Efficient and Privacy-Preserving Collective Remote Attestation for NFV

Ghada Arfaoui[1], Thibaut Jacques[1,2], and Cristina Onete[2]

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

Abstract. The virtualization of network functions is a promising technology, which can enable mobile network operators to provide more flexibility and better resilience for their infrastructure and services. Yet, virtualization comes with challenges, as 5G operators will require a means of verifying the state of the virtualized network components (*e.g.*, Virtualized Network Functions (VNFs) or managing hypervisors) in order to fulfill security and privacy commitments. One such means is the use of attestation protocols. In this paper, we focus on Collective Remote Attestation (cRA), which is used to attest the state of *a group of devices*. Although cRA has been extensively studied in the context of IoT, it has not been used yet in virtualized mobile networks – a different use-case, with constraints of its own.
In this paper, we propose the first protocol to efficiently and securely attest a group of Virtualized Network Functions which make up a VNF Forwarding Graph. Our protocol comes with *strong and provable* guarantees of: unforgeability of attestation, the linkability of attestations for related components, and the privacy of sensitive configuration details for the infrastructure provider. In particular, we are the first to formally define and analyze such properties for VNF-FG attestation. Finally, through our Proof-of-Concept implementation, we show that our construction is not only strongly secure, but also efficient.

Keywords: Collective Remote Attestation · Telco Networks · NFV · Multi-Tenant · Privacy

1 Introduction

Network Function Virtualization (NFV) is an approach in which network functions run within virtual appliances (such as virtual machines) rather than on a dedicated hardware. To provide fully-fledged network services, NFV relies on the concept of VNF-FG: a set of Virtualized Network Functions (VNFs) connected together and modelled as a graph, with an entry and an exit point (left graph in Figure 3).

The European Telecommunications Standards Institute (ETSI) defined the notion of VNF Forwarding Graph (VNF-FG) as "*a graph of logical links connecting Network Function (NF) nodes – where at least one node is a Virtualized NF (VNF) – for the purpose of describing traffic flow between these network functions*" [16]. In common terms, if a single network functionality (*e.g.*, user authentication) can be run through one VNF, complex functionalities (like on-demand banking) require the composition of various VNF in a structure called a forwarding-graph. The VNF-FG's virtual components, be they VNFs or hypervisors, could belong to several separate entities. In this context, the operator needs a means of efficiently and securely verifying the state of all the nodes of a VNF-FG in order to guarantee its business commitments – whilst at the same time, the owners of those components are entitled to the privacy of their devices' particular setup and configurations.

A means of remote verification of virtual components is *remote attestation*, which allows a target to prove to a verifier that it satisfies some properties, such as code integrity or the execution of some code in a given (secure) environment. At a minimum, remote attestation must be able to detect tampering on the code of a virtual machine; however, when complex infrastructure constraints are present (such as multi-tenancy or hybrid infrastructures including virtual components belonging to several operators), additional strong properties such as *layer linking*, or *privacy* become paramount.

Attesting a group of nodes is known as *Collective Remote Attestation (cRA)*. Although prominent in the context of IoT [5,1,7,2,18,25,19,10,21,24], collective remote attestation has never been investigated for VNF-FGs – which is unfortunate, as the two use-cases are only superficially similar and solutions for IoT swarm-attestation will be suboptimal at best, and unusable at worst, in the context of NFV.

One of the main differences between the two scenarios lies in the complex infrastrcture underlying VNF-FG, which is absent in the IoT case. Indeed, attesting any VNF typically implies attesting underlying components such as the hosting hypervisor and *layer-linking* the two types of attestations (thus proving that the VNF is managed by that hypervisor). Layer-linking is a fundamental property of Deep Attestation (DA), but irrelevant in the context of IoT.

Another specificity of VNF-FGs is that VNFs and their managing hypervisors may belong to different owners. Yet, classical remote attestation typically leaks potentially-sensitive state information (such as kernel version, the presence of a particular type of IDS, the brand name of some software or hardware components etc.), which the infrastructure owner might want to hide from the competition. By contrast, in IoT Collective Remote Attestation, neighbours share their exact states.

1.1 Our contributions

In this paper we address the complex, but critical challenge of designing remote attestation for NFV technology, and in particular, for VNF-FGs.

Our protocol **D-cRA** is efficient, provably-secure, and achieves all the critical properties stated above:

- The verifier can ascertain the *valid state* of all virtual components (VNFs and hypervisors) in the VNF-FG;
- The verifier can ascertain the *layer-linking* between VNFs and hypervisors;
- The scheme guarantees *configuration-hiding*: exact state details (*i.e.*, the configuration) of VNFs and hypervisors will be kept hidden.

By its nature a VNF-FG can be modelled as a graph. Our protocol additionally relies on the use of spanning trees, along which nodes broadcast attestation requests and collect attestations from a VNF-FG. Each node represents a component (a VNF or a hypervisor) and must attest to its parent nodes. The latter verify the attestations and forward binary aggregates of the verification results for all the child-nodes [3]. This use of intermediate unforgeable verification aggregates renders large group attestation efficient.

A crucial novelty of our scheme, which sets it apart from IoT cRA, is guaranteeing *configuration-hiding* which is a business must in multi-tenant environments, enforcing 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.*" [14]. Our protocol achieves this property through the use of zero-knowledge set-membership proofs, through which components attest that their state belongs to a public set of valid states, without revealing their exact configuration.

Finally, our paper is first to formalize the privacy and linking properties required in collective remote attestation. We build on prior recent work by Arfaoui *et al.* [3,4]; however, our definitions are novel and important contributions in an area of cybersecurity in which protocols abound, but proofs are lacking. Crucially, our protocol D-cRA provably guarantees our proposed notions.

We also empirically evaluate the performance of D-cRA by means of an implementation using OMNeT++ [23]. 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 $200ms$. Concretely, this demonstrates that even a complex infrastructure, made up of thousands of different VNFs mounted on tens or even hundreds of hypervisors, can be remotely attested in a linkable, privacy-preserving way. In this, security and privacy need not be sacrificed for the sake of efficiency.

1.2 Background and Related work

In this section we examine related work about collective attestation, as well as deep attestation and privacy-preserving attestation.

[3] If all child-node attestations are valid, then the parent's aggregate is set to 1; else, it will be set to 0

Network Function Virtualization (NFV) NFV is an approach in which network functions run within virtual appliances (such as virtual machines) rather than on a dedicated hardware. This results in a flexible and programmable network, in which Virtualized Network Functions (VNFs) can be easily added, removed, or migrated across servers to scale up or out on demand. To provide fully fledged network services, NFV rely on the concept of VNF-FG: a set of VNFs connected together and modelled as a graph, with an entry and an exit point. The NFV Management and Orchestration (MANO) interface contains a description of a graph associated with a network service. Hence it is possible to automatically deploy network services on demand. In this work we examine how to verify such a network service by deep attesting every component of the graph without disclosing any detail about its state.

Collective Remote Attestation (cRA) Collective remote attestation was introduced in 2015 by Asokan *et al.*[5] by means of SEDA. This scheme, like most of the work on cRA, focused on the attestation of IoT swarms. The basic idea of SEDA (which later will be 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 on Figure 1. Upon receiving the request, a device retrieves the measurements of its configuration, signs it and sends the result to its parent node device, which can verify the attestation (*i.e.*, the signature), aggregate the results of its children nodes, compute its own attestation and send the overall (*i.e.*, its attestation and the aggregation of its children 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 every one of their neighbours. Finally neighbours nodes exchange their expected configuration. Hence at the end of the offline phase each nodes 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 children nodes. Once a node receives the request, it can attest to its parent node in a traditional manner using the keypair established during the offline phase for secure communication. Parents nodes, when they receive their children attestations, verify it using the expected configuration and produce 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.

SEDA might have been a promising option for attesting a VNF-FG however it does not ensure "Layer linking" and "Configuration hiding" properties which are essential in an NFV use case.

An alternative solution called SANA [1] 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[7]. It also introduces a metric called Quality of Swarm Attestation (QoSA) which measure the information given by the attestation report (*i.e.*, SEDA is Binary-QoSA as it only give a binary result). LISA comes in two flavour which reach different level of QoSA: a synchronous one where report are aggregated similarly to SEDA and an asynchronous one in which devices directly forward the result upward.

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

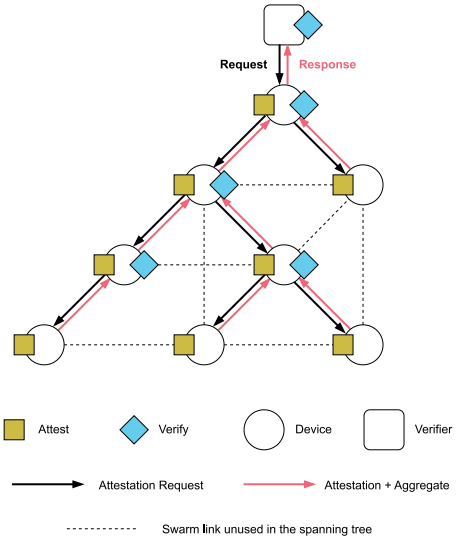

Fig. 1. SEDA overview

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 [2,18,19,10,24].

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 protocol which are not based on trees.

Deep Attestation (DA) DA consists of verifying the states of all the layers in a virtualized environment. In the context of NFV attestation, ETSI [15] describes two approaches of deep attestation: single-channel and multiple-channel DA.

In *single-channel DA*, for each VM, the verifier opens an attestation channel with the VM as shown on the left side of Figure 2. The attestation will include the VM attestation itself, as well as an attestation from the hypervisor. This method provides an instant linking between the VM and the hypervisor attestations, but does not scale with the number of VMs due to the need to perform a hypervisor attestation for each VM.

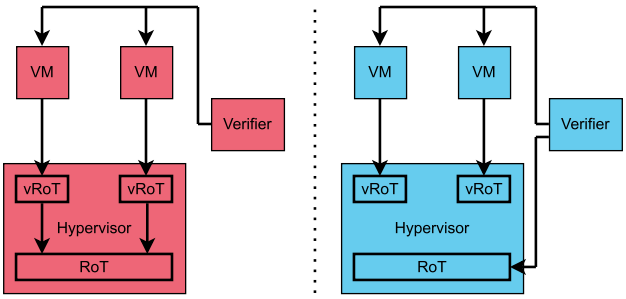

Fig. 2. Single-channel DA (left) and Multiple-channel DA (right) of two VMs

Multiple-channel DA uses a different channel to attest independently the VM and the hypervisor, as shown on the right side of Figure 2. The hypervisor will be attested only a single time; in parallel, each VM can also be attested in an independent fashion, which is impossible in single-channel DA. The single hypervisor attestation and parallelism provide much more scalability than single-channel DA, but at the cost of being able to link hypervisor and VM attestations. Yet, such

linking might prove crucial whenever the owner of the VM wants to correlate the VM with the hardware it runs on.

A *layer-linking* mechanism used to verify that VM and hypervisor attestations were issued by the same infrastructure, while guaranteeing the scalability of multiple-channel DA, was presented by Arfaoui *et al.* at ACNS 2022 [3]. In this approach, hypervisors provide the public key of the VMs they manage within their multiple-channel attestation. Later when the VM itself attests, the issued quote can be linked to that of the hypervisor attestation. The public keys included into the attestation are called linking information.

Property-based attestation (PBA) Configuration privacy should be considered when using remote attestation with a potentially untrusted verifier, as knowing details of the configuration can lead to attacks. For instance, knowing the version of the software installed on the target may allow attacker to exploit vulnerabilities specific to that version.

Property-based attestation (PBA) was hence introduced as a privacy-preserving alternative for traditional attestation [26]. 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) [26,8], used for the verification of a traditional attestation from the provers, and the latter certification 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) [9,11,4]. In this case, the prover will be able to prove that its configuration complies with certain properties without revealing its precise configuration. This is an interesting approach, though it can be tricky to implement efficiently, that we consider it in our work.

1.3 Cryptographic building blocks

In this section, we recall formal definitions and properties of cryptographic tools that we leverage in our work. We indicate the security properties we required for these schemes and we recall their formal definitions.

Signature A signature scheme $\texttt{SIG}$ is a tuple of four algorithms ($\texttt{SIG.Setup}$, $\texttt{SIG.KeyGen}$, $\texttt{SIG.Sign}$, $\texttt{SIG.Verify}$). After a setup phase ($\texttt{SIG.Setup}(1^\lambda) \rightarrow \mathsf{ppar}$) and the generation of a signature key pair ($\texttt{SIG.KeyGen}(\mathsf{ppar}) \rightarrow (\mathsf{sk}, \mathsf{pk})$), one can sign a message m using the private key sk ($\texttt{SIG.Sign}(\mathsf{sk}, m) \rightarrow \sigma$). Given a message m, a signature σ and a public key pk, one can verify the signature validity ($\texttt{SIG.Verify}(\mathsf{pk}, m, \sigma) \rightarrow \{0,1\}$). In this paper, we require a signature scheme that is Existentially Unforgeable against Chosen Message Attacks (EUF-CMA).

Definition 1 (EUF-CMA). *Let $G_{\mathsf{EUF-CMA}}(\lambda)$ be a game in which an adversary must try to forge a signature over a fresh message, while having unlimited and adaptive access to a signing oracle. Next, we define $\mathsf{Adv}^{\mathsf{EUF-CMA}}_{\mathsf{SIG}}(\mathcal{A})$ as the probability that the adversary $\mathcal{A}$ wins the game. A signature scheme is $(q_{\texttt{SIG.Sign}}, \epsilon)$-EUF-CMA if any adversary making at most $q_{\texttt{SIG.Sign}}$ 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 $\mathcal{A}$.*

Commitment A commitment scheme $\texttt{COM}$ is a triplet of algorithms ($\texttt{COM.Setup}$, $\texttt{COM.Commit}$, $\texttt{COM.Verify}$). After a setup phase ($\texttt{COM.Setup}(1^\lambda) \rightarrow \mathsf{ppar}_{\texttt{COM}}$), one can produce a commitment c based on a message m and a random string r ($\texttt{COM.Commit}(m, r) \rightarrow c$) . Given a message m, a random string r, and a commitment c, one can verify if c is a commitment of m ($\texttt{COM.Verify}(m, c, r) \rightarrow \{0,1\}$). In this paper, we require a commitment scheme that is computationally binding and hiding.

Definition 2 (Binding). *We say that a commitment scheme is* binding *if for every PPT adversary that outputs a tuple (c, m_1, r_1, m_2, r_2) the advantage $\mathsf{Adv}^{\mathsf{Bind}}_{\texttt{COM}}(\mathcal{A}) = \Pr[m_1 \neq m_2 \wedge \texttt{COM.Verify}(m_1, c, r_1) = 1 \wedge \texttt{COM.Verify}(m_2, c, r_2) = 1]$ is negligible.*

Definition 3 (Hiding). *Let $G_{\mathsf{ComHide}}(\lambda)$ be a game in which the adversary submits two messages m_0 and m_1. Depending on a bit b the challenger computes the commitment of one of them. Given adaptive access to a commitment oracle and to c, the adversary must find the value of the bit b. We define $\mathsf{Adv}^{\mathsf{Hide}}_{\mathtt{COM}}(\mathcal{A})$ 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 $\mathsf{Adv}^{\mathsf{Hide}}_{\mathtt{COM}}(\mathcal{A}) = |\Pr[\mathcal{A}$ wins $G_{\mathsf{ComHide}}(\lambda)] - \frac{1}{2}| \leq \mathsf{negl}(\lambda)$.*

Non-Interactive Zero-Knowledge Proof of Knowledge (NIZK) A NIZK scheme allows to prove knowledge of a witness w such that $(x, w) \in \mathcal{R}$ ($\mathcal{R}$ is a public binary relation) without revealing w. A NIZK is composed of three algorithms ($\mathtt{NIZK.Setup}$, $\mathtt{NIZK.Pr}$, $\mathtt{NIZK.Ver}$). After a setup phase ($\mathtt{NIZK.Setup}(1^\lambda) \rightarrow \mathsf{ppar}_{\mathtt{NIZK}}$) and given a public statement x, and a private witness w, $\mathtt{NIZK.Pr}$ produces a ZK-proof π. Given a proof π and a statement x, one can verify the proof π ($\mathtt{NIZK.Ver}(\mathsf{ppar}_{\mathtt{NIZK}}, x, \pi) \rightarrow \{0, 1\}$). NIZKs must guarantee soundness and zero-knowledge properties.

Definition 4 (Knowledge Soundness). *For all PPT adversaries $\mathcal{A}$ there exists an extractor $\mathsf{Ext}_{\mathcal{A}}$ and a negligible function $\mathsf{negl}()$ with $(\mathsf{ppar}_{\mathtt{NIZK}}) \leftarrow \mathtt{NIZK.Setup}(1^\lambda)$ and $((x, \pi^*); w) \leftarrow \mathcal{A}||\mathsf{Ext}_{\mathcal{A}}$ 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 5 (CRS indistinguishability). *We say that that a NIZK scheme has CRS indistinguishability if for all PPT adversary $\mathcal{A}$ there exist a simulator Sim such that :*

$$\Pr[\mathsf{crs} \leftarrow \mathtt{NIZK.Setup}(1^\lambda) : \mathcal{A}(\mathsf{crs}) = 1] \approx \Pr[(\mathsf{crs}, \tau) \leftarrow \mathsf{Sim}(1^\lambda) : \mathcal{A}(\mathsf{crs}) = 1]$$

Definition 6 (Simulation indistinguishability). *We say that that a NIZK scheme has simulation indistinguishability if for all PPT adversary $\mathcal{A}$ there exist a simulator Sim such that :*

$$\Pr[(\mathsf{crs}, \tau) \leftarrow \mathsf{Sim}(1^\lambda); (x, w) \leftarrow \mathcal{A}(\mathsf{crs}, \tau); \pi \leftarrow \mathtt{NIZK.Pr}(\mathsf{crs}, w, x) : \mathcal{A}(\pi) = 1 \text{ and } (x, w) \in \mathcal{R}]$$

$$\approx$$

$$\Pr[(\mathsf{crs}, \tau) \leftarrow \mathsf{Sim}(1^\lambda); (x, w) \leftarrow \mathcal{A}(\mathsf{crs}, \tau); \pi \leftarrow \mathsf{Sim}(\mathsf{crs}, w, x) : \mathcal{A}(\pi) = 1 \text{ and } (x, w) \in \mathcal{R}]$$

Definition 7 (Composable zero-knowledge). *A NIZK scheme is composable zero-knowledge if it has both CRS indistinguishability and simulation indistinguishability.*

Set Membership Proof A set membership proof allows one to prove knowledge of some value contained into a commitment, and moreover, that this value belongs to a specific set of possible values. More formally it is a proof of knowledge of the following statement: $\mathtt{NIZK.Pr}(\{(\mathsf{conf}, r) : \mathsf{c} \leftarrow \mathtt{COM.Commit}(\mathsf{conf}, r) \wedge \mathsf{conf} \in \mathsf{confset}\})$. Such proofs can additionally be zero-knowledge and thus reveal no information about the witness conf and r.

2 Technical Overview

Our protocol takes in input a forwarding-graph in which the nodes are virtualized components (VNFs and hypervisors) which may or may not be linked[4]. Note that the variety in node types sets our use-case apart from traditional IoT swarms, which are more homogeneous; however like the first IoT cRA protocols [5,1], our protocol D-cRA makes use of a spanning tree over the VNF-FG, over which requests will be broadcast and attestations will be aggregated. At the origin of the VNF-FG is a point of entry (also called a root node).

Including hypervisors in the VNF-FG model is crucial when the goal is to guarantee layer-linking: a strong property, which allows a verifier to not only check the integrity of the virtual

[4] In the ETSI definition of a VNF-FG, the term *node* corresponds to a VNF; however, in the rest of the paper we use that term to refer to either a VNF or a hypervisor. We assume that both VNFs and hypervisors are equipped with a Root of Trust (RoT), either physical or virtual.

components, but also to map VNFs to the hypervisors managing them. Unfortunately, including hypervisors is a non trivial task and can lead to much larger and complex forwarding-graphs. Indeed, in NFV structures, hypervisors may manage several VNFs – and this relationship needs translating into the graph's structure. This results in a strong structural guarantee of the VNF-FG, guaranteeing deep attestation for the entire infrastructure.

Keys, configurations, linking. Like in 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. Attestation can then be run online, potentially many times. While a trusted one-time setup might be deemed expensive for IoT swarms, which can change rapidly, the same is not true for NFV, in which infrastructures are more perennial and a single setup-phase will be useful for several attestation runs.

During setup, each node generates its cryptographic attestation credentials (sk, pk) (the secret key is kept into the RoT). We also explicitly consider the notion of node *configuration*, detailing the exact setup of the VNF or hypervisor, which the owner of the node might want to keep private from potentially semi-trusted neighbour nodes. Each node shares both the public attestation key and a *set* confset of potential configurations (rather than its exact configuration) with its neighbours. As opposed to traditional collective remote attestation (like [5,1]), we require VNF-FG cRA to guarantee *configuration-hiding*, a privacy property recently introduced in [4].

Finally, our scheme will require attestations to be linkable in the sense of [3]. As a result, during setup, each node generates and transmits its *linking information* link, which will enable verifiers to link two or more related attestations. A VNF's linking information is a single public key. A hypervisor's linking information is the set of public keys of the VNFs it manages. While layer-linking is irrelevant in an IoT context, it is crucial in network function virtualization.

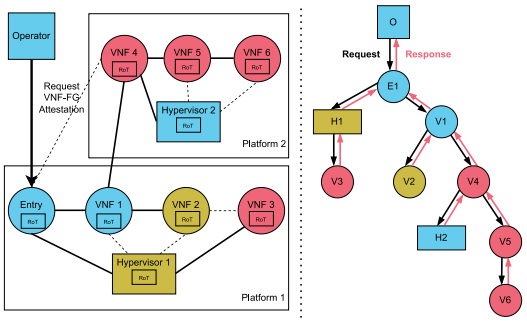

Fig. 3. A multi-tenant VNF-FG featuring hypervisors. Tenants owning various parts of the infrastructure are colour-coded. The bold links show an example of a spanning tree used to flood the request and aggregate the results. On the right a view of the resulting tree abstracting the underlying infrastructure.

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 underlying support for its services. The operator triggers the computation 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 roots, which is forwarded by each node to its children nodes. The response will be forwarded in the reverse order, from children to parent nodes and all the way back to the root. This is illustrated in Figure 3.

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 of Arfaoui *et al.* [3]. However, in our paper we also consider the *privacy* of node-configurations. Thus, in our protocol, when a node receives the request req (including an attestation nonce), the node's Root of Trust retrieves the node's configuration, then computes and sends an attestation (to the parent node), consisting of: (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 confset; (3) a signature σ over the commitment, the nonce (received in the request), and the node-linking information.

When a node receives an attestation, it first retrieves the nonce it has received in the latest request, and the set of configurations and linking information associated with the node which sent the attestation. Then it checks the proof π and signature σ under that linking information and nonce. Once a node has received and verified the attestations of all its children, it aggregates the verifications in a SEDA-like way [5]: if all the attestations were valid, the aggregated bit is 1, and it is 0 otherwise. Finally, the verifier node computes and sends a signature over the aggregate, the nonce, and 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 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 using set membership proofs.

3 Model

In this section, we formalize our scheme D-cRA and define the security and privacy properties that we want to achieve. Table 1 provides a summary of the used notations.

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, graphs containing loops, as well as those containing parallel links between nodes.

Offline Phase This phase consists of four algorithms :

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

$\underline{\mathtt{cRA.InitVerifier}(\mathsf{ppar}) \rightarrow (\mathsf{sk}_{\mathsf{ver}}, \mathsf{pk}_{\mathsf{ver}})}$: This algorithm initializes the verifier (the operator in our case) with a keypair consisting of the private key $\mathsf{sk}_{\mathsf{ver}}$ and corresponding 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 of the nodes of the 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 init algorithm 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, must imperatively include an attestation nonce $\mathsf{rand}_{\mathsf{req}}$.

Table 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
$\mathsf{confset}$	Node list of possible valid configuration
link	Node current linking information
stg	Storage of the node stored in the RoT
$\mathsf{pk}_{\mathsf{neib}}$	A neighbour public key
$\mathsf{confset}_{\mathsf{neib}}$	A neighbour set of configurations
$\mathsf{link}_{\mathsf{neib}}$	A neighbour 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

$\mathtt{cRA.Attest}(\mathsf{stg}, \mathsf{sk}, \mathsf{conf}, \mathsf{confset}, \mathsf{link}, \mathsf{req}) \rightarrow \mathsf{a} \cup \perp$: 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 verification of the validity of the request verifies, the request 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.

$\mathtt{cRA.Verify}(\mathsf{stg}, \mathsf{a}, \mathsf{req}) \rightarrow \{0,1\} \cup \perp$: 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[5] 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*

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

state (*i.e.*, the node has a 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.

$\underline{\texttt{cRA.Aggregate}(\mathsf{stg}, \mathsf{sk}, (\mathsf{a}_1, ..., \mathsf{a}_i), (\mathsf{agg}_1, ..., \mathsf{agg}_j), \mathsf{req}) \rightarrow \mathsf{agg}'}$: 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.

$\underline{\texttt{cRA.AggVerify}(\mathsf{stg}, \mathsf{agg}, \mathsf{req}) \rightarrow \{0,1\} \cup \perp}$: 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.

$\underline{\texttt{cRA.Link}(\mathsf{stg}_{\mathsf{ver}}, \mathsf{a}_{\mathsf{root}}, \mathsf{agg}_{\mathsf{root}}, \mathsf{req}) \rightarrow \{0,1\}}$: The linking algorithm is the final algorithm of the online phase, ran 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 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.

3.2 Properties

In this section we define the properties that collective remote attestation should guarantee. We use the seminal work of Arfaoui *et al.* [3], which first 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.

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 configuration in that node's (publicly known) configuration set. We give $\mathcal{A}$ access to the oracle $\mathsf{oChooseConf}_b$, presented below.

- $\underline{\mathsf{oChooseConf}_b(\mathsf{node}_i, \mathsf{conf}_0, \mathsf{conf}_1, \mathsf{confset}) \rightarrow (\mathsf{pk}_i, \mathsf{link}_i) \cup \perp}$: This oracle can only be called once. On input two configurations conf_0 and conf_1 as well as a configuration set confset and a bit b, this oracle checks that both configurations are in the set of valid configuration 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 confset and then simulates all the other steps of the 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.

Definition 8 (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:*

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

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

Fig. 4. The configuration-hiding game

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

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.

- $\mathsf{oCorrupt}(\mathsf{node}_i) \rightarrow \mathsf{OK}$: 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.
- $\mathsf{oReqGen}(\mathcal{G}) \rightarrow \mathsf{req}$: 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.
- $\mathsf{oAttest}(\mathsf{node}_i, \mathsf{req}) \rightarrow \mathsf{a}$: 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.
- $\mathsf{oAggreg}(\mathsf{node}_i, (\mathsf{a}_1, ..., \mathsf{a}_n), (\mathsf{agg}_1, ..., \mathsf{agg}_m), \mathsf{req}) \rightarrow \mathsf{agg} \cup \perp$: 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 $\perp$.

Definition 9 (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:*

$$\mathsf{Adv}^{\mathsf{UF}}_{\mathtt{cRA}}(\mathcal{A}) := \mathbb{P}[\mathcal{A} \text{ wins } \mathbb{G}_{\mathsf{UF}}(\lambda)] \leq \epsilon$$

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

Linking The pioneering effort by [3] defines 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. In this paper we are aiming for a similar property, adapted to the context of collective attestation.

In the context of 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

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

Fig. 5. The unforgeability game

verification of linking information through the attestation verification algorithm and provide an explicit result only at the end of the attestation process for the all VNF-FG.

As before, the adversary's goal will be 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 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 3.2. In addition, the adversary gains access to the following 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. Those modification includes moving a VNF to a different location within the graph or removing a VNF but also adding a VNF. The adversary have full control over added VNFs and specifies their information as it wants as input of the oracle. The oracle run the `cRA.InitStg` for those new VNF and only for them, all other VNF storage remain unchanged.

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 lists the graph $\mathcal{G}$ given in input to oReqGen and the resulting request req, which includes a nonce $\mathsf{rand}_{\mathsf{req}}$.

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

Fig. 6. 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 $\mathbb{G}_{\mathsf{LK}}(\lambda)$.

Definition 10 (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:*

$$\mathsf{Adv}^{\mathsf{LK}}_{\mathtt{cRA}}(\mathcal{A}) := \mathbb{P}[\mathcal{A} \textit{ wins } \mathbb{G}_{\mathsf{LK}}(\lambda)] \leq \epsilon$$

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

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 $\mathsf{NIZK} = (\mathtt{NIZK.Setup}, \mathtt{NIZK.Pr}, \mathtt{NIZK.Ver})$.

As in Section 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 set of configurations and linking information. Then they share those public information with their neighbours. This is done by running the `cRA.InitNode` and `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 $\mathsf{pk}_{\mathsf{ver}}$ (which is 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 (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 will be the public attestation (signature) key of that VNF (the same as the key generated by the `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 remain constant in the number of managed VNFs.

Remark 2. 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 3. 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 $\mathcal{T}_\mathcal{G}$[6]. 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 is ran 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 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. For that 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 $\mathsf{link}_{\mathsf{root}}$).

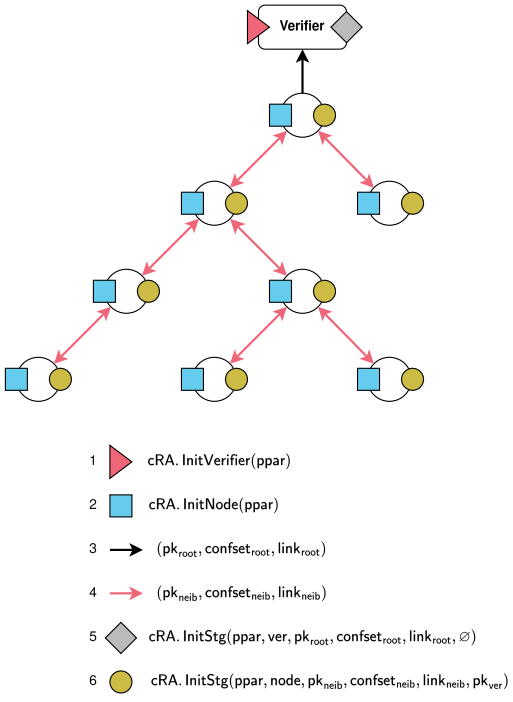

Fig. 7. The offline phase after setup.

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.

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

Each attestation is produced in response to an attestation request, produced by the verifier (*i.e.*, the operator) through the use of the `cRA.ReqGen` algorithm. The request contains an attestation nonce $\mathsf{rand}_{\mathsf{req}}$, 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 conf, run the `cRA.Attest` algorithm, and send that result to the parent-nodes. Upon receiving authenticated attestations from their children, parents verify them by using the `cRA.Verify` algorithm, then aggregate those values by using `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.

$\mathsf{a} = (\pi_{\mathsf{a}}, \sigma_{\mathsf{a}}, \mathsf{c})$: An attestation consists of: (1) A commitment c of the true configuration of the attested node[7]; (2) a proof π_{a} that the commitment is made with respect to a configuration value included in a set of valid configuration $\mathsf{confset}$ (the attestation itself does not contains 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.

$\mathsf{agg} = (\mathsf{res}_{\mathsf{agg}}, \sigma_{\mathsf{agg}})$: 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 8.

Remark 4. 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 children 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[8]. 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 the node is in an invalid state.

Remark 5. 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 `cRA.ReqGen`, `cRA.Attest`, `cRA.Verify`, `cRA.Aggregate`, `cRA.AggVerify` and `cRA.Link` algorithms :

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

[8] 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.

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

$\mathtt{cRA.Attest}(\mathsf{stg}, \mathsf{sk}, \mathsf{conf}, \mathsf{confset}, \mathsf{link}, \mathsf{req}) \rightarrow \mathsf{a} \cup \perp$: 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 $(\mathsf{rand_{req}}, \sigma_{\mathsf{req}})$, the node uses the verifier's public key value $\mathsf{pk_{ver}}$ (included in each node's storage) to verify the signature $\mathtt{SIG.Verify}(\mathsf{pk_{ver}}, \mathsf{rand_{req}}, \sigma_{\mathsf{req}})$. 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 $\mathsf{c} \leftarrow \mathtt{COM.Commit}(\mathsf{conf}, r)$ with some randomness r. Then it computes the proof $\pi_{\mathsf{a}} \leftarrow \mathtt{NIZK.Pr}(\{(\mathsf{conf}, r) : \mathsf{c} \leftarrow \mathtt{COM.Commit}(\mathsf{conf}, r) \wedge \mathsf{conf} \in \mathsf{confset}\})$ 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 $\sigma_{\mathsf{a}} \leftarrow \mathtt{SIG.Sign}(\mathsf{sk}, \mathsf{c}|\mathsf{link}|\mathsf{rand_{req}})$, 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.

$\mathtt{cRA.Verify}(\mathsf{stg}, \mathsf{a}, \mathsf{req}) \rightarrow \{0,1\} \cup \perp$: 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 $(\mathsf{rand_{req}}, \sigma_{\mathsf{req}})$. 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 $\mathsf{pk_{neib}}$ 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 $\mathsf{pk_{neib}}$ of the attested node, the expected configuration $\mathsf{confset_{neib}}$, and the expected linking-information $\mathsf{link_{neib}}$, and checks that $\mathtt{NIZK.Ver}(\pi_{\mathsf{a}}) = 1$ (*i.e.*, the node checks that the commitment c contains a configuration which is into the set $\mathsf{confset_{neib}}$). It also verifies that $\mathtt{SIG.Verify}(\mathsf{pk_{neib}}, \mathsf{c}\ |\mathsf{link_{neib}}|\mathsf{rand_{req}}, \sigma_{\mathsf{a}}) = 1$. 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 $(\mathsf{rand_{req}}, \sigma_{\mathsf{req}})$, 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 $\sigma'_{\mathsf{agg}} = \mathtt{SIG.Sign}(\mathsf{sk}, \mathsf{res}'_{\mathsf{agg}}|\mathsf{rand_{req}})$ 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 $(\mathsf{res_{agg}}, \sigma_{\mathsf{agg}})$ and the request req as $(\mathsf{rand_{req}}, \sigma_{\mathsf{req}})$. There are 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 $\mathsf{pk_{neib}}$ associated with the aggregate from its storage and verifies the signature over the result $\mathtt{SIG.Verify}(\mathsf{pk_{neib}}, \mathsf{res_{agg}}|\mathsf{rand_{req}}, \sigma_{\mathsf{agg}})$. If the signature is valid it outputs 1 otherwise it outputs 0.

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

Remark 6. 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.

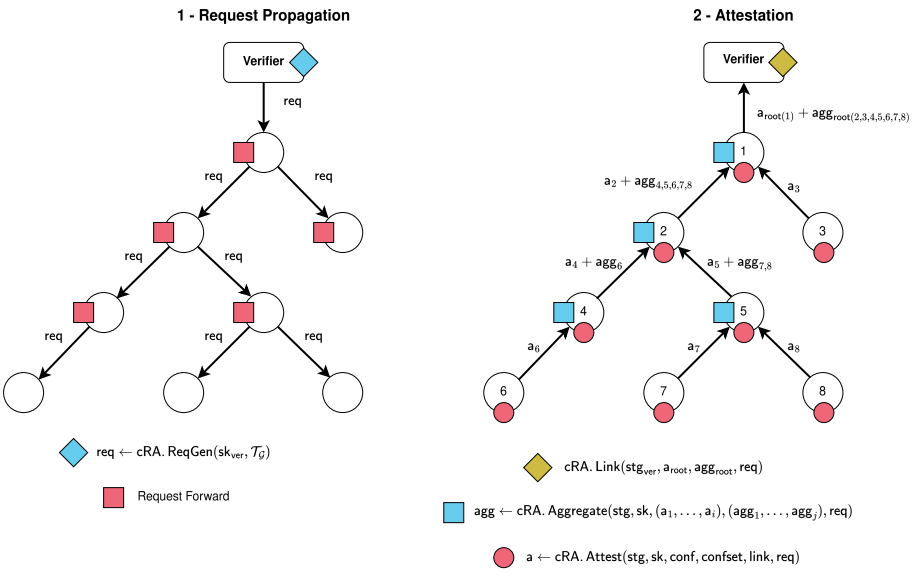

Fig. 8. The online phase.

5 Security Analysis

In this section we prove that the construction presented in Section 4 guarantees the properties defined in Section 3.

5.1 Configuration Hiding

We prove the following statement.

Theorem 1. *Let* cRA *be the privacy-preserving collective remote attestation scheme described in Section 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 [13], then D-cRA is configuration hiding. More specifically, assume that there exists an adversary $\mathcal{A}$ winning the configuration-hiding game ($\mathbb{G}_{\mathsf{CHide}}(\lambda)$) 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 $\mathsf{oAttest}$ 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 $\mathsf{oAttest}$ 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.

5.2 Unforgeability

We claim the following theorem.

Theorem 2. *Let* `cRA` *be the privacy-preserving collective remote attestation scheme described in Section 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 ($\mathbb{G}_{\mathsf{UF}}(\lambda)$) 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 this statement is true, recall that, in the unforgeability 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 $\mathbb{G}_{\mathsf{UF}}(\lambda)$ 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 $\mathbb{G}_{\mathsf{UF}}(\lambda)$.

$\mathbb{G}_1$**:** 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 $\mathbb{G}_{\mathsf{UF}}(\lambda)$ 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 verify, and they were 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.

5.3 Linking

We prove the following statement.

Theorem 3. *Let* $\mathtt{cRA}$ *be the privacy-preserving collective remote attestation scheme described in Section 4, 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-inter-active zero-knowledge proof of knowledge* NIZK *=* $(\mathtt{NIZK.Setup}, \mathtt{NIZK.Pr}, \mathtt{NIZK.Ver})$*. If the signature scheme is EUF-CMA, then D-cRA is linking. More specifically, assume that there exists an adversary* $\mathcal{A}$ *winning the linking game* $(\mathbb{G}_{\mathsf{LK}}(\lambda))$ *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* $\mathtt{SIG}$ *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:*

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

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 seems must 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. One of the most important 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 $\mathbb{G}_{\mathsf{LK}}(\lambda)$.

$\mathbb{G}_1$**:** 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 $\mathbb{G}_{\mathsf{LK}}(\lambda)$ 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.

6 Implementation

We implemented the total of 10 offline and online algorithms of D-cRA in C++ (Our implementation will be publicly available.) and simulated D-cRA over a group of nodes (up to 2.5×10^5 nodes) using OMNet++ [23]. 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 9).

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, efficient protocol due to Camenisch *et al.*[6] (rendered non-interactive via the Fiat-Shamir heuristic). The protocol is pairing based, and we used the BLS 12-381 curve implementation of mcl [22]. For the signature scheme, we used Libsodium [20] ed25519.

Memory cost We denote $|\mathsf{confset}|$ the number of element of a configuration set. After the offline phase of D-cRA, each node stores at least its signing key pair (sk,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[9] and its associated $\mathsf{confset}$ ($32+32+48\cdot|\mathsf{confset}|_i$ bytes for neighbour i). Thus the size in bytes of a configuration set then 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 the 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 maximum its own attestation and an 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.

Computational Load per Node In Table 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 [12] and set $|\mathsf{confset}|$ (the size of the set of configuration) to 100.

[9] 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.

Table 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

We compare our result with SEDA [5] implemented on the TrustLite [17] TEE.

Table 3. Time in ms for a single node to compute an attestation report depending on the number of children c.

	D-cRA	SEDA
Node	$1.64 + 2.437 \cdot (n-1)$	$0.6 + 4.4 \cdot (n-1)$
Leaf	$1.617 + 2.38 \cdot (n-1)$	-

VNF-FG Attestation Performance In order to evaluate the performance of our scheme D-cRA, we set $|\mathsf{confset}| = 100$, the communication time between two neighbour nodes to 1ms, and varied the number of children (for trees that range from 2-ary to 8-ary) and the tree height. Only for test purposes, we considered complete trees which enables to variate the number of involved nodes up to 2.5×10^5. For each combination, we ran multiple trials and plotted the mean value in Figure 9. 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.

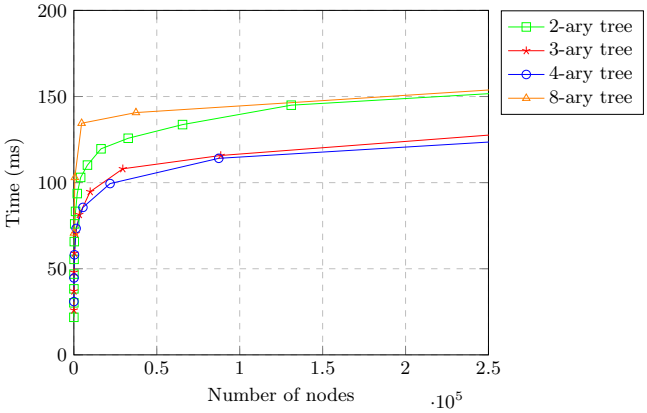

Fig. 9. VNF-FG attestation time

7 Conclusion

In this paper 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 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.

References

1. Ambrosin, M., Conti, M., Ibrahim, A., Neven, G., Sadeghi, A.R., Schunter, M.: SANA: Secure and Scalable Aggregate Network Attestation. In: CCS (2016)
2. Ambrosin, M., Conti, M., Lazzeretti, R., Rabbani, M.M., Ranise, S.: PADS: Practical Attestation for Highly Dynamic Swarm Topologies. In: SIoT (2018)
3. Arfaoui, G., Fouque, P.A., Jacques, T., Lafourcade, P., Nedelcu, A., Onete, C., Robert, L.: A Cryptographic View of Deep-Attestation, or How to Do Provably-Secure Layer-Linking. In: ACNS (2022)
4. Arfaoui, G., Jacques, T., Lacoste, M., Onete, C., Robert, L.: Towards a Privacy-Preserving Attestation for Virtualized Networks. In: Computer Security – ESORICS 2023 (2023)
5. Asokan, N., Brasser, F., Ibrahim, A., Sadeghi, A.R., Schunter, M., Tsudik, G., Wachsmann, C.: SEDA: Scalable Embedded Device Attestation. In: CCS (2015)
6. Camenisch, J., Chaabouni, R., shelat, a.: Efficient Protocols for Set Membership and Range Proofs. In: ASIACRYPT (2008)
7. Carpent, X., ElDefrawy, K., Rattanavipanon, N., Tsudik, G.: Lightweight Swarm Attestation: A Tale of Two LISA-s. In: ASIA CCS (2017)
8. Chen, L., Landfermann, R., Löhr, H., Rohe, M., Sadeghi, A.R., Stüble, C.: A Protocol for Property-Based Attestation. In: STC (2006)
9. Chen, L., Löhr, H., Manulis, M., Sadeghi, A.R.: Property-Based Attestation without a Trusted Third Party. In: ISC (2008)
10. Dushku, E., Rabbani, M.M., Conti, M., Mancini, L.V., Ranise, S.: SARA: Secure Asynchronous Remote Attestation for IoT Systems. IEEE Transactions on Information Forensics and Security (2020)
11. Fajiang, Y., Jing, C., Yang, X., Jiacheng, Z., Yangdi, Z.: An efficient anonymous remote attestation scheme for trusted computing based on improved CPK. Electronic Commerce Research (2019)
12. Google: Benchmark (2023), https://github.com/google/benchmark
13. Groth, J.: Simulation-sound NIZK proofs for a practical language and constant size group signatures. In: Advances in Cryptology - ASIACRYPT (2006)
14. IETF: IAB Statement on the risks of attestation of software and hardware on the open internet (2023), https://datatracker.ietf.org/doc/statement-iab-statement-on-the-risks-of-attestation-of-software-and-hardware-on-the-open-internet/
15. ISG-NFV: Network Functions Virtualisation (NFV); Trust; Report on Attestation Technologies and Practices for Secure Deployments. Tech. rep., ETSI (2017)
16. ISG-NFV: Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV. Tech. rep., ETSI (2021)
17. Koeberl, P., Schulz, S., Sadeghi, A.R., Varadharajan, V.: TrustLite: a security architecture for tiny embedded devices. In: Proceedings of the Ninth European Conference on Computer Systems (2014)
18. Kohnhäuser, F., Büscher, N., Katzenbeisser, S.: SALAD: Secure and Lightweight Attestation of Highly Dynamic and Disruptive Networks. In: ASIA CCS (2018)
19. Kohnhäuser, F., Büscher, N., Katzenbeisser, S.: A Practical Attestation Protocol for Autonomous Embedded Systems. In: EuroS&P (2019)
20. Libsodium: Libsodium (2023), https://doc.libsodium.org/
21. Mansouri, M., Jaballah, W.B., Önen, M., Rabbani, M.M., Conti, M.: FADIA: Fairness-Driven Collaborative Remote Attestation. In: WiSec (2021)
22. Mitsunari, S.: mcl (2023), https://github.com/herumi/mcl
23. OMNeT++: OMNeT++ (2023), https://omnetpp.org/
24. Petzi, L., Yahya, A.E.B., Dmitrienko, A., Tsudik, G., Prantl, T., Kounev, S.: SCRAPS: Scalable Collective Remote Attestation for Pub-Sub IoT Networks with Untrusted Proxy Verifier. In: USENIX Security (2022)
25. Rabbani, M.M., Vliegen, J., Winderickx, J., Conti, M., Mentens, N.: SHeLA: Scalable Heterogeneous Layered Attestation. IEEE Internet of Things Journal (2019)
26. Sadeghi, A.R., Stüble, C.: Property-Based Attestation for Computing Platforms: Caring about Properties, Not Mechanisms. In: NSPW (2004)