof using a simple and efficient protocol due to Camenisch *et al.*[57] (rendered non-interactive via the Fiat-Shamir heuristic). The protocol is pairing based, and we used the BLS 12-381 curve implementation of mcl [58]. For the signature scheme, we used Libsodium [59] ed25519.

Memory cost. We denote by $|\mathsf{confset}|$ the number of element in a configuration set. After the offline phase of D-cRA, each node stores at least its signing key pair $(\mathsf{sk},\mathsf{pk})$ (32 and 32 bytes), the public parameter for the proof $\mathsf{ppar}_{\mathtt{NIZK}}$ $(864 + 48 * |\mathsf{confset}|)$, the verifier public key $\mathsf{pk}_{\mathsf{ver}}$ (32 bytes), and for every child node the following tuple: its public key, its linking information link[6] and its associated $\mathsf{confset}$ ($32+32+48 \cdot |\mathsf{confset}|_i$ bytes for neighbour i). Thus a node must store at least $960 + 48 \cdot |\mathsf{confset}| + \sum_{i=1}^{n-1} 64 + 48 \cdot |\mathsf{confset}|_i$ bytes of information for n neighbours.

Online communication cost. A considerable advantage of collective attestation is that it greatly reduces bandwidth which is critical in an NFV use case. In our scheme D-cRA, attestation messages are of constant size. Indeed every node sends at most its own attestation and one aggregate. An attestation is made up of a proof (816 bytes), a commitment (64 bytes) and a signature (64 bytes) whereas an aggregate is simply one single bit and a signature (64 bytes). Thus we have a total constant size of 8065 bits so around 1 kilo bytes.

[6]The linking information is a set of public keys of the linked components. In order to have a constant size, this can be implemented by the Hash of the public keys set.

Computational Load per Node. In Table 5.2, we detail the time (in ms) for one attestation computation and verification, and an ed25519 signature computation and verification (this will be later considered for the aggregate computation and verification). In these benchmarks, we used the google benchmark library [60] and set $|\mathsf{confset}|$ (the size of the set of configuration) to 100.

Table 5.2: Time in ms to perform one attestation and one ed25519 signature.

	Mean	Median
Attestation	1.56	1.55
Attestation Verification	2.38	2.37
Signature	0.023	0.023
Signature Verification	0.057	0.057

In Table 5.3, we compare our result with SEDA [22] one of the first collective remote IoT remote attestation protocols.

Table 5.3: Time in ms for a single node to compute an attestation report depending on the number of neighbours n.

D-cRA	SEDA
$1.64 + 2.437 \cdot (n-1)$	$0.6 + 4.4 \cdot (n-1)$

VNF-FG Attestation Performance. In order to evaluate the performance of our scheme, we set $|\mathsf{confset}| = 100$, the communication time between two neighbour nodes to 1ms, with variable number of children nodes (for trees that range from 2-ary to 8-ary) and the tree height. Although they are unlikely to occur in practice, we considered –for test purposes– complete trees, allowing us to vary the number of nodes up to 2.5×10^5. For each combination, we ran multiple trials and plotted the mean value in Figure 5.8. These results show that (i) attesting a VNF-FG consisting of thousands of VNFs does not exceed $200ms$ and (ii) a VNF-FG attestation time is logarithmic in the number of nodes, making our scheme D-cRA very efficient and scalable.

5.7 Conclusion

In this Chapter, we proposed a method to attest a large group of VNFs forming a VNF-FG, while also guaranteeing layer-linking: ensuring that every VNF is linked to

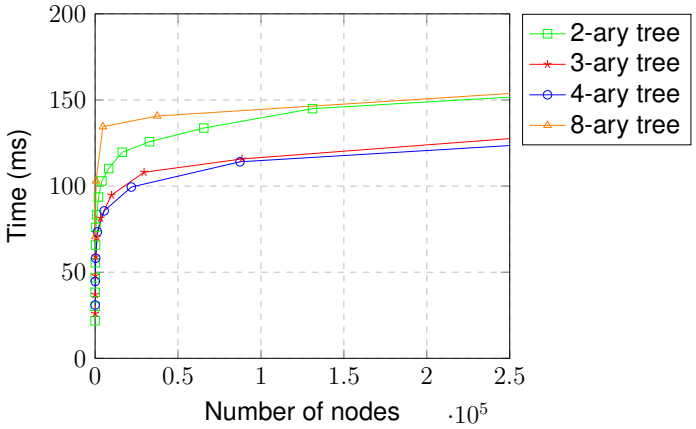


Figure 5.8: VNF-FG attestation time

the correct hypervisor. More importantly, our protocol preserves the *privacy* of each node's actual configuration through the use of set membership proofs. This renders our protocol suitable for multi-tenant VNF-FGs in which multiple operators collaborate in order to provide a full network service.

Our scheme leverages deep-attestation, property-based attestation, and collective remote attestation, and achieves *provable, formal privacy and security*: unforgeability, layer-linking, and configuration-hiding. Our proof-of-concept implementation demonstrates the efficiency of our scheme and demonstrates that privacy can be achieved without a loss of scalability.

Our protocol currently provides a yes/no verification and linking output: yes, the entire VNF-FG is in a valid state, no, it is not. A possible extension could identify which devices failed their attestation. This would involve including, in the aggregates, the identifiers corresponding to failed attestations/aggregates. However, this would increase bandwidth costs if attestation fails and could be used by adversaries to cause congestion.

Conclusion

NETWORK Infrastructures are rapidly evolving from a static to a virtualised and flexible architecture. While this evolution is necessary in order to meet new demands and absorb an eve increasing traffic, it raises serious security challenges. Remote attestation is a potentially powerful tool in this context despite the lack of construction tailored or even suitable for virtualized networks. In this thesis, we have made several contributions along this line, and hope that our work will prove usable for practitioners and seminal in future researchs.

Contributions

Our first contribution is a simple deep attestation protocol (authorized linked attestation), which can be seen as an improvement on the ETSI multi-channel deep attestation approach. It remains as efficient as the multi-channel approach but also includes a linking mechanism, so that a verifier can check that two attestations are linked. In our case this means that a verifier can check that a VM attestation is linked to a hypervisor attestation and therefore that the VM is actually running on that hypervisor. From a cryptographic point of view, linking is achieved by adding some public keys in the attestation report, which is then signed. The hypervisor includes all the keys of the currently running VMs, and each VM includes its own public key in the attestation. The verifier simply checks that there is a match. This approach allows for a modular suite of formal definitions, which provide gradual properties, including authentication, authorization, and provable layer-linking. In addition, we demonstrated the practicality of the protocol through a Proof of Concept implementation, which allowed to attest 50 VMs in less than 8 seconds. This basic protocol provides the best of both worlds : the parallelizable efficiency of multiple-channel attestation and the layer-linked security of single-channel attestation. However, it assumes that all the involved entities (*i.e.*, the hypervisor, the VMs and the verifier) are operated by a single operator.

Our second advanced use case is attesting a multi-tenant infrastructure where a provider hosts multiple VMs owned by different tenants. The purpose is to enable each tenant to deep attest its own VMs (*i.e.*, VMs, hypervisor, and layer linking) without leaking any information about the other tenants. In this context, implementing an efficient deep attestation mechanism poses more challenges. The straightforward concept of multiple-channel (*i.e.*, separating VM attestation from hypervisor attestation, where the hypervisor would only require a single attestation) is not feasible and compromises the configuration privacy of the hypervisor. Instead, we used a batching approach to provide a hypervisor attestation to multiple tenants with a single call to the RoT.

We also had to address privacy concerns. We used an approach where the prover shows that it has a valid configuration among a set of possible ones, without revealing which one, to reduce information leakage about the provider hypervisor. This is possible thanks to zero-knowledge proofs. Linking attestations in a multi-tenant use case also leaks information about each VM on a hypervisor. We used the hiding property of vector commitments to avoid any such leakage and obtain tenant privacy. We defined two novel formal privacy properties (inter-tenant privacy and configuration hiding) and modified the linking property. To account for multiple tenants our protocol adheres to these properties. To make our proofs work we had to introduce two new properties, one for vector commitments (collision resistance) and one for authenticated key exchange (partner-hiding). This proves that achieving privacy-preserving, linkable deep attestation for multiple tenant environments is non trivial and requires careful consideration. We proved that Merkle tree-based vector commitment schemes and TLS 1.3 satisfy collision resistance and partner-hiding, respectively. We also implemented the protocol, which can handle requests from 500 tenants in 2.5 seconds (given an incompressible TPM attestation of 1s).

Finally, we focus on a broader use-case: attesting a large infrastructure (called a VNF-FG) of inter-dependent VNFs, belonging to various entities. This use-case touches upon the real-world complexity of operating a Telco Cloud which aims to provide whole, complex network services, rather than just a single VNF. With a single attestation, we can verify that all the elements that make up a network service are in an appropriate state. This greatly reduces the network traffic caused by attestation messages. Using methods borrowed from collective attestation in the context of IoT and techniques from our two previous schemes, we designed an efficient and privacy-preserving network service attestation protocol. Likewise, we provided a formal model with proofs and an implementation with experimental results showing that we can attest a network services made of 250000 targets in less than 150ms.

Perspectives

Our schemes answer to concrete scenarios of Telco Cloud attestation. However, there are still questions (both technical and operational) that need to be answered in order to make remote attestation fully operational in virtualized networks. We end this thesis by highlighting some interesting research areas to consider as future work.

VM Migration. VM migration allows a VM to move seamlessly from one hypervisor to another. Live migration even allows VMs to be moved while they are running, with

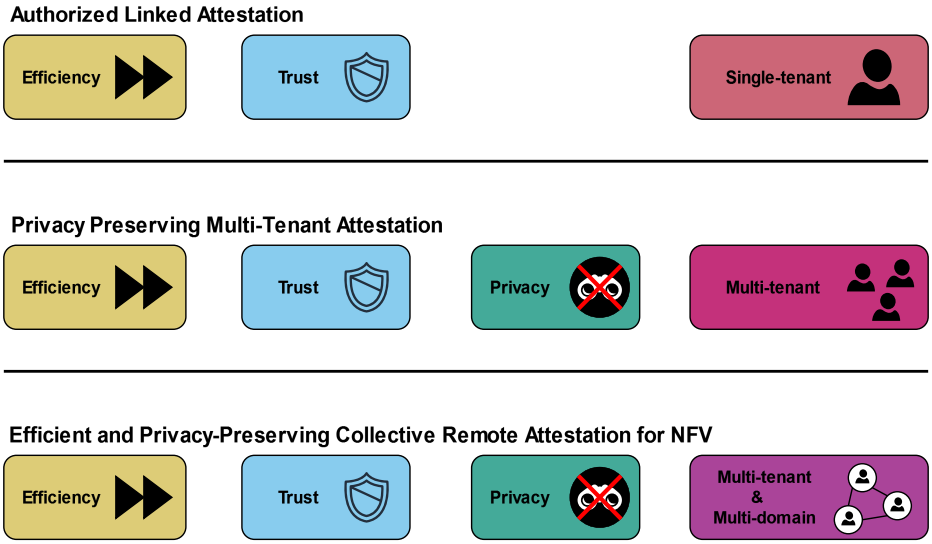


Figure 5.9: A summary of our three protocols.

downtime low enough to be unnoticeable to users. This technology is essential to provide flexibility in NFV environment. Yet, in terms of attestation, migration comes with certain challenges. There is some work on the subject [61], but it has not yet been explored in the context of NFV. As we have seen throughout this thesis, privacy is critical for several NFV use cases. However, most existing protocols start with a mutual attestation between the source and the destination, which in our case could be operated by different entities. Therefore, this process needs to be privacy preserving. Another problem of existing works is that they mostly consider TPM attestation, but other form of attestation and TEE are valuable in the context of NFV.

Beyond Virtual Machines. All the protocols presented in this work are defined for VNF, which are in fact virtual machines. However, these protocols might be not usable when virtualization is not based on virtual machines. We illustrate this problem with the case of containers which can also be used to package network function (and sometimes called CNF, for Container Network Function, to distinguish them from VNF). Depending on the underlying RoT, our protocols can be used transparently whether we are using VMs or containers, but for others this is not possible. This is particularly the case with the TPM. TPM attestation is about measuring the boot sequence, but a container does not boot like a physical machine. With containers, attestation only makes sense using the Integrity Measurement Architecture (IMA) [7].

References Value Management. Throughout our work, we abstracted the notion

[7] IMA is a kernel module which allows to extend measurement beyond the kernel.

of configuration, designing our protocols to be agnostic of the precise state details. Because we're focusing on the protocol, we could afford not to look at the details of configuration. However, in practice we need to define what a validation configuration is to be able to produce the reference values in order to verify the attestation. We could wonder whether one could automatically define what are the reference values, or whether they can be updated automatically when updating the system. Reference values management has been a recurring problem since the introduction of remote attestation and there are still no satisfactory solutions.

Standards. Applying attestation implies some form of cooperation between entities playing various roles (*e.g.*, verifier, attester, etc). Unfortunately, this can be seen as a major barrier for attestation widespread. But standardising the exchange protocols and messages formats will enable a shorter time to market. Some works in this direction are emerging. For example the IETF [62] proposes a format of attestation evidence but there is still a long way to go.

Bibliography

Contents

References

[1] ARCEP, “The state of the Internet in France,” Tech. Rep., 2022. [Online]. Available: https://en.arcep.fr/uploads/tx_gspublication/report-state-internet-2022-300622.pdf.

[2] G. Arfaoui, P.-A. Fouque, T. Jacques, *et al.*, “A Cryptographic View of Deep-Attestation, or How to Do Provably-Secure Layer-Linking,” in *Applied Cryptography and Network Security*, 2022.

[3] G. Arfaoui, T. Jacques, M. Lacoste, C. Onete, and L. Robert, “Towards a Privacy-Preserving Attestation for Virtualized Networks,” in *Computer Security – ESORICS 2023*, 2023.

[4] G. Arfaoui, T. Jacques, and C. Onete, “Efficient and privacy-preserving collective remote attestation for NFV,” in *NCA*, 2024.

[5] ISG-NFV, “Network Functions Virtualisation (NFV); Architectural Framework,” European Telecommunications Standards Institute, Tech. Rep., 2014. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.02.01_60/gs_NFV002v010201p.pdf.

[6] E. Nikolouzou, G. Milenkovic, G. Bafoutsou, S. Bryska, M. Hajj, and C. Loiseaux, “NFV Security in 5G Challenges and Best Practices,” European Union Agency for Cybersecurity, Tech. Rep., 2022. [Online]. Available: https://www.enisa.europa.eu/publications/nfv-security-in-5g-challenges-and-best-practices.

[7] ISG-NFV, “Network Functions Virtualisation (NFV); NFV Security; Problem Statement,” European Telecommunications Standards Institute, Tech. Rep., 2014. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/nfv-sec/001_099/001/01.01.01_60/gs_nfv-sec001v010101p.pdf.

[8] S. Pearson, *Trusted Computing Platforms: TCPA Technology in Context*. Prentice Hall PTR, 2002.

[9] M. Sabt, M. Achemlal, and A. Bouabdallah, “Trusted Execution Environment: What It is, and What It is Not,” in *2015 IEEE Trustcom/BigDataSE/ISPA*, 2015.

[10] N. Agarwal and K. Paul, “XEBRA: XEn Based Remote Attestation,” in *2016 IEEE Region 10 Conference (TENCON)*, 2016.

[11] B. Vetter and D. Westhoff, “Simulation study on code attestation with compressed instruction code,” in *2012 IEEE International Conference on Pervasive Computing and Communications Workshops*, 2012.

[12] R. Vieira Steiner and E. Lupu, “Towards more practical software-based attestation,” *Computer Networks*, 2019.

[13] S. Berger, R. Caceres, K. A. Goldman, R. Perez, R. Sailer, and L. van Doorn, “vTPM: Virtualizing the Trusted Platform Module,” in *15th USENIX Security Symposium (USENIX Security 06)*, 2006.

[14] ISG-NFV, “Network Functions Virtualisation (NFV); Trust; Report on Attestation Technologies and Practices for Secure Deployments,” European Telecommunications Standards Institute, Tech. Rep., 2017. [Online]. Available: https://www.etsi.org/deliver/etsi_gr/NFV-SEC/001_099/007/01.01.01_60/gr_nfv-sec007v010101p.pdf.

[15] “Virtualized Trusted Platform Architecture Specification,” TCG Virtualized Platform Working Group, Tech. Rep., 2011.

[16] A.-R. Sadeghi and C. Stüble, “Property-Based Attestation for Computing Platforms: Caring about Properties, Not Mechanisms,” in *Proceedings of the 2004 Workshop on New Security Paradigms*, 2004.

[17] L. Chen, R. Landfermann, H. Löhr, M. Rohe, A.-R. Sadeghi, and C. Stüble, “A Protocol for Property-Based Attestation,” in *Proceedings of the First ACM Workshop on Scalable Trusted Computing*, 2006.

[18] L. Chen, H. Löhr, M. Manulis, and A.-R. Sadeghi, “Property-Based Attestation without a Trusted Third Party,” in *Information Security*, 2008.

[19] Y. Fajiang, C. Jing, X. Yang, Z. Jiacheng, and Z. Yangdi, “An efficient anonymous remote attestation scheme for trusted computing based on improved CPK,” *Electronic Commerce Research*, 2019.

[20] “Trusted Platform Module Library Specification, Family “2.0”, Level 00, Revision 01.59,” Trusted Computing Group, Tech. Rep., 2019.

[21] E. Brickell, J. Camenisch, and L. Chen, “Direct Anonymous Attestation,” in *Proceedings of the 11th ACM Conference on Computer and Communications Security*, 2004.

[22] N. Asokan, F. Brasser, A. Ibrahim, *et al.*, “SEDA: Scalable Embedded Device Attestation,” in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015.

[23] M. Ambrosin, M. Conti, A. Ibrahim, G. Neven, A.-R. Sadeghi, and M. Schunter, "SANA: Secure and Scalable Aggregate Network Attestation," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016.

[24] X. Carpent, K. ElDefrawy, N. Rattanavipanon, and G. Tsudik, "Lightweight Swarm Attestation: A Tale of Two LISA-s," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, 2017.

[25] M. Ambrosin, M. Conti, R. Lazzeretti, M. M. Rabbani, and S. Ranise, "PADS: Practical Attestation for Highly Dynamic Swarm Topologies," in *2018 International Workshop on Secure Internet of Things (SIoT)*, 2018.

[26] F. Kohnhäuser, N. Büscher, and S. Katzenbeisser, "SALAD: Secure and Lightweight Attestation of Highly Dynamic and Disruptive Networks," in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, Association for Computing, 2018.

[27] F. Kohnhäuser, N. Büscher, and S. Katzenbeisser, "A Practical Attestation Protocol for Autonomous Embedded Systems," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2019.

[28] E. Dushku, M. M. Rabbani, M. Conti, L. V. Mancini, and S. Ranise, "SARA: Secure Asynchronous Remote Attestation for IoT Systems," *IEEE Transactions on Information Forensics and Security*, 2020.

[29] L. Petzi, A. E. B. Yahya, A. Dmitrienko, G. Tsudik, T. Prantl, and S. Kounev, "SCRAPS: Scalable Collective Remote Attestation for Pub-Sub IoT Networks with Untrusted Proxy Verifier," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022.

[30] C. E. Shannon, "Communication theory of secrecy systems," *The Bell System Technical Journal*, 1949.

[31] D. Catalano and D. Fiore, "Vector Commitments and Their Applications," in *Public-Key Cryptography – PKC 2013*, 2013.

[32] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof-systems," in *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, 1985.

[33] O. Goldreich, S. Micali, and A. Wigderson, "Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems," *J. ACM*, 1991.

[34] A. Fiat and A. Shamir, "How To Prove Yourself: Practical Solutions to Identification and Signature Problems," in *Advances in Cryptology — CRYPTO' 86*, 1987.

[35] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, “From Extractable Collision Resistance to Succinct Non-Interactive Arguments of Knowledge, and Back Again,” in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, 2012.

[36] T. Jager, F. Kohlar, S. Schäge, and J. Schwenk, “On the Security of TLS-DHE in the Standard Model,” in *Advances in Cryptology – CRYPTO 2012*, 2012.

[37] H. Krawczyk, K. G. Paterson, and H. Wee, “On the Security of the TLS Protocol: A Systematic Analysis,” in *Advances in Cryptology – CRYPTO 2013*, 2013.

[38] *Qemu*. [Online]. Available: https://www.qemu.org/.

[39] *Libtpms*. [Online]. Available: https://github.com/stefanberger/libtpms.

[40] *Swtpm*. [Online]. Available: https://github.com/stefanberger/swtpm.

[41] *Tpm2-software*. [Online]. Available: https://github.com/tpm2-software.

[42] ISG-NFV, “Network Functions Virtualisation (NFV); Use Cases,” European Telecommunications Standards Institute, Tech. Rep., 2021. [Online]. Available: https://www.etsi.org/deliver/etsi_gr/NFV/001_099/001/01.03.01_60/gr_NFV001v010301p.pdf.

[43] *Keylime*, 2022. [Online]. Available: https://github.com/keylime.

[44] N. Schear, P. T. Cable, T. M. Moyer, B. Richard, and R. Rudd, “Bootstrapping and Maintaining Trust in the Cloud,” in *Proceedings of the 32nd Annual Conference on Computer Security Applications*, 2016.

[45] K. Foy, *Keylime software is deployed to IBM cloud*, 2021. [Online]. Available: https://news.mit.edu/2021/keylime-security-software-deployed-ibm-cloud-0727.

[46] N. Santos, R. Rodrigues, K. P. Gummadi, and S. Saroiu, “Policy-Sealed Data: A New Abstraction for Building Trusted Cloud Services,” in *21st USENIX Security Symposium (USENIX Security 12)*, 2012.

[47] M. Schneider, R. J. Masti, S. Shinde, S. Capkun, and R. Perez, *Sok: Hardware-supported trusted execution environments*, 2022.

[48] G. Arfaoui, X. Bultel, P. Fouque, A. Nedelcu, and C. Onete, “The privacy of the TLS 1.3 protocol,” *PoPETs*, 2019.

[49] Tierion, *Pymerkletools*, 2022. [Online]. Available: https://github.com/Tierion/pymerkletools.

[50] Filecoin, *Bellperson*, 2022. [Online]. Available: https://github.com/filecoin-project/bellperson.

[51] J. Groth, “On the Size of Pairing-Based Non-interactive Arguments,” in *Advances in Cryptology – EUROCRYPT 2016*, 2016.

[52] S. Setty, *Bellperson-nonnative*, 2022. [Online]. Available: https://crates.io/crates/bellperson-nonnative.

[53] A. Jorge and B. Heisler, *Criterion*, 2022. [Online]. Available: https://github.com/bheisler/criterion.rs.

[54] IETF, *IAB Statement on the risks of attestation of software and hardware on the open internet*, 2023. [Online]. Available: https://datatracker.ietf.org/doc/statement-iab-statement-on-the-risks-of-attestation-of-software-and-hardware-on-the-open-internet/.

[55] OMNeT++, *OMNeT++*, 2023. [Online]. Available: https://omnetpp.org/.

[56] J. Groth, "Simulation-Sound NIZK Proofs for a Practical Language and Constant Size Group Signatures," in *Advances in Cryptology - ASIACRYPT*, 2006.

[57] J. Camenisch, R. Chaabouni, and a. shelat abhi, "Efficient Protocols for Set Membership and Range Proofs," in *Advances in Cryptology - ASIACRYPT 2008*, 2008.

[58] S. Mitsunari, *Mcl*, 2023. [Online]. Available: https://github.com/herumi/mcl.

[59] Libsodium, *Libsodium*, 2023. [Online]. Available: https://doc.libsodium.org/.

[60] Google, *Benchmark*, 2023. [Online]. Available: https://github.com/google/benchmark.

[61] B. Danev, R. J. Masti, G. O. Karame, and S. Capkun, "Enabling secure VM-vTPM migration in private clouds," in *Proceedings of the 27th Annual Computer Security Applications Conference*, 2011.

[62] H. Birkholz, D. Thaler, M. Richardson, N. Smith, and W. Pan, *Remote ATtestation procedureS (RATS) Architecture*, RFC 9334, 2023.

Publications

Peer-Reviewed International Conference

- G. Arfaoui, P.-A. Fouque, T. Jacques, *et al.*, “A Cryptographic View of Deep-Attestation, or How to Do Provably-Secure Layer-Linking,” in *ACNS*, 2022.
- G. Arfaoui, T. Jacques, M. Lacoste, C. Onete, and L. Robert, “Towards a Privacy-Preserving Attestation for Virtualized Networks,” in *ESORICS*, 2023.
- G. Arfaoui, T. Jacques, and C. Onete, “Efficient and privacy-preserving collective remote attestation for NFV,” in *NCA*, 2024.

Protocoles d'attestation dans les environnements dynamiques

Résumé : La virtualisation des fonctions réseau (NFV) est une approche de la mise en oeuvre d'un réseau dans laquelle les fonctions réseau sont implémentées sous forme de logiciels s'exécutant au sein de compartiments virtuels (comme une machine virtuelle ou un conteneur) plutôt que sous forme de matériel dédié. Il en résulte une architecture de type cloud où les fonctions réseau virtuelles (VNF) sont gérées sur une plateforme appelée infrastructure de virtualisation des fonctions de réseau (NFVI).
Le paradigme NFV apporte de la flexibilité supplémentaire au réseau. Les opérateurs peuvent facilement adapter leur réseau en ajoutant, supprimant, ou déplaçant des VNF entre les serveurs pour déployer de nouveaux services à la demande. Toutefois, cette flexibilité s'accompagne de défis en matière de sécurité. L'Institut européen des normes de télécommunication (ETSI) recommande l'utilisation de l'attestation à distance comme l'un des outils permettant de sécuriser une telle infrastructure.
Un protocole d'attestation est un protocole cryptographique bipartite dans lequel un prouveur fournit la preuve d'une ou plusieurs propriétés à un vérificateur. L'attestation approfondie étend le concept d'attestation à un environnement virtuel, dans lequel l'instance virtuelle et l'infrastructure sous-jacente sont attestées. Bien que l'attestation à distance, et dans une moindre mesure, l'attestation approfondie, ne soient pas des techniques nouvelles, les approches actuelles ne sont pas adaptées aux environnements NFV. Cette thèse introduit trois schémas d'attestation adaptés au contexte NFV. La sécurité de chacun de ces protocoles est formellement prouvée. De plus, nous démontrons l'efficacité de ces protocoles à l'aide d'expérimentations réalisées sur des implémentations.
Un premier schéma formalise le concept d'attestation approfondie, tout en améliorant l'état de l'art avec une solution qui offre un compromis entre la sécurité et l'efficacité des deux principales approches existantes. Sur la base de cet élément de base, deux schémas sont présentés pour répondre à des situations concrètes. Ces schémas prennent en compte les problèmes de confidentialité qui se posent dans les réseaux virtualisés multi-tenant, tout en étant efficaces même sur de grandes infrastructures et en conservant les garanties de sécurité du schéma de base.

Mots clés : Attestation, Attestation approfondie, NFV, Multi-tenant, Respectueux de la vie privée, Protocoles cryptographiques.

Attestation protocols in dynamic environments

Abstract: Network Functions Virtualisation (NFV) is a networking paradigm where network functions are implemented as software running inside virtualised instances (such as virtual machines or containers) rather than on dedicated hardware. This results in a cloud-like architecture where the virtual network functions (VNFs) are managed on a platform called the Network Function Virtualisation Infrastructure (NFVI).

The use of NFVs adds more flexibility to the network. Operators can easily adapt their network by adding, removing, or moving VNFs between servers to scale up or down, or to deploy new services on demand. However, with this flexibility come security challenges. The European Telecommunications Standards Institute (ETSI) recommends the use of remote attestation as one of the tools to secure such an infrastructure.

An attestation protocol is a two-party cryptographic protocol in which a prover provides evidence about one or more properties to a verifier. Deep attestation extends the concept of attestation to a virtualized environment, where both the virtual instance and the underlying infrastructure are attested. Although remote attestation, and to a lesser extent deep attestation, are not new techniques, current approaches are not suitable for the NFV environment. In this thesis, we propose three privacy-preserving attestation protocols for the NFV context. We formally prove the security of our proposals. In addition, we demonstrate the effectiveness of our protocols with implementations and experimental results.

A first scheme establishes the concept of deep attestation and enhances the current state-of-the-art by providing a solution that strikes a balance between the security and performance of the two predominant existing methods. Based on this building block, we introduce two new schemes that respond to more complex use cases. These schemes take into account the privacy issues that arise in multi-tenant virtualised networks, while being efficient even on large infrastructures and maintaining the security guarantees of the basic building block.

Keywords: Attestation, Deep attestation, Collective attestation, NFV, Multi-tenant, Privacy-preserving, Cryptographic protocols.

Univ. de Limoges, CNRS, XLIM, UMR 7252, F-87000 Limoges, France