CoRA: A Scalable Collective Remote Attestation Protocol for Sensor Networks

Aïda Diop[1], Maryline Laurent[2], Jean Leneutre[3] and Jacques Traoré[4]
[1]*Télécom SudParis, Télécom ParisTech, Orange Labs, Caen, France*
[2]*SAMOVAR, CNRS, Télécom SudParis, Institut Polytechnique de Paris, France*
[3]*LTCI, Télécom ParisTech, Université Paris-Saclay, France*
[4]*Orange Labs, Caen, France*

Keywords: Security, Remote Attestation, Collective Attestation.

Abstract: Embedded Internet of Things (IoT) devices are deployed in the functioning of a number of applications such as industrial control, building automation, and the smart grid. The lack of robustness of IoT devices has however rendered such systems vulnerable to a number of remote cyber-attacks. Remote attestation is a security mechanism which enables to remotely verify the integrity of the software running on IoT devices. Similarly, *collective* remote attestation protocols are designed to efficiently verify the integrity of a *group* of devices. Existing collective attestation protocols do not provide an efficient and secure mechanism to detect compromised devices. In particular, it is not possible to efficiently trace the origin of an erroneous attestation response back to the concerned node. In this paper, we introduce CoRA, a highly scalable collective attestation protocol, which leverages the aggregating property of the underlying cryptographic scheme during the attestation process. CoRA is the first collective attestation protocol to also provide sequential detection, where the identity of the compromised node is revealed. We provide rigorous security proofs for our protocol and its underlying cryptographic primitive, and demonstrate its efficiency in highly scalable networks.

1 INTRODUCTION

Typically, connected Internet of Things (IoT) devices are deployed in data-sensitive and safety-critical applications, spanning a number of IoT systems. The safety of these systems is hindered by the lack of security and robustness of low-cost devices, resulting in a number of remote cyber-attacks (Mansfield-Devine, 2016; Falliere et al., 2010). Remote attestation is a security process which allows a trusted party, called verifier, to verify the integrity of the software running on each device. Traditionally, the attestation process runs between a single device (the prover), and the verifier. However, many IoT applications rely on self-organizing device mesh networks, also called swarms, to perform a given task. The concept of *collective (or swarm) attestation*, has been developped to provide scalable attestation protocols tailored for such networks (Ambrosin et al., 2016; Carpent et al., 2017; Ibrahim et al., 2016; Ibrahim et al., 2017; Kohnhäuser et al., 2017; Kohnhäuser et al., 2018). The general model of a collective attestation protocol, as illustrated in Figure 1, is comprised of the network operator, who deploys devices in the field, the group of devices performing a specific task, and the verifier. Nodes in the swarm generate their individual attestation responses, which are in turn accumulated into a single attestation response. The topology of the mesh network plays a crucial role in the effectiveness of the attestation protocol. Notably, during the attestation process, a spanning tree is generated over the network, allowing each node to propagate their response in the tree via their parent node. The root of the spanning tree is directly linked to the verifier, who collects a single aggregated report at the end of each execution (Chan et al., 2006). The successful verification of the final attestation response guarantees the integrity of the entire network. Initially, collective attestation protocols only mitigated against software attacks (Asokan et al., 2015). The majority of IoT applications we consider however require deploying devices in the field, thus rendering them vulnerable to physical attacks. Recent collective attestation propositions leverage the security provided by trusted com-

Diop, A., Laurent, M., Leneutre, J. and Traoré, J.
CoRA: A Scalable Collective Remote Attestation Protocol for Sensor Networks.
DOI: 10.5220/0008962700840095
In *Proceedings of the 6th International Conference on Information Systems Security and Privacy (ICISSP 2020)*, pages 84-95
ISBN: 978-989-758-399-5; ISSN: 2184-4356

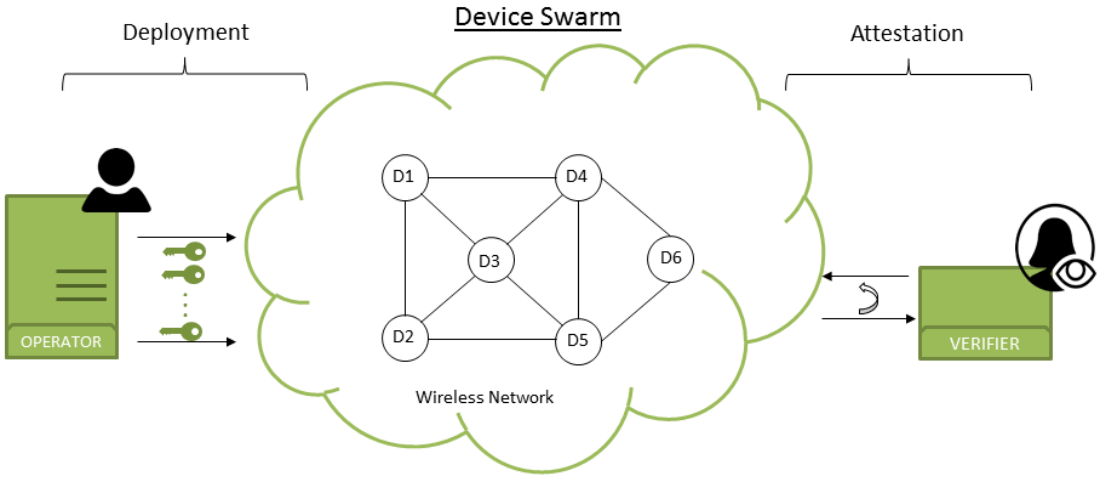


Figure 1: Overview of the collective attestation process.

puting by requiring each node to authenticate their attestation report with a cryptographic key (Ambrosin et al., 2016; Ibrahim et al., 2016; Kohnhäuser et al., 2017). The key is stored in a hardware module, hence, an adversary who attempts to forge an attestation is compelled to physically tamper with the devices.

Existing collective attestation protocols produce a binary response, stating whether or not the attestation process was successful, thus proving the integrity, or lack thereof, of the entire network. The nature of sensor networks, which consists of multihop networks of constrained and potentially remote devices, renders the attestation process vulnerable to injection attacks. In such attacks, the adversary is able to aggregate erroneous data in the final attestation response, without physically compromising a given node. Notably, he is able to remove the target node from the network, and aggregate a random value in lieu of the expected attestation response from said node. Upon receiving the aggregate response, the verifier has no way of tracing the origin of the erroneous attestation back to the concerned node. Indeed, existing aggregation schemes based on signatures or MACs, do not allow the extraction of an individual signature (or MAC) from the aggregated result. Such attacks may therefore trigger the verification process indefinitely, inducing a Denial of Service (DoS) attack on the verifier. A detection mechanism capable of tracing the origin and position in the network of similar attacks is therefore crucial to the overall security of the protocol. Indeed, the network administrator will thus be able to take further actions, which may consist in declaring the target device as compromised, and induce further ramifications which are out of scope for this paper.

Contributions. We present CoRA, a collective attestation protocol for wireless networks, composed of large numbers of low-cost and computationally constrained devices. CoRA is a secure and highly scalable attestation protocol, which efficiently verifies the integrity of a large number of devices in sensor networks. Indeed, the protocol is based on *aggregate algebraic MACs*, which guarantees the unforgeability of individual attestation responses, even in the presence of adversaries in the network tampering with the attestation process. The use of algebraic MACs, as opposed to signatures, in the aggregation process guarantees a more efficient verification step. In addition, we leverage the properties of algebraic MACs, notably the secret key embedded in each device, to provide a sequential detection mechanism. CoRA thus enforces device accountability, and enables tracing the origin of compromised nodes that attempt to inject erroneous data during the attestation process. We provide rigorous security proofs for CoRA and its underlying cryptographic construction, namely aggregate algebraic MACs. We then prove the efficiency of the protocol, and its application in large wireless networks of heterogeneous devices.

Outline. We present in Section 2 the related work on collective attestation, and their limitations. Section 3 illustrates the system model, device requirements, and threat model. In Section 4, we present the underlying cryptographic scheme, used for secure and scalable attestation generation and aggregation. The CoRA protocol is introduced in Section 5, including the attestation and detection processes. We evaluate the efficiency of CoRA in Section 6, and prove the security of the protocol in Section 7. Section 8 concludes the paper.

2 RELATED WORK

In this section, we present the system and threat model of a collective attestation protocol, and analyse existing solutions and their shortcomings.

2.1 Collective Attestation

Remote attestation is a security mechanism which enables a trusted third party, namely the verifier, to remotely check the integrity of a device called the prover. The process is initiated by the verifier, who issues a challenge to the prover. The prover in turn generates an attestation report on its current software state, which the verifier is able to correctly verify. *Collective attestation protocols* have been proposed to efficiently attest large networks of interconnected devices. In this scenario, the verifier issues the challenge to a network of devices via an entry node A_s, which is chosen based on its geographic proximity to the verifier. The entry node propagates the challenge to its children in a process that results in nodes being classified in a spanning tree rooted at A_s. This topology allows the efficient aggregation of attestation reports, and reduces communication overhead. The concept of collective attestation was introduced in SEDA (Asokan et al., 2015). Subsequently, a number of solutions targeting the security and efficiency shortcomings of SEDA were proposed. SANA (Ambrosin et al., 2016) leverages the structure of aggregate and multi-signature schemes, in order to obtain a highly scalable attestation protocol. The solution provides public verifiability, and limits physical attacks by authenticating reports within secure hardware.Their solution however offers scalability at the cost of efficiency, since the verification step requires a number of costly pairing computations, that are in addition linear in the number of devices in the group. DoS attacks are mitigated in SeED (Ibrahim et al., 2017) through the use of a secure clock, which enables provers to periodically forward their attestation reports in a non-interactive protocol. LISA$_\alpha$ and LISA$_s$ (Carpent et al., 2017) provide more practial alternatives to SEDA by proposing a qualitative classification of attestation protocols, each tailored to specific security needs. DARPA (Ibrahim et al., 2016) and SCAPI (Kohnhäuser et al., 2017) provide mitigation against physical attacks, while SALAD (Kohnhäuser et al., 2018) is a colletive attestation protocol suitable for dynamic and disruptive networks.

However, no solution provides the individual detection of nodes responsible for a false attestation response in the network. Indeed, the nature of wireless communication in mesh networks facilitates the injection of a false attestation report (i.e. which has not been authenticated by a valid key). In existing attestation protocols, the verification step of the resulting attestation report fails when such falsifications are performed, however no further information on the origin of the injection is provided. Indeed, the underlying model of secure in-network aggregation schemes (Chan et al., 2006) does not allow report manipulation detection. The result only stating retrospectively that the aggregate message was manipulated. Existing collective attestation protocols relying on aggregate cryptographic authentication schemes, namely aggregate MACs (Katz and Lindell, 2008; Eikemeier et al., 2010) and aggregate signatures (Boneh et al., 2003; Lu et al., 2006; Boldyreva et al., 2007), do not allow a verifier to check the origin of an erroneous report which has been aggregated into the final response. Furthermore, the verification step in aggregate signature schemes requires a number of pairing computations linear in the number of nodes, which is prohibitively expensive in sensor networks.

2.2 Efficient *in-network aggregation*

Large networks of embedded devices require highly scalable data collection solutions in order to forward individually collected data to a central base station. In sensor networks, the base station in the attestation process is the verifier. Aggregation methods allow a collection of sensors to securely and collaboratively compute an aggregation function on their respective reports. The result of said function is thus forwarded to the verifier, generating extremely low communication overhead in the process. This concept, known as *in-network aggregation* (Chan et al., 2006), considerably reduces communication overhead, and provides a highly scalable data collection mechanism. Existing aggregation methods that are linear in the number of nodes, and that can be executed in a single round (Ambrosin et al., 2016), are based on aggregate digital signatures (Boneh et al., 2003). However as discussed in Section 2.1, such constructions are extremely costly due to the use of pairing-based cryptography.

In sensor networks, and more generally distributed networks of embedded devices, it is mostly the case that the network administrator plays the role of both the operator and the verifier (or both are operated by the same entity, i.e. the owner). For example, in the Smart grid, electrical companies issue smart meters with a valid key, and periodically verify their proper operation. Thus the efficiency issues in aggregate signatures, can be avoided by the use of algebraic MAC schemes.

3 SYSTEM MODEL AND ASSUMPTIONS

3.1 Background

Collective Attestation Architecture. In the architecture presented in Figure 1, a collective attestation protocol is divided into two main phases, namely the deployment phase and the attestation phase. During the former, the operator O deploys devices in the field with a unique identifier ID_i for device D_i, and a secret key sk_i. The verifier $\mathcal{V}$ initiates the attestation phase by generating a challenge Ch, as part of the attestation request for a device A_s in the network. The attestation process will create a spanning tree rooted at A_s, thus enabling an efficient and scalable aggregation of the individual attestation reports. A_s then propagates the request down the spanning tree. Upon each node generating, authenticating, and aggregating their attestation reports, the final aggregated response reaches A_s, who aggregates its own response, before forwarding the result to $\mathcal{V}$. The verifier, who is in possession of the expected internal state of each device, is thus able to verify the integrity of the entire network. For simplicity, we assume in the remaining of the paper that $\mathcal{V}$ and O are either the same entity, or are managed by the same entity.

Definition and Security Model. We use the formal remote attestation protocol definition and security model of Francillon et al. (Francillon et al., 2012). Let t_{exp} be the upper-bound on the execution time of the attestation procedure. The value t_{exp} is defined as a function of the total number of nodes in the network, the time complexity of an individual attestation computation, intermediary transmission time, and the aggregation time. Francillon et al. (Francillon et al., 2014) define the security of a remote attestation protocol, to be resistance against forgery. The security notion of resistance against forgery considers the case where an adversary has oracle access to the attestation generation function. Considering such adversary $\mathcal{A}$, that is thus able to request $n-1$ attestations, $\mathcal{A}$ should still be unable to produce a new valid attestation on a message he has not previously queried to the attestation oracle.

3.2 Overview

We consider a network of low-end, heterogeneous, embedded devices, communicating over a wireless mesh network. Devices in the network have limited computational power and storage capacity. In the infrastructure of the collective attestation protocol, we assume the operator O to be the network administrator, who initially deploys devices in the field. The deployment phase is only executed once, and consists in each device D_i being initialized with a secret symmetric key sk_i shared with O (and the verifier $\mathcal{V}$), as well as a unique identifier ID_i, which can be the IP address of wireless devices for example. In order to verify the integrity of the internal software state of every device in the network, the trusted verifier $\mathcal{V}$ periodically engages in the attestation protocol with the device swarm. At the beginning of the attestation phase, a spanning tree of all nodes is formed, thus facilitating the aggregation process. At the end of the attestation phase, $\mathcal{V}$ collects a single attestation report, which guarantees the integrity (or lack thereof) of the entire swarm. The verifier possesses the list of expected software state, hence allowing him to verify the validity of the final attestation response in correlation with the expected values. Each device is equipped with the minimal hardware requirements for a collective attestation protocol, namely a Read-Only Memory (ROM) to store the protocol code and cryptographic keys, as well as a Memory Protection Unit (MPU). The MPU controls access to the ROM, and ensure that only protocol code can access the secret keys. Each device D_i can compute a collision-resistant hash function, as well as multiplications in elliptic curve groups. Furthermore, we assume that initialized counters are stored in the ROM and are only accessible by the MPU, thus preventing any desynchronization attempt.

3.3 Threat Model

Adversary Model. We consider an adversary $\mathcal{A}$, who has full access to the communication channel (Dolev-Yao model). As such, $\mathcal{A}$ can eavesdrop, modify, insert, and drop messages exchanged between devices. $\mathcal{A}$ can also capture nodes and access and modify their software state. Security against physical adversaries (who are able to extract cryptographic keys), cannot however be guaranteed (Ibrahim et al., 2016). In contrast, we consider the following threat model: an adversary $\mathcal{A}$ who has compromised the networks, and potentially captured a number of devices (up to $n-1$ devices in a network comprising n devices), should still be unable to forge a valid aggregate attestation for the remaining nodes. With regards to Denial of Service (DoS) attacks, existing attestation protocols consider attacks where devices are rendered unavailable in the network, thus preventing the successful completion of the process. Such attacks cannot be prevented, as it is indeed impossible to guarantee availability for an adversary $\mathcal{A}$ capable of dropping all messages. In this paper, we also consider DoS attacks

against the verifier. Indeed, $\mathcal{A}$ is able to continuously aggregate an erroneous message in the final response, triggering a failed verification process each time. We mitigate against such attacks by providing an efficient detection mechanism in CoRA.

Security Objectives. As defined in Section 3.1, an attestation protocol is secure if no adversary is able to forge an attestation report, thus faking a healthy state for a compromised device. In Section 7, we prove the security of CoRA against such adversaries, based on the security of the underlying cryptographic scheme.

4 $\mathsf{MAC}_{\mathsf{BLS}}$ AGGREGATION SCHEME

We present in this section a hybrid cryptographic scheme introduced by Dodis et.al. (Dodis et al., 2012), namely algebraic MACs. Algebraic MACs provide the best of both worlds, i.e. the algebraic properties inherent to signature schemes, while avoiding the costly public verification step. This is possible considering the fact that in sensor networks, the same entity manages both the operator and the verifier. We later provide an aggregate algebraic MAC scheme, which is subsequently used as the underlying scheme for CoRA to authenticate individual attestation reports, and perform detection.

4.1 Notations

We introduce in this section the mathematical and cryptographic notations used in the remaining of the paper.
We define the sampling of a random element x from a set X using the following notation: $x \xleftarrow{\$} X$. $Pr[A]$ denotes the probability of an event A. The notation $\mathbb{G}$ denotes an algebraic group, while $g \in \mathbb{G}$ represents an element g in group $\mathbb{G}$. $M = \{m_1, ..., m_n\}$ denotes a set M, comprised of n elements $m_1, ..., m_n$.

4.2 Algebraic MACs

Algebraic MACs are MAC schemes based on group operations, as opposed to block ciphers or hash functions. Their primary goal is to take advantage of the symmetry between the signer and the verifier in applications where the signer and verifier are the same entity, in order to provide a more efficient verification step. Algebraic MACs are used in CoRA to first generate unforgeable attestations, while also allowing the construction of an efficient and secure detection mechanism based on each device's knowledge of their secret keys.

4.3 Aggregate Algebraic MACs

The underlying scheme for CoRA is an aggregate algebraic MAC, named aggregate $\mathsf{MAC}_{\mathsf{BLS}}$, and derived from the Boneh, Lynn, Shacham (BLS) signature scheme (Boneh et al., 2001). Aggregate MAC schemes, introduced by Katz and Lindell (Katz and Lindell, 2008), allow a set of n users to generate n tags on n potentially different messages, and aggregate the result into a single tag of the same size as an individual tag. The verification process, which is linear in the number of users, attests of the validity of all tags in the final aggregate. We define an aggregate algebraic MAC, where tags are group elements as opposed to block ciphers or hash functions. Tags are single group elements of size 256-bit (for example elliptic curve group elements). The output size of secure hash functions being comparable to such output sizes, the use of algebraic MACs does not induce additional space complexity.
The security of the aggregate $\mathsf{MAC}_{\mathsf{BLS}}$ scheme is based on a standard discrete logarithm-based security assumption, namely the Computational Diffie-Hellman (CDH) assumption, defined as follows:

Computational Diffie-Hellman Problem (CDH). Given $g, g^a, g^b \in \mathbb{G}$ for $a, b \in Z_q$, compute $g^{ab} \in \mathbb{G}$.

The CDH assumption stipulates that any algorithm $\mathcal{A}$ that runs in polynomial time has negligible advantage in solving the CDH problem in $\mathbb{G}$.

$\mathsf{MAC}_{\mathsf{BLS}}$ Construction. We first give the construction of an individual algebraic MAC scheme $\mathsf{MAC}_{\mathsf{BLS}}$. The $\mathsf{MAC}_{\mathsf{BLS}}$ scheme comprises the following algorithms:

Setup(1^λ)**.** Create the public parameters $params = (\mathbb{G}, q, g, \mathcal{H})$ where $\mathbb{G}$ is a group of prime order q of size λ, where CDH is hard. $g \xleftarrow{\$} \mathbb{G}$ is a random generator of $\mathbb{G}$. $\mathcal{H}_1 : \{0,1\}^* \rightarrow \mathbb{G}$ is a hash function, modeled as a random oracle in the security analysis.

KeyGen(1^λ)**.** Select a random element $x_i \xleftarrow{\$} \mathbb{Z}_q$, and sets the secret key $sk_i = x_i$. The algorithm also outputs the public parameter $iparams = X_i$, where $X_i = g^{x_i}$.

Mac$(params, sk_i, m_i)$**.** Compute $h_i \leftarrow \mathcal{H}_1(m_i)$, and compute the tag $\tau_i = h_i^{x_i}$.

Verify$(params, sk_i, pk_i, m_i, \tau_i)$**.** Compute $h_i = \mathcal{H}_1(m_i)$, and accept if $\tau_i = h_i^{x_i}$.

Theorem 1 *A probabilistic polynomial-time adversary against MAC_{BLS} succeeds in returning a valid forgery with only negligible probability under the CDH assumption.*

The proof for Theorem 1 is provided in appendix A.1.

Aggregate $\mathsf{MAC}_{\mathsf{BLS}}$ Construction. Based on the $\mathsf{MAC}_{\mathsf{BLS}}$ scheme construction introduced above, we define the aggregate MAC scheme in groups of prime order. The aggregation function AggMac is public and unkeyed, namely any node is able to aggregate its own tag without a secret key (as opposed to the MAC computation step). The aggregate $\mathsf{MAC}_{\mathsf{BLS}}$ scheme is the $\mathsf{MAC}_{\mathsf{BLS}}$ scheme with the following additional algorithms:

AggMac$(\{\tau_i\}_{i=1}^n)$**.** Given a set of tags $\{\tau_i\}_{i=1}^n$, output the aggregate tag $\tau = \prod_{i=1}^n \tau_i$.

AggVerify$(params, \{(sk_i, pk_i)\}_{i=1}^n, \{m_i\}_{i=1}^n, \tau)$**.** For $1 \leq i \leq n$, compute $h_i = \mathcal{H}_1(m_i)$. Check if $\tau = \prod_{i=1}^n h_i^{x_i}$. Return accept if true, and reject otherwise.

Theorem 2 *Aggregate MAC_{BLS} is unforgeable provided that the underlying MAC_{BLS} scheme is unforgeable.*

The security definition of unforgeability for aggregate algebraic MACs is that of existential unforgeability under chosen message attack. This definition illustrates the fact that an adversary who has access to at most $n-1$ secret keys, is still unable to output a valid aggregate tag by n users (Boneh et al., 2003). We prove Theorem 2 in appendix A.2.

5 CoRA

CoRA is a secure and scalable collective attestation protocol, which consists of two phases, namely the *deployment phase* presented in Section 5.2 and the *attestation phase* in Section 5.3. The protocol also comprises an optional *detection phase*, presented in Section 5.4. The *deployment phase* is executed only once by the operator O, who initializes the network. Following deployment, the verifier $\mathcal{V}$ can periodically verify the software integrity of all devices in the network by initiating the *attestation phase*. In the case where verification fails, $\mathcal{V}$ initiates the *detection phase*, in order to identify the origin of the failure. We demonstrate the security of CoRA in Section 7, based on the security of its underlying aggregation mechanism.

5.1 Preliminaries

In the construction of CoRA, we use anonymous authentication schemes, namely non-interactive zero-knowledge proofs of knowledge (also known as signatures of knowledge (Schnorr, 1991)). Signatures of knowledge are highly efficient signature schemes, that leverage the knowledge of a cryptographic secret, in order to authenticate a given user. They are the building block of numerous authentication schemes such as group signatures or anonymous credentials. In CoRA, signatures of knowledge are used in the *attestation* and *detection phases* to prove that a given node has generated a given tag during the aggregation process. To define our signatures of knowledge, we use the Camenisch and Stadler notation (Camenisch and Stadler, 1997), whereby $\pi(m) = SoK\{sk_i : y = g^{sk_i}\}(m)$ denotes a signature of knowledge on message m. A device D_i with secret key sk_i signs message m by proving that he knows the secret sk_i, corresponding to a discrete logarithm in base g.

Theorem 3 *CoRA is secure against forgery attacks, under the assumption that aggregate MAC_{BLS} is unforgeable.*

The proof for Theorem 3 is provided in Section 7.

5.2 Deployment Phase

In the deployment phase, the network operaror O runs the Setup and KeyGen of the aggregate $\mathsf{MAC}_{\mathsf{BLS}}$ scheme, and obtains the public parameters *params*, and the secret keys sk_i for every device D_i. Prior to deployment, O initializes each device with a $\mathsf{MAC}_{\mathsf{BLS}}$ secret key sk_i for $i \in \{1,...,n\}$, and its own $\mathsf{MAC}_{\mathsf{BLS}}$ public value $pk_0 = X_0$. The secret keys are used by devices to generate $\mathsf{MAC}_{\mathsf{BLS}}$ tags τ_i on their internal state S_i. The tags will later be aggregated and forwarded to $\mathcal{V}$. The attestation and aggregation steps are developped in Section 5.3. In addition, each D_i is initialized with a counter value cnt_k, which corresponds to the attestation sequence k. cnt_k is initialized to 0, and incremented by the verifier $\mathcal{V}$ and each node after each attestation process. The counters are used to monitor the attestation sequence number, thus preventing replay attacks. Indeed, upon receiving the attestation request, node D_i verifies that the value of the associated counter is greater than or equal to the

Table 1: CoRA parameters notation.

Acronyms	Description
O	Network operator
$\mathcal{V}$	Network verifier
sk_O, pk_O	O's $\mathsf{MAC_{BLS}}$ key pair
D_i	Device i with identifier ID_i
A_s	root device
sk_i, pk_i	D_i's $\mathsf{MAC_{BLS}}$ key pair
S_i	attestation report
τ_i	MAC tag on message s_i
$attReq$	attestation request
GenAtt(key, s_i)	generates and authenticates attestation report
AggAtt($\tau_1, ..., \tau_m$)	aggregates attestation reports
VrfAggAtt($\{s_i\}_{i=1}^m, \tau$)	verifies aggregated report

counter value stored locally. Each device in the network possesses a unique identifier ID_i, which can for example, be derived from its IP address. O is able to deploy new devices in the swarm at any time, by executing the initialization process described above.

5.3 Attestation Phase

The attestation phase allows the network operator O and the verifier $\mathcal{V}$, to verify the integrity of every device in the swarm.

Overview. The goal of the attestation phase is to iteratively authenticate each device D_i's internal software state S_i, and propagate said attestation proofs up the attestation tree until it reaches the verifier $\mathcal{V}$. The attestation phase of CoRA (Figure 2), is divided in three sub-phases, namely request dissemination, attestation, and verification. $\mathcal{V}$ initiates the request dissemination phase by sending an attestation request $attReq$ to the closest device in his communication range A_s. Upon receiving $attReq$, A_s verifies that it is indeed a valid request from $\mathcal{V}$, and forwards to its children down the attestation tree. Each device in the tree receiving the request, verifies its authenticity and freshness. This initial step securely communicates to each device that a new attestation process has been initiated. Upon authenticating the request, leaf nodes generate the attestation report on their internal software state and send it to their parents. Intermediary nodes authenticate said reports, generate their own attestation, and in turn send the aggregated result to their parents. $\mathcal{V}$ receives the final report from A_s. He then verifies the validity of the aggregate report, hence confirming the integrity (or lack thereof) of every device in the network.

(1) Request Dissemination. $\mathcal{V}$ initiates the attestation phase by disseminating an attestation request to the nearest device in its communication range A_s. The request $attReq$ is generated as follows:

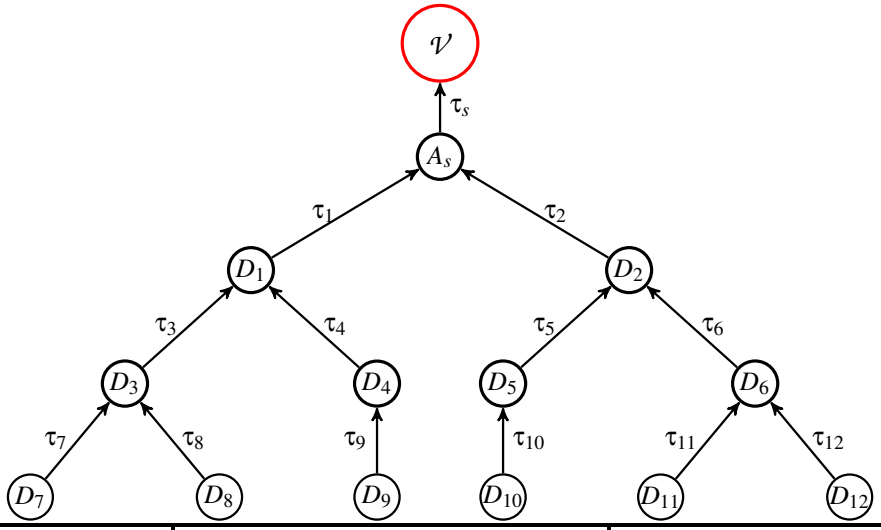
- Generate a random nonce $N_k \xleftarrow{\$} \{0,1\}^{l_N}$ for the current attestation process, where l_N is the size of the nonce.
- Update $cnt_k \leftarrow cnt_k + 1$
- Choose a random collision-resistant hash function H
- Generate a signature of knowledge on message $m = N_k$ as $\pi_0 = SoK\{x_0 : X_0 = g^{x_0}\}(m)$.
- Define challenge $c_0 \leftarrow \{n, X_0, Cert_{X_0}, \pi_0\}$, where X_0 is the $\mathsf{MAC_{BLS}}$ public key for O and $Cert_{X_0}$ a valid certificate on X_0. n is the size of the swarm.
- $attReq \leftarrow c_0$

Upon receiving $attReq$, A_s authenticates $\mathcal{V}$ by verifying π_0. It then propagates the request down the attestation tree. Intermediary nodes authenticate the request before forwarding it to their children. This authentication step mitigates against distributed DoS attacks, whereby an attacker might render devices unavailable by sending them false attestation requests.

(2) Attestation. Upon checking the authenticity and freshness of $attReq$, each leaf node D_i proceeds to measuring its internal software software state S_i. It then runs the attestation generation function **GenAtt**(sk_i, S_i) defined as follows:

GenAtt: (sk_i, S_i).

1: Generate $\tau_i \leftarrow \mathsf{MAC_{BLS}}(sk_i, m_i)$, where $m_i = (N_k || cnt_k || S_i)$. Including N_k and cnt_k in the message ensures the freshness of the tag.
2: Generate a signature of knowledge π_i, of the secret key on the tag τ_i, where $\pi_i = SoK\{x_i : \tau_i = h_i^{x_i} \wedge X_i = g^{x_i}\}(\tau_i)$.
3: Define an empty list $\mathcal{C}_i = \emptyset$
4: Return report $a_i \leftarrow (ID_i, \mathcal{H}(N_k || cnt_k), \tau_i, \pi_i, \mathcal{C}_i)$



Level 1	Level 2	Level 3 & 4
• $[\tau_7] = \mathsf{Mac}_{sk_7}(s_7)$	• $[\tau_3] = \mathsf{AggMac}(\{\tau_7, \tau_8, \tau_3\})$	• $[\tau_1] = \mathsf{AggMac}(\{\tau_3, \tau_4, \tau_1\})$
• $[\tau_8] = \mathsf{Mac}_{sk_8}(s_8)$	• $[\tau_4] = \mathsf{AggMac}(\{\tau_9, \tau_4, \})$	• $[\tau_2] = \mathsf{AggMac}(\{\tau_5, \tau_6, \tau_2\})$
⋮	⋮	
• $[\tau_{12}] = \mathsf{Mac}_{sk_{12}}(s_{12})$	• $[\tau_6] = \mathsf{AggMac}(\{\tau_{11}, \tau_{12}, \tau_6\})$	• $[\tau_s] = \mathsf{AggMac}(\{\tau_1, \tau_2, \tau_s\})$

Figure 2: Aggregation tree containing nodes $\{A_s, D_1, ..., D_{12}\}$.

Upon receiving the attestation reports a_i and a_j from its children, intermediary node D_l runs the attestation aggregation function **AggAtt**(τ_i, τ_j):

AggAtt: (τ_i, τ_j).

1: Verify π_i and π_j. If the verification succeeds, proceed to step 2. Else, set $\mathcal{C}_l = \{ID_k\}_{k \in \{i,j\}}$, where the identifier(s) in $\mathcal{C}_l$ corresponds to those whose signature of knowledge verification failed.
2: Generate $\tau_l \leftarrow \mathsf{MAC}_{\mathsf{BLS}}(sk_l, m_l)$, where message $m_l = (N_k || cnt_k || S_l)$
3: Compute $\tau \leftarrow \mathsf{AggMAC}_{BLS}(\tau_i, \tau_j, \tau_l)$
4: Generate $\pi_l = SoK\{x_l : \tau_l = h_l^{x_l} \wedge X_l = g^{x_l}\}(\tau)$
5: Return $a_l \leftarrow (ID_l, \mathcal{H}(N_k || cnt_k), \tau, \pi_l, \mathcal{C}_l)$

At the end of the attestation generation, aggregation, and propagation phase, $\mathcal{V}$ receives the final attestation report a_s from A_s, and proceeds to verifying its validity.

(3) Verification. Upon receiving the final report in time $t \leq t_{exp}$, $\mathcal{V}$ runs the aggregated attestation verification function **VrfAggAtt**$(sk_1, ..., sk_n, \tau_1, ..., \tau_n, \{S_1, ..., S_n\})$ on the list of expected healthy states $\{S_1, ..., S_n\}$ it possesses. The time t_{exp} is defined through prior real-world experiments, in order to estimate an upper-bound of the overall attestation time. Such experiments must include network delays, transmission times, and individual attestation generation and aggregation times.

VrfAggAtt: $(\{sk_i\}_{i=1}^n, \{\tau_i\}_{i=1}^n, \{S_i\}_{i=1}^n)$.

1: $b \leftarrow \mathsf{AggVerifyMAC}_{BLS}(\{sk_i\}_{i=1}^n, \tau, \{S_i\}_{i=1}^n)$
2: If $b = 1$ **and** $\mathcal{C}_s = \emptyset$, return 1
3: Otherwise return 0

If the function returns 1, $\mathcal{V}$ concludes that the network is in a trustworthy state. Otherwise, he concludes that at least one device is compromised and proceeds to the detection phase.

5.4 Detection Phase

An attacker $\mathcal{A}$ who performed a mismatch attack, by aggregating an erroneous attestation report, will be identified in the detection phase. Let D_c be the target device in such attacks. Upon the function **VrfAggAtt** returning 0, $\mathcal{V}$ proceeds as follows:

1. $\mathcal{V}$ requests the signatures of knowledge $\{\pi_i\}_{i=1}^n$ for every node in the network.

2. $\mathcal{V}$ verifies the corresponding tags $\tau_{i\,i=1}^{n}$.
3. $\mathcal{V}$ proceeds iteratively down the attestation tree until it reaches the node D_i which did not aggregate the valid tag/SoK pair.

The signature of knowledge ensures that each node remains accountable for the value it has previously aggregated during the attestation phase. Considering the assumption that no adversary can easily perform physical attacks, and generate a valid $\mathsf{MAC}_{\mathsf{BLS}}$ tag without a valid secret key sk_i, this ensures that $\mathcal{V}$ is able to identify each node that aggregates an erroneous report.

6 PERFORMANCE EVALUATION

We implemented CoRA on a Teensy 3.2 microcontroller. The Teensy 3.2 board is an Arduino-compatible microcontroller, featuring a 32 bit ARM processor with a 72 MHz Cortex-M4 core. It also features 256 kB Flash and 64 kB RAM memory. The board provides the minimal hardware requirements for remote attestation as stated in Section 3.2. Cryptographic algorithms are implemented based on the micro-ecc(MacKay, 2017) library. As presented in Table 2, we use a SHA-256 hash function.

Table 2: Performance of cryptographic algorithms on Teensy 3.2.

Algorithm	Run-time (ms)
SHA-256	0.244
SHA-256 HMAC	0.809

Table 3: Performance of $\mathsf{MAC}_{\mathsf{BLS}}$ functions.

Mac Run-time (s)	Number of devices	AggMac Run-time (s)
0.047	100	5.55
	500	27.8
	1000	56.1
	1500	84.3

Table 3 shows the runtime performance of $\mathsf{MAC}_{\mathsf{BLS}}$. The choice of algebraic MACs (in elliptic curve groups) provides a more efficient verification step, considering that computing **AggVer** is the same as computing **AggMac**. The implementation makes use of secp160r1 curves (Research, 2000).

7 SECURITY ANALYSIS

In this section, we provide a proof of security of CoRA, as stated by Theorem 3. As specified in Section 3.1, a swarm attestation scheme is secure against forgery if no adversary is able to produce a forgery for an uncompromised device (for which he doesn't know the secret key), even after accessing at most $n-1$ attestation results. We derive the security of our swarm attestation protocol by drawing the parallel between the security definition for swarm attestation forgery (Francillon et al., 2014), and the security of the aggregate $\mathsf{MAC}_{\mathsf{BLS}}$ scheme.
In formal swarm attestation definition, the verifier $\mathcal{V}$ returns 1 if $\mathsf{AggVerify}(params, \{sk_i\}_{i=1}^{n}, \{m_i\}_{i=1}^{n}, \tau) = 1$, i.e. if τ is a valid aggregate tag on the set of expected valid states $\{S_i\}_{i=1}^{n}$, and the incorporated nonce N_k is the session nonce provided by $\mathcal{V}$. Let $\mathcal{A}$ be an attacker on the network whose goal is to produce a valid forgery of the attestation response. $\mathcal{A}$ has compromised prover D_c. We consider two types of adversaries:

Type 1: $\mathcal{A}_1$ does not alter the attestation of D_c to be included in the aggregate response.
Type 2: $\mathcal{A}_2$ modifies the internal state of D_c, and generates an attestation τ_c on $(ID_c, S_c||N_k||cnt_k)$.

In the first case, according to our assumptions, $\mathcal{A}_1$ cannot physically compromise D_c and retrieve its secret key x_c. $\mathcal{A}_1$'s only strategy is to use an attestation response of a previously generated attestation $\tau_{c_{\text{prev}}}$, on the same software configuration S_c and an old nonce N_{prev}. Let N_{curr} be the random nonce generated by $\mathcal{V}$ in the current round of attestation. The probability that $N_{\text{prev}} = N_{\text{curr}}$ is equal to 2^{-l_N}, which is negligible for a sufficiently large nonce. $\mathcal{A}_1$ will therefore output a valid attestation response only with negligible probability.

In the second case, we observe the following analogy: the security definition of resistance against forgery for a remote attestation protocol, is exactly the security definition of unforgeability of the aggregate $\mathsf{MAC}_{\mathsf{BLS}}$ scheme, namely forgery for an adversary who had access to up to $n-1$ tags. $\mathcal{A}_2$'s advantage in successfully producing a forgery is therefore the same as $\mathcal{A}_2$'s advantage in producing a valid aggregate $\mathsf{MAC}_{\mathsf{BLS}}$ forgery, which is negligible under the CDH assumption in the random oracle

model (see proof in Appendix A.1). Indeed, in order for $\mathcal{A}_2$ to generate a valid attestation on a modified state without physically compromising D_c, he needs to forge an aggregate $\mathsf{MAC}_{\mathsf{BLS}}$ scheme on all aggregated tags, assuming that at least one device in the network is honest. According to Theorem 2, aggregate $\mathsf{MAC}_{\mathsf{BLS}}$ is unforgeable provided that $\mathsf{MAC}_{\mathsf{BLS}}$ is unforgeable (see proof in Appendix A.2). The probability of $\mathcal{A}_2$ generating said valid attestation is therefore negligible.

8 CONCLUSION

In this work we introduced CoRA, the first collective attestation protocol with verifier detection for sensor networks. Collective (or swarm) attestation is a security mechanism which efficiently verifies the integrity of large numbers of devices in wireless multihop networks. CoRA leverages the aggregating property of its underlying in-network aggregation mechanism, namely aggregate $\mathsf{MAC}_{\mathsf{BLS}}$, to provide a highly scalable swarm attestation protocol with efficient verification. In order to detect the malicious injection of erroneous attestation, CoRA comprises a scalable detection algorithm, which leverages the algebraic property of algebraic MACs to generate proofs of knowledge, on a device's secret key. The detection method allows the identification of a compromised node in the network, thus preventing DoS attacks on the verifier. We provide a rigorous proof for the underlying cryptographic construction, as well as the CoRA protocol. Finally, we prove the efficiency of our scheme, based on a prototype implementation on a standard microcontroller.

REFERENCES

Ambrosin, M., Conti, M., Ibrahim, A., Neven, G., Sadeghi, A., and Schunter, M. (2016). SANA: secure and scalable aggregate network attestation. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 731–742.

Asokan, N., Brasser, F. F., Ibrahim, A., Sadeghi, A., Schunter, M., Tsudik, G., and Wachsmann, C. (2015). SEDA: scalable embedded device attestation. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 964–975.

Boldyreva, A., Gentry, C., O'Neill, A., and Yum, D. H. (2007). Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing. In *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007*, pages 276–285.

Boneh, D., Gentry, C., Lynn, B., and Shacham, H. (2003). Aggregate and verifiably encrypted signatures from bilinear maps. In *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, pages 416–432.

Boneh, D., Lynn, B., and Shacham, H. (2001). Short signatures from the weil pairing. In *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, pages 514–532.

Camenisch, J. and Stadler, M. (1997). Proof systems for general statements about discrete logarithms. Technical report.

Carpent, X., Defrawy, K. E., Rattanavipanon, N., and Tsudik, G. (2017). Lightweight swarm attestation: A tale of two lisa-s. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2017, Abu Dhabi, United Arab Emirates, April 2-6, 2017*, pages 86–100.

Chan, H., Perrig, A., and Song, D. X. (2006). Secure hierarchical in-network aggregation in sensor networks. In *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, Ioctober 30 - November 3, 2006*, pages 278–287.

Dodis, Y., Kiltz, E., Pietrzak, K., and Wichs, D. (2012). Message authentication, revisited. In *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*.

Eikemeier, O., Fischlin, M., Götzmann, J., Lehmann, A., Schröder, D., Schröder, P., and Wagner, D. (2010). History-free aggregate message authentication codes. In *Security and Cryptography for Networks, 7th International Conference, SCN 2010, Amalfi, Italy, September 13-15, 2010. Proceedings*, pages 309–328.

Falliere, N., Murchu, L. O., and Chien, E. (2010). W32.stuxnet dossier. *Symantec*.

Francillon, A., Nguyen, Q., Rasmussen, K. B., and Tsudik, G. (2012). Systematic treatment of remote attestation. *IACR Cryptology ePrint Archive*.

Francillon, A., Nguyen, Q., Rasmussen, K. B., and Tsudik, G. (2014). A minimalist approach to remote attestation. In *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE.

Ibrahim, A., Sadeghi, A., Tsudik, G., and Zeitouni, S. (2016). DARPA: device attestation resilient to physical attacks. In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks, WISEC 2016, Darmstadt, Germany, July 18-22, 2016*, pages 171–182.

Ibrahim, A., Sadeghi, A., and Zeitouni, S. (2017). Seed: secure non-interactive attestation for embedded devices. In *Proceedings of the 10th ACM Conference*

on Security and Privacy in Wireless and Mobile Networks, WiSec 2017, Boston, MA, USA, July 18-20, 2017, pages 64–74.

Katz, J. and Lindell, A. Y. (2008). Aggregate message authentication codes. In *Topics in Cryptology - CT-RSA 2008, The Cryptographers' Track at the RSA Conference 2008, San Francisco, CA, USA, April 8-11, 2008. Proceedings*, pages 155–169.

Kohnhäuser, F., Büscher, N., Gabmeyer, S., and Katzenbeisser, S. (2017). SCAPI: a scalable attestation protocol to detect software and physical attacks. In *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks, WiSec 2017, Boston, MA, USA, July 18-20, 2017*, pages 75–86.

Kohnhäuser, F., Büscher, N., and Katzenbeisser, S. (2018). SALAD: secure and lightweight attestation of highly dynamic and disruptive networks. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security, AsiaCCS 2018, Incheon, Republic of Korea, June 04-08, 2018*, pages 329–342.

Lu, S., Ostrovsky, R., Sahai, A., Shacham, H., and Waters, B. (2006). Sequential aggregate signatures and multisignatures without random oracles. In *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, pages 465–485.

MacKay, K. (2017). micro-ecc library.

Mansfield-Devine, S. (2016). Ddos goes mainstream: how headline-grabbing attacks could make this threat an organisation's biggest nightmare. *Network Security*.

Research, C. (2000). Sec 2: Recommended elliptic curve domain parameters.

Schnorr, C. (1991). Efficient signature generation by smart cards. *J. Cryptology*, 4(3):161–174.

A Security Analysis of $\mathsf{MAC}_{\mathsf{BLS}}$ and Aggregate $\mathsf{MAC}_{\mathsf{BLS}}$

A.1 $\mathsf{MAC}_{\mathsf{BLS}}$ Security

In this section, we prove Theorem 1, which states that $\mathsf{MAC}_{\mathsf{BLS}}$ is UF-CMVA in the random oracle model. Let q_H be the maximum number of queries an algorithm $\mathcal{A}$ can make to O_{Hash}, q_M the maximum number of queries to O_{MAC}, and q_V the maximum number of queries to O_{Verify}. Assume an algorithm $\mathcal{A}$ $(t, q_H, q_M, q_V, \varepsilon)-$breaks the UF-CMVA security of $\mathsf{MAC}_{\mathsf{BLS}}$. We use $\mathcal{A}$ to construct an algorithm $\mathcal{B}$ that $(t', \varepsilon)-$breaks computational Diffie-Hellman on the group $\mathbb{G}$. Using a sequence of security experiments, we use the forger $\mathcal{A}$ to build the algorithm $\mathcal{B}$ that breaks CDH on $\mathbb{G}$.

Setup. $\mathcal{B}$ is given the challenge (g, g^a, g^b). $\mathbb{G}$ being a GDH group, $\mathcal{B}$ also has access to a DDH oracle O_{DDH} that outputs 1 if an argument of the form (g, g^a, h, h^b) is a valid Diffie-Hellman tuple. $\mathcal{B}$ maintains a list (h_i, τ_i) for all hash and MAC queries. $\mathcal{A}$ can request a hash on any message of its choice, to which $\mathcal{B}$ responds with the corresponding h_i. Similarly, $\mathcal{A}$ can request a tag on any message of its choice, and receives the corresponding τ_i where $i \in \{1, ..., l\}$ and $l \leq q_M$ is the number of requests to O_{MAC}. $\mathcal{B}$ sets the operator's parameters $X = g^a$, thus implicitely setting $sk = a$.

For $1 \leq i \leq q_H$, $\mathcal{A}$ picks a random $i^* \xleftarrow{\$} \{1, ..., q_H\}$. $\mathcal{A}$ then selects a random $r_i \xleftarrow{\$} \mathbb{Z}_q^*$ for $1 \leq i \leq q_H$, sets $h_i \leftarrow (g^b) g^{r_{i^*}}$ for $i = i^*$, and $h_i \leftarrow g^{r_i}$ otherwise. If $i \neq i^*$, it sets $\tau_i \leftarrow (g^a)^{r_i}$. If $i = i^*$, it sets $\tau_{i^*} = \star$ which is a placeholder value.

Queries. When $\mathcal{A}$ requests a hash on m_i, $\mathcal{B}$ outputs h_i. When $\mathcal{A}$ requests a tag on m_i, if $i \neq i^*$ $\mathcal{B}$ outputs τ_i. Otherwise if $i = i^*$, $\mathcal{B}$ declares failure and aborts. $\mathcal{A}$ may also make verification queries on input (m_i, τ_i). $\mathcal{B}$ then uses its DDH oracle O_{DDH}, and outputs the result obtained from O_{DDH} on the tuple (g, g^a, h_i, τ_i).

Response. At some point, $\mathcal{A}$ outputs a forgery (m^*, τ^*) such that m was never queried to O_{MAC}, and $\mathsf{Verify}(sk, m, \tau) = 1$.

- The probability that $\mathcal{B}$ aborts is equal to $1/q_M$, which is inverse polynomial. We can reduce this value by repeating the experiment a polynomial number of times.
- The r_i's are selected at random, therefore h_i's are uniformly distributed in $\mathbb{G}$, making O_{Hash} a random oracle. Moreover, if $i \neq i^*$, the tags τ_i areall valid.If $\mathcal{B}$ does not abort, the simulation for $\mathcal{A}$ is indistinguishable from a real execution of the MAC algorithms.
- If $\mathcal{A}$ successfully outputs a forgery (m^*, τ^*) and $\mathcal{B}$ does not abort, $h^* = h_{i^*} = (g^b) g^{r_{i^*}}$. Therefore $h_{i^*}^a = (g^{ab} g^{r_{i^*}})$. $\mathcal{B}$ retrieves $g^{ab} = \frac{\tau^*}{(g^a)^{r_{i^*}}}$. $\mathcal{B}$ outputs the valid answer to the CDH challenge.
- $Adv_{\mathcal{A}}^{\text{UF-CMVA}}(1^\lambda) = \varepsilon / q_M$. If we repeat the experiment k times, the probability that algorithm $\mathcal{B}$ outputs a valid answer to the CDH challenge is equal to $(\frac{\varepsilon}{q_M})^k$ which is non-negligible.

A.2 Aggregate $\mathsf{MAC}_{\mathsf{BLS}}$ Security

In this section, we give the prove by reduction of Theorem 2 in the random oracle model. We consider the

setting where the reduction maintains a list KeyList of all public keys attached to each secret key sk_i.
Let $\mathcal{A}$ be an adversary against the EUF-CMVA property of aggregate $\mathsf{MAC}_{\mathsf{BLS}}$. Using $\mathcal{A}$ as a subroutine, we construct a reduction $\mathcal{B}$ against the UF-CMVA property of the $\mathsf{MAC}_{\mathsf{BLS}}$ scheme run by a challenger $\mathcal{C}$. $\mathcal{B}$ receives the system public parameters $params = (\mathbb{G}, q, g)$, and the public key pk^* for an unknown identity $X_{i^*} = g^{x_{i^*}}$ where $i^* \in \{1,...,l\}$ and $l = poly(\lambda)$. $\mathcal{B}$ has access to two oracles: an oracle $O_{\mathsf{MAC}_{\mathsf{BLS}},sk^*}$ (that we will refer to as O_{Mac} to simplify notations) for the unknown key $x^* = x_{i^*}$, and the oracle O_{Verify} that checks the validity of any message/tag pair. $\mathcal{C}$ initializes an empty list $\mathcal{M}$ that will store the subsequent messages from O_{Mac} queries.

1. $\mathcal{B}$ chooses $i^* \xleftarrow{\$} \{1,...,t\}$;
2. For $i = 1$ to t:
   - If $i \neq i^*$, select $r_i \xleftarrow{\$} \mathbb{Z}_q^*$, and set $h_i \leftarrow g^{r_i}$. Choose $x_i \xleftarrow{\$} \mathbb{Z}_q$. Set $X_i = g^{x_i}$ and add $(i, pk_i = X_i, sk_i = x_i)$ to KeyList. $iparams = \{pk_1,...,pk_t\}$.
   - If $i = i^*$, do nothing but implicitely set $sk_{i^*} = (sk^*)$.
3. $\mathcal{B}$ runs $\mathcal{A}(params, iparams)$ answering the queries as follows:
   - $\underline{\mathsf{Hash}^*(m_i)}$:$\mathcal{B}$ uses its $\mathsf{MAC}_{\mathsf{BLS}}$ hash oracle $\underline{O_{\mathrm{Hash}}}$ to output the corresponding hash h_i.
   - $\underline{\mathsf{AggMac}^*(\{m_1,...,m_q\})}$: $\mathcal{B}$ computes the aggregate tag τ_i using the known secret key sk^*. If there is $i \in \{1,...,q\}$ such that $i = i^*$, $\mathcal{B}$ queries its own oracle O_{Mac} on m_i. Finally, $\mathcal{B}$ returns the τ_i.
4. At some point, $\mathcal{A}$ outputs $M = \{m_1,...,m_n)\}$ and an aggregate tag τ. Let j be the first index such that $\mathcal{A}$ has never queried m_j to Mac^*, and $pk_j \neq pk^*$. If $j \neq i^*$ then $\mathcal{B}$ abort; Otherwise, proceed as follows:
   (a) We therefore consider the case where $j = i^*$. $\mathcal{B}$ computes the tag τ^* from $\tau = \prod_{i=1}^{t} h_i^{x_i}$ the following way: $\tau^* = \frac{\tau}{\prod_{l=1, l\neq i^*}^{t} h_l^{x_l}}$.
   (b) $\mathcal{B}$ has the secret keys corresponding to all identifiers that are not i^*, he is therefore able to compute a valid τ^*. When $j = i^*$, $\mathcal{B}$ simply queries $\mathsf{Mac}^* m$ to get the corresponding tag. Finally, $\mathcal{B}$ outputs (m_j, τ^*) as a valid forgery of a $\mathsf{MAC}_{\mathsf{BLS}}$ tag.

The proof of correctness follows from the following observations:

- The probability that $\mathcal{B}$ aborts is equal to $1/t$, which is inverse polynomial. If $\mathcal{B}$ does not abort, the simulation for $\mathcal{A}$ is indistinguishable from a real execution of the aggregate MAC algorithms.
- If $\mathcal{A}$ successfully outputs a forgery for the aggregate MAC scheme, and $\mathcal{B}$ does not abort, the assumption stipulates that $\mathcal{A}$ has never queried $\mathcal{B}$ on m_j. Therefore $\mathcal{B}$ has never queried $\mathsf{Mac}^*(m_j)$. The success of algorithm $\mathcal{A}$ means that $\tau = \prod_{i=1}^{t} \tau_i$ containing τ^*. The tag $\mathcal{B}$ outputs is therefore a valid forgery of a $\mathsf{MAC}_{\mathsf{BLS}}$ tag.