Multi-Key Fully-Homomorphic Aggregate MAC for Arithmetic Circuits

Suvasree Biswas and Arkady Yerukhimovich

George Washington University
suvasree@gwu.edu arkady@gwu.edu

Abstract. Homomorphic message authenticators allow a user to perform computation on previously authenticated data producing a tag σ that can be used to verify the authenticity of the computation. We extend this notion to consider a multi-party setting where we wish to produce a tag that allows verifying (possibly different) computations on all party's data at once. Moreover, the size of this tag should not grow as a function of the number of parties or the complexity of the computations. We construct the first aggregate homomorphic MAC scheme that achieves such aggregation of homomorphic tags. Moreover, the final aggregate tag consists of only a single group element. Our construction supports aggregation of computations that can be expressed by bounded-depth arithmetic circuits and is secure in the random oracle model based on the hardness of the Computational Co-Diffie-Hellman problem over an asymmetric bilinear map.

Keywords: homomorphic authenticators · aggregate MAC · verifiable computation

1 Introduction

Consider a scenario where a scientist wishes to collect aggregate measurements about some scientific phenomena, for example average or standard deviation of ocean temperature around the world. She can deploy sensors to collect data in different regions and, to minimize communication, have the sensors upload their readings to the cloud. Moreover, since the scientist is only interested in aggregate statistics, she does not need to view all of the uploaded measurements, just the resulting aggregate statistics. So, instead of having the cloud store all the collected data, she can have it compute the aggregates and only review the results.

However, this requires that the scientist rely on the cloud to compute the aggregate statistics. A malicious cloud can change the original readings or add error into the computations to produce faulty results. Thus, the scientist needs to check that the aggregate statistics are correct for *all* the sensors even without seeing the original sensor readings. Moreover, to reduce communication and storage, we want the "proof" of correctness to be small – of size not growing with either the complexity of the computed statistics or the number of sensors.

Moreover, even if the cloud is able to learn the secret keys of some of the sensors, we want the authenticity of the computation on the remaining sensors' data to be preserved.

To address this challenge we turn to the use of homomorphic authenticators [1, 2, 3, 4, 5, 6, 7]. Homomorphic authenticators allow a user to use their secret key to authenticate an input x. Anybody can then use an associated (public) evaluation key to evaluate a program $\mathcal{P}$ on the authenticated input producing the result along with a tag authenticating the output. Finally, a party with knowledge of the user's verification key – a public key in the case of signatures, and a secret key in the case of message authentication codes – can verify that the final output was computed correctly on the originally authenticated input. In our previously described application this would amount to the scientist provisioning secret keys to each sensor, the sensor authenticating their measurement and sending it to the cloud, who evaluates the program $\mathcal{P}$ allowing the scientist to verify the output.

Originally, homomorphic authenticators focused on the case of a single user authenticating their input, but more recent work [6, 7] has considered the case of homomorphic authenticators over multiple users. These allowed authenticating computation that took input from multiple users. The first work to consider this setting by Fiore et al. [6] restricted the computation to low-depth circuits. A very recent breakthrough work by Anthoine et al. [7] overcame this limitation, but the resulting authenticators grow either with the number of users or the depth of the computation.

Our goal is different from these works. First, while we aim to support multiple users with different authentication keys so that compromising one user's key does not compromise the other users, we only consider computations that work over the inputs of a single user – i.e., we only wish to compute over readings for each sensor, not across sensors. However, to minimize storage and communication we insist that the size of the final authenticator be independent of the number of users whose computation is authenticated and the depth and size of the computed circuits. Specifically, it will consist of only a single group element. Thus our goal is both weaker and stronger than the prior work supporting a more restrictive class of computations, but enforcing a much stricter length bound on the authenticator.

In order to compress the authenticators to achieve this length bound, we turn to another well-studied tool in the cryptographic literature – aggregate signatures [8, 9, 10, 11]. Aggregate signatures allow the compression of multiple signatures into a single short signature that does not grow with the number of input signatures. However, they do not provide any homomorphic functionality and the original inputs are necessary to verify the signature.

We build on these two tools to develop a primitive we call a HA-MAC which combines the benefits of both of these primitives allowing verification of computation over the sensors' inputs while only requiring an authenticator consisting of only one group element.

1.1 Our Construction

Our Homomorphic Aggregate MAC (HA-MAC) combines the properties of homomorphic authenticators and aggregate authenticator to achieve functionality and security that are a good fit for the outsourced measurement use-case described previously. We briefly describe these in what follows before sketching the intuition behind our construction.

Functionality A HA-MAC is a symmetric-key primitive where each of the P users is given a secret key sk_i. Using this secret key each user can authenticate inputs of his choice. An evaluator who does not know the secret key, but has a corresponding evaluation key ek_i can evaluate (possibly different) programs over each user's inputs and aggregate the results of any subset of the users into a short aggregate MAC. Finally, a verifying party who knows all of the secret keys of the included users can use this MAC to verify that the evaluator correctly computed all of the aggregated values. Importantly, the verifier can do this even without knowing the original values authenticated by the users.

Security Our HA-MAC also provides strong security guarantees. The MAC guarantees the integrity of an honest user's input and computation even if all of the remaining users collude with the evaluator meaning that users have nothing to fear from participating. Moreover, security is achieved even against an adversary who is additionally allowed to ask for an unbounded number of authentications, as well as MAC verifications and aggregate MAC verifications with tags including the honest user. We prove our HA-MAC secure in the random oracle model under the standard co-CDH assumption over pairing-friendly elliptic curves.

Specifically, we prove the following theorem:

Theorem 1 (Informal). *For any pairing friendly elliptic curve E with a bilinear map $e : \mathcal{G}_1 \times \mathcal{G}_2 \to \mathcal{G}_T$ and efficiently computable isomorphism $\psi : \mathcal{G}_2 \to \mathcal{G}_1$ such that the co-Computational Diffie-Hellman (co-CDH) problem is hard, the HA-MAC construction Θ given in Figure 2 is a secure HA-MAC in the random oracle model.*

1.2 Overview of Our Techniques

We begin with the construction of a single-user homomorphic message authentication code due to Catalano and Fiore [3]. In their construction, all authentication tags correspond to polynomials in $\mathbb{Z}_p[x]$ represented as a vector of their coefficients. To authenticate an input $m \in \mathbb{Z}_p$ with a label[1] $\tau \in \{0,1\}^\lambda$ the initial authentication tag is a degree-1 polynomial $y \in \mathbb{Z}_p[x]$ such that $y(0) = m$ and $y(x) = \mathrm{PRF}_K(\tau)$ for a hidden point $x \in \mathbb{Z}_p$ and a PRF key K. Looking forward, x and K will be part of a user's secret key.

[1] Labels are used to identify the input wires in an arithmetic circuit describing a homomorphic computation.

For any depth-D arithmetic circuit f with n input labels, the homomorphic tag authenticating the output of f is a degree-D polynomial $y_f \in \mathbb{Z}_p[x]$ constructed as follows from the input polynomials y_i. Going through the circuit f gate-by-gate, for every addition gate, we add the input polynomials (by adding the coefficients). For every multiplication gate, we multiply the input polynomials (by computing a convolution of their coefficients). It is easy to see that these operations are naturally homomorphic with respect to the evaluation of the polynomials at any evaluation point. Specifically, we observe that $y_f(0) = f(y_1(0), \ldots, y_n(0)) = f(m_1, \ldots, m_n)$ where $y_1, \ldots, y_n$ are the original input tags on inputs $m_1, \ldots, m_n$ and $y_f(x) = f(y_1(x), \ldots, y_n(x)) = f(\mathrm{PRF}_K(\tau_1), \ldots, \mathrm{PRF}_K(\tau_n))$ where x is the hidden point used in constructing the initial authentication tags. Observe that if the hidden evaluation point x is random and unknown to an adversary, then to check the validity of a claimed Mac y', it is sufficient to check whether these two equalities hold. Specifically, on an input polynomial y' if $y'(0) = y_f(0)$ and $y'(x) = y_f(x)$ then by the Schwartz-Zippel lemma, with overwhelming probability, $y' = y_f$ is a valid tag. Note, however, that when computed in this way the degree of the output polynomial y_f is D and thus $D + 1$ coefficients are necessary to represent this polynomial resulting in a tag that grows linearly with the depth of the circuit f.

To avoid this increase in the size of the tags, Catalano and Fiore instead encode and evaluate these polynomials in the exponent in a multiplicative group $\mathcal{G}$. More concretely, for some program $\mathcal{P} = (f, (\tau_i)_{\forall i \in [n]})$, let $(y_i)_{\forall i \in D}$ be the coefficients of the polynomial y_f computed as before. Succinct tag Λ is computed as $\Lambda = \prod_{i=1}^{D} (u^{x^i})^{y_i}$ where the values u^{x^i} for $i \in [D]$ are part of a user's public evaluation key and u is a generator of the group $\mathcal{G}$. Intuitively, Λ is computed by taking an inner product between the coefficient vector of y with powers of x excluding the degree-0 coefficient. This corresponds to evaluating $y(x) - y(0)$ in the exponent, so $\Lambda = u^{y(x)-y(0)}$. Now, on input a tuple $(m, \mathcal{P}, \Lambda)$, we can perform the Schwartz-Zippel test in the exponent by checking that $u^m \cdot \Lambda = u^\rho$ where $\rho = y(x)$, the evaluation of the polynomial at the hidden point x. For purposes of the security proof and in order to allow further functionality instead of performing the above check directly – as done by Catalano and Fiore – we instead perform this check using a bilinear pairing to map the values into a target group $\mathcal{G}_T$. Specifically, for groups $(\mathcal{G}_1, \mathcal{G}_2)$ with a bilinear map $e : \mathcal{G}_1 \times \mathcal{G}_2 \to \mathcal{G}_T$, Λ is computed in $\mathcal{G}_1$. We then choose a random element $w = \mathcal{H}_2(m) \in \mathcal{G}_2$ (for a hash function $\mathcal{H}_2$ modeled as a random oracle) and use the bilinear map to compute $\mathrm{e}(\Lambda, w)$, $\mathrm{e}(u^m, w)$, and $\mathrm{e}(u^\rho, w)$ and use these values to verify that $\mathrm{e}(u^m, w) \cdot \mathrm{e}(\Lambda, w) = \mathrm{e}(u^\rho, w)$.

Armed with this single-user homomorphic Mac construction, we now recall our goal of an aggregate homomorphic Mac. Specifically, we wish to aggregate the Macs of a subset U of users to authenticate computations performed by each of the users. An immediate way to realize this aggregation is via the trivial approach of concatenating the homomorphic Macs of all the users in U. Specifically, each user $l \in U$ has a secret key sk_l and evaluation key ek_l, where ek_l contains l's generator u_l for the previously described Mac. Then, an aggregate Mac for U

can just consist of $(\Lambda_1, \ldots, \Lambda_{|U|})$ and can be verified by checking that for all $l \in U$ with $w_l = \mathcal{H}_2(m_l)$,

$$\mathrm{e}(\Lambda_l, w_l) = \mathrm{e}(u_l, w_l^{\rho}) \cdot \mathrm{e}(u_l, w_l^{m_l})^{-1} \tag{1}$$

However, this trivial aggregation fails to meet our goal of an aggregate Mac that does not grow with the number of parties in the set U. Thus, to achieve this goal, we modify the aggregation procedure to combine all of the users' Macs into a single group element. Intuitively, this is done by taking a random linear combination of the users' Macs in the exponent. To ensure that the verifier can recompute the linear combination for verification, but at the same time an adversary cannot predict the random weights that will be used in the linear combination we again turn to the hash function $\mathcal{H}_2$. Specifically, for any subset of users $U \subseteq P$, for each user in U the aggregate algorithm computes $\mathrm{e}(\Lambda_l, w_l)$ as before and then it multiplies all of these into one group element in $\mathcal{G}_T$. Using $\mathrm{e}(g_1, g_2)$ as the generator in $\mathcal{G}_T$, correct aggregation under evaluation keys $(\mathsf{ek}_l)_{\forall l \in U} \leftarrow (u_l^{x_{l_0}^k})_{\forall k \in D}$ evaluates to the following in the exponent:

$$\Sigma_{\forall l \in U}\, \gamma_l b_l \left(m_l + \sum_{k=1}^{D} y_{l,k} x_{l_0}^k \right) = \Sigma_{\forall l \in U}\, \gamma_l b_l f_l(r_{\tau_{l,1}}, \ldots, r_{\tau_{l,n}})$$

where γ_l and b_l are exponents such that $g_2^{b_l} \leftarrow w_l$, $g_1^{\gamma_l} \leftarrow u_l$ for some $\gamma_l \leftarrow\!\!\$\, \mathbb{Z}_p$. Note that b_l and γ_l are both secret to the adversary since b_l is the discrete log of the hash output w_l and γ_l is the discrete log of the random generator u_l.

Verification of aggregate MAC σ^* involves checking whether σ^* is equal to $\mathrm{e}(g_1, g_2)^{\Sigma_{\forall l \in U}\, \gamma_l b_l \left(f_l(r_{\tau_{l,1}}, \ldots, r_{\tau_{l,n}}) - m_l\right)}$. Since the adversary does not know γ_l or b_l, he cannot produce a forged Mac σ^* that passes this verification.

2 Related Work

Homomorphic Authenticators: Since their introduction by Ronald Rivest [12], homomorphic authenticators have been widely investigated (e.g. [13, 1, 2, 3, 4, 5, 6, 7]) both in the public-key (i.e., homomorphic signatures) and private-key (i.e., homomorphic Mac) variants. Of these, we follow the line of work on fully-homomorphic authenticators initiated by Gennaro and Wichs [4] and follow-on work [3, 6, 7].

In particular, our starting point is the work of Catalano and Fiore [3] who showed how to build practical fully-homomorphic Macs from the ℓ-Diffie-Hellman Inversion assumption. However, this work is restricted to the single-key setting whereas we are aiming to support authentications under multiple keys. Follow-on work by Fiore et al. [6] did achieve support for multi-key homomorphic Macs, however they were limited to only low-degree computations. Very recently, Anthoine et al. [7] showed how to lift this limitation achieving homomorphic Macs for arbitrary computation across inputs authenticated by multiple clients. However, their resulting Mac, while sublinear in the number of clients and the size

of the computation, still required that the number of group elements grows with the number of parties. We, on the other hand, only aim for a weaker primitive allowing homomorphic computation only on inputs from each party separately followed by an aggregation of the results. But, this allows us to achieve a final Mac consisting of only a single group element.

Aggregate Signatures and MACs: Another line of work closely related to ours is aggregate signatures and Macs (e.g. [8, 9, 10, 14, 11, 15]). These primitives focus on aggregating multiple signatures or Macs into a single (very) short authenticator. However, they do not consider computation on these authenticators. Our construction directly builds on the work of Boneh et al. [8] and thus inherits their need of a random oracle. While later work [14] showed how to avoid this random oracle assumption, this was at the cost of very strong computational assumptions. We see constructing a HA-MAC scheme secure in the standard model as an interesting open question.

Verifiable Computation: Homomorphic aggregate MACs are also closely related to the more general notion of verifiable computation [16, 17, 18, 19, 20, 21, 22, 23, 24]. Generally, this line of work differs from homomorphic Macs in that verifiable computation often requires interaction whereas homomorphic Macs are non-interactive. Moreover, homomorphic Macs allow any party with knowledge of the public evaluation keys to verifiably compute functions on authenticated inputs with no additional communication. On the other hand, verifiable computation is heavily focused on reducing the amount of computation necessary to verify while we focus primarily on reducing communication. We refer readers to [4] for a more in-depth discussion.

Succinct Non-Interactive Arguments of Knowledge: Another way to achieve a homomorphic aggregate Mac is via the use of SNARKs [25, 26, 27, 28]. While SNARKs do provide an alternative solution to ours, they are generally more expensive – especially for the prover, and necessarily rely on non-falsifiable assumptions [29].

3 Preliminaries

3.1 Notation

We let $\lambda \in \mathbb{N}$ denote a security parameter. A function ν: $\mathbb{N} \to \mathbb{N}$ is *negligible* if $\nu(\lambda) = O(\lambda^{-c})$ for every constant $c > 0$ and a function p is polynomial if $p(\lambda) = O(\lambda^c)$ for some constant $c > 0$. For a finite set S, we use $x \leftarrow\!\!\$\, S$ to denote sampling a uniformly random element from S. When analyzing the concrete security of a primitive, we say that a primitive is (t, ϵ)-secure if any adversary running for time *at most* t succeeds in breaking the scheme with probability *at most* ϵ. We use $\mathbb{F}$ to refer to a finite field, bold variables (e.g., $\mathbf{r}$) to represent vectors, and PRF to denote a pseudorandom function. We refer the reader to the text by Katz and Lindell [30] for the formal definition.

3.2 Arithmetic Circuits

Definition 1. ***Arithmetic Circuit [31, 3, 32]*** *An arithmetic circuit over a field $\mathbb{F}$ and a set of variables $X = (\tau_i)_{\forall i \in [n]}$ is a directed acyclic graph representing polynomial computation. We say that a polynomial $f \in \mathbb{F}[\tau_1, \ldots, \tau_n]$, over a field $\mathbb{F}$ is computable by a circuit of size s and depth D if there exists a directed acyclic graph with s nodes and depth D such that its leaf nodes are labelled by variables or field constants, internal nodes are labelled with $+$ and $\times$, and f is the polynomial computed at the root node.*

In this paper, we consider evaluations of arithmetic circuits over field elements and over polynomials. We briefly describe these below:

- Computing on field elements: We interpret f as a bounded depth arithmetic circuit over $\mathbb{F}_p$. Meaning f on input $\mathbf{x} \in \mathbb{F}_p^n$ outputs a scalar in $\mathbb{F}_p$. Here the evaluation takes place gate by gate where each gate is either $+$ or $\times \mod p$.
- Computing on polynomials: We interpret f as a D degree polynomial that f computes. Meaning f on input n degree-1 polynomials $(\{y_i\}_{\forall i \in [n]}$ outputs a degree-D polynomial y^* over $\mathbb{F}_p$. We represent polynomials in coefficient notation, i.e., each $y_i = (y_i^0, y_i^1)$ and y^* is a vector of $D+1$ coefficients. Here the evaluation is performed gate by gate evaluation where multiplication gates correspond to polynomial multiplication (corresponding to a convolution of the coefficient vectors) and addition gates correspond to polynomial additions (corresponding to component-wise addition of the coefficient vectors.)

3.3 Labeled Programs

We recall the definition of labeled programs [4, 3]. A labelled program is a tuple $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$, where $f : \mathbb{F}^n \to \mathbb{F}$ is a function represented by an arithmetic circuit and $\tau_1, \ldots, \tau_n$ are binary strings that are used as labels to identify the inputs of this function.

All data is authenticated with a label $\sigma \leftarrow Auth(sk, (\tau, m))$ which is used to identify a particular input to the circuit and to indicate which input of the circuit it corresponds to. Specifically, given a labeled program $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ and a set of tags $(\sigma_1, \ldots, \sigma_n)$ that authenticate message m_i under label τ_i, homomorphic evaluation allows anyone to evaluate the program $\mathcal{P}$ on messages m_i as long as the labels τ_i match those specified in the program.

We denote the identity program associated with label τ as $\mathcal{I}\tau = (g_{id}, \tau)$, where g_{id} signifies the canonical identity function, i.e., $g_{id}(x) = x$ for all inputs x, and $\tau \in \{0,1\}^*$ denotes an input label.

We recall the definition of well defined program [3].

Definition 2. *A labeled program $\mathcal{P} = \big(f, (\tau_i)_{\forall i \in [n]}\big)$ is considered **well-defined** over a table $\mathcal{T}$ if one of the following conditions holds:*

- *Either, for every label $\tau_i \in \mathcal{P}$, $\mathcal{T}$ contains an entry $(\tau_i, m_i, \cdot)$ for some message m_i previously authenticated under label τ_i. Or,*

– *if there exists an index $i \in [n]$ such that the tuple $(\tau_i, \cdot, \cdot)$ is absent from $\mathcal{T}$ then the output $f(\{m_j\}_{(\tau_j, m_j, \cdot) \in \mathcal{T}} \cup \{\tilde{m}_j\}_{(\tau'_j, \cdot, \cdot) \notin \mathcal{T}})$ remains constant over all possible choices of $\tilde{m}_j$ from $\mathcal{M}$.*
In this case the output of $f(\{m_j\}_{(\tau_j, m_j, \cdot) \in \mathcal{T}} \cup \{\tilde{m}_j\}_{(\tau'_j, \cdot, \cdot) \notin \mathcal{T}})$ remains the same over all possible choices for $\tilde{m}_j$ from $\mathcal{M}$. I.e., the output of f is fixed given the already authenticated inputs.

We also recall proposition 1 from [3] below for completeness.

Proposition 1 ***[3]*** *Let $\lambda, n \in \mathbb{N}$ and let $\mathcal{F}$ be the class of arithmetic circuits $f : \mathbb{F}^n \to \mathbb{F}$ over a finite field $\mathbb{F}$ of order p and such that the degree of f is at most d, with $\frac{d}{p} < \frac{1}{2}$. Then, there exists a probabilistic algorithm that given f st $f \in \mathcal{F}$ decides whether $\exists y \in \mathbb{F}$ such that $f(\mathbf{u}) = y$, $\forall \mathbf{u} \in \mathbb{F}^n$ (i.e., if f is constant) and is correct with probability at least $(1 - 2^{-\lambda})$.*

Proof. The algorithm begins by sampling uniformly at random $\lambda + 1$ tuples $\{\mathbf{u}_i\}_{i=0}^{\lambda}$ from $\mathbb{F}^n$. It then checks if $f(\mathbf{u}_0) = \cdots = f(\mathbf{u}_\lambda)$. If this condition holds, it concludes *constant*; otherwise, it concludes *non-constant*. If f is constant, the algorithm is correct with certainty. In the case where f is not constant, the probability of the algorithm being wrong is essentially the probability that $f(\mathbf{u}_0) = \cdots = f(\mathbf{u}_\lambda)$ over all possible choices of $\mathbf{u}_i$'s $i \in [0, \lambda]$. This probability can be bounded above by $(\Pr_{\mathbf{u}_i \leftarrow \mathbb{F}^n}[f(\mathbf{u}_i) = y_0 \,|\, y_0 = f(\mathbf{u}_0)])^\lambda \leq \left(\frac{d}{p}\right)^\lambda \leq 2^{-\lambda}$, where the upper bound by $\frac{d}{p}$ follows from the Schwartz-Zippel Lemma[33], [34].

3.4 Computational Assumptions

We now recall the necessary definitions of a bilinear maps and associated computational hardness assumptions.

Let $\mathcal{G}_1$ and $\mathcal{G}_2$ are two (multiplicative) cyclic groups of prime order p. g_1 is a generator of $\mathcal{G}_1$ and g_2 is a generator of $\mathcal{G}_2$. ψ is an isomorphism from $\mathcal{G}_2$ to $\mathcal{G}_1$, with $\psi(g_2) = g_1$. e is a bilinear map $\mathrm{e} : \mathcal{G}_1 \times \mathcal{G}_2 \to \mathcal{G}_T$ such that $\mathrm{e}(g_1, g_2) \neq 1$. For our paper, we consider subgroups $\mathcal{G}_1, \mathcal{G}_2$ as $\mathcal{G}_1 \subseteq E(\mathbb{F}_q)$ and $\mathcal{G}_2 \subseteq E(\mathbb{F}_{q^l})$ where E is the elliptic curve over the respective finite field. Two groups $(\mathcal{G}_1, \mathcal{G}_2)$ are a bilinear group if the group action on either can be computed in one time unit, the map ψ from $\mathcal{G}_2$ to $\mathcal{G}_1$ can be computed in one unit time, a bilinear map e exists and e is computable in one unit time.

Computational co-Diffie Hellman Problem The Computational co-Diffie-Hellman problem(co-CDH) over groups $\mathcal{G}_1$ and $\mathcal{G}_2$ is defined as the problem of computing $h^a \in \mathcal{G}_1$ given $g_2, g_2^a \in \mathcal{G}_2$ and $h \in \mathcal{G}_1$.

Next we define the advantage of any PPT algorithm $\mathcal{A}$ in solving the Computational co-Diffie-Hellman problem in groups $\mathcal{G}_1$ and $\mathcal{G}_2$ as

$$\mathbf{Adv}_{\mathcal{A}}^{co-CDH} \stackrel{\text{def}}{=} \Pr[\mathcal{A}(g_2, g_2^a, h) = h^a | a \leftarrow_\$ \mathbb{Z}_p, h \leftarrow_\$ \mathcal{G}_1] \tag{2}$$

where the probability is taken over the uniform random choice of a from $\mathbb{Z}_p$ and h from $\mathcal{G}_1$ and over the coin tosses of $\mathcal{A}$.

Definition 3. ***co-CDH [35, 8]*** *Co-CDH is (t', ϵ')-hard over $(\mathcal{G}_1, \mathcal{G}_2)$ if for any adversary running in time at most t', $\mathbf{Adv}_{\mathcal{A}}^{co-CDH} < \epsilon'$.*

OBSERVATION: We note the following useful property that holds for any tuple (g_2, g_2^a, h, h^b) where $g_2 \in \mathcal{G}_2$, and $h \in \mathcal{G}_1$

$$a = b \bmod p \iff \mathrm{e}(h, g_2^a) = e(h^b, g_2) \tag{3}$$

4 Multi-Key Fully-Homomorphic Aggregate MAC (HA-MAC) for Bounded-Depth Arithmetic Circuits

We now describe our main primitive, a multi-key homomorphic MAC that allows authenticating arbitrary bounded-depth computation on each user's inputs and aggregating the resulting MACs into a single short tag that can be used to verify all users' computations. We now proceed to describe the functionality and security of this primitive.

4.1 Functionality

In the description below, we abuse notation to use P both as the set of parties and the number of parties (i.e., $|P|$). A multi-key fully-homomorphic aggregate MAC consists of the following six algorithms:

- $\mathsf{KeyGen}(1^\lambda, 1^n, 1^D, 1^P)$: On input the security parameter $\lambda \in \mathbb{N}$, input size n, depth bound D, and number of users P, KeyGen outputs a secret key and a public evaluation key $(\mathsf{sk}, \mathsf{ek})$ for all P parties.
- $\mathsf{Auth}(\mathsf{sk}_l, \tau, m)$: On input the secret key sk_l of user $l \in P$ and a message $m \in \mathcal{M}$ with input label $\tau \in \{0,1\}^\lambda$, Auth outputs a message authentication code σ.
- $\mathsf{Eval}(\mathsf{ek}_l, f, \sigma_1, \ldots, \sigma_n)$: On input the evaluation key of party $l \in P$, a function $f : \mathcal{M}^n \to \mathcal{M}$ and input message authentication codes $\sigma_1, \ldots, \sigma_n$, Eval outputs a message authentication code Λ. Note that since the parties' ek's are public any party is able to run Eval if it has valid input MACs.
- $\mathsf{Ver}(\mathsf{sk}_l, m, \mathcal{P}, \Lambda)$: On input secret key sk_l for some $l \in P$, a program $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$, a message m and a message authentication code Λ, Ver outputs $0/1$ to indicate whether Λ is a valid tag authenticating that m is a valid output of computation of program $\mathcal{P}$.
- $\mathsf{Aggregate}((\Lambda_l, m_l)_{\forall l \in U})$: On input $|U|$ tuples of message and message authentication codes for some set of parties $U \subseteq P$, $\mathsf{Aggregate}$ outputs an HA-MAC σ^*.
- $\mathsf{AggVer}((m_l, \mathsf{sk}_l, \mathcal{P}_l)_{\forall l \in U}, \sigma^*)$: On input $|U|$ tuples of (message, secret key, program) for some subset of parties $U \subseteq P$ and HA-MAC σ^*, AggVer outputs a $0/1$ indicating whether σ^* is a valid HA-MAC on messages $m_1, \ldots, m_{|U|}$ resulting from evaluating the programs $\mathcal{P}_1, \ldots, \mathcal{P}_{|U|}$ for each of the $|U|$ parties.

We now define three notions of correctness that a HA-MAC scheme must satisfy.

Authentication Correctness:
For any message $m \in \mathcal{M}$, any label $\tau \in \{0,1\}^\lambda$, for all $(\mathsf{sk}, \mathsf{ek}) \leftarrow\!\!\$\ \mathsf{KeyGen}(1^\lambda, 1^n, 1^D, 1^P)$ and any tag $\sigma \leftarrow\!\!\$\ \mathsf{Auth}(\mathsf{sk}, \tau, m)$ we require that

$$Pr[\mathsf{Ver}(\mathsf{sk}, m, \mathcal{I}_\tau, \sigma) = 1] = 1$$

where $\mathcal{I}_\tau$ is a special identity program that on input m outputs m and the probability is taken over the random coins of KeyGen.

Evaluation Correctness:
For any $(\mathsf{sk}, \mathsf{ek}) \leftarrow\!\!\$\ \mathsf{KeyGen}(1^\lambda, 1^n, 1^D, 1^P)$, a function $f : \mathcal{M}^n \rightarrow \mathcal{M}$, any set of message tag tuple $\{(m_i, \sigma_i)\}_{i=1}^n$, if for all $i \in [n]$ $\mathsf{Ver}(\mathsf{sk}, m_i, \mathcal{I}_{\tau_i}, \sigma_i) = 1$ then for $m^* \leftarrow f((m_i)_{i \in n})$ and $\Lambda \leftarrow \mathsf{Eval}(\mathsf{ek}, f, \sigma_1, \ldots, \sigma_n)$ we require that

$$\mathsf{Ver}(\mathsf{sk}, m^*, \mathcal{P}, \Lambda) = 1$$

Aggregation Correctness:
For any set of parties $U \subseteq P$, given $|U|$ tuples $(m_l, \Lambda_l, \mathcal{P}_l)_{\forall l \in U}$ such that $\mathsf{Ver}(\mathsf{sk}_l, m_l, \mathcal{P}_l, \Lambda_l) = 1$ for all $l \in U$ then,

$$Pr[\mathsf{AggVer}((\mathsf{sk}_l, m_l, \mathcal{P}_l)_{\forall l \in U}, \sigma^*) = 1] = 1$$

where $\sigma^* \leftarrow \mathsf{Aggregate}((m_l, \Lambda_l)_{\forall l \in U})$ and the probability is taken over the random coins of KeyGen.

4.2 Security

We now turn to defining security of an HA-MAC scheme. Roughly, we want a strong definition of security where an adversary can corrupt, and hence learn the secret keys of, all the parties except one designated *challenge* party. Without loss of generality we refer to this party as p_1.

We allow the adversary to see authentications under secret key of p_1 on messages and labels of its choice and he can thus evaluate programs of his choice on these messages and to aggregate the resulting tags with arbitrary computations over inputs from all other parties. Additionally, we allow the adversary to ask verification and aggregate verfication queries on initial and aggregate MACs that include p_1 respectively.

Informally, security of our HA-MAC requires that the adversary not be able to forge original or aggregate MACs that include p_1's input. Of course, the adversary can run evaluation on the MACs that he has received for p_1. Thus, what we require is that $\mathcal{A}$ not be able to produce a valid MAC or aggregate MAC that authenticates some program $\mathcal{P}'$ for p_1 such that $\mathcal{P}'$ cannot be computed from the inputs and labels on which $\mathcal{A}$ has requested tags. This is akin to the usual chosen message security of a MAC.

This definition guarantees that even if all but one parties are controlled by the adversary any inputs and computations provided by an honest party will be validly authenticated. So, the verifier can be certain that the the honest party's values are correctly computed given the final aggregate MAC.

Formally, we define existential unforgeability against a chosen message attack for a homomorphic aggregate MAC using the game $\mathbf{G}^{\text{HA-UF-CMA}}_{\Theta,\mathcal{A}}(1^\lambda, 1^n, 1^D, 1^P)$ between a challenger and an adversary $\mathcal{A}$ given in Figure 1.

$\underline{\mathbf{G}^{\text{HA-UF-CMA}}_{\Theta,\mathcal{A}}(1^\lambda, 1^n, 1^D, 1^P)}$

INITIALIZE$(1^\lambda, 1^n, 1^D, 1^P)$:
1 $\mathcal{T} \leftarrow \emptyset$
2 $(\mathsf{sk}_l, \mathsf{ek}_l)_{\forall l \in P} \leftarrow\!\!\$\ \mathsf{KeyGen}(1^\lambda, 1^n, 1^D, 1^P)$
3 Return ek_1, $(\mathsf{ek}_l, \mathsf{sk}_l)_{\forall l \in [2,P]}$

AUTHO(τ, m):
4 if $(\tau, m, \cdot) \in \mathcal{T}$, $\sigma \leftarrow \mathcal{T}(\tau, m, \cdot)$
5 if $(\tau, m, \cdot) \notin \mathcal{T}$, $\sigma \leftarrow \mathsf{Auth}_{\mathsf{sk}_1}(\tau, m)$; $\mathcal{T} = \mathcal{T} \cup (\tau, m, \sigma)$
6 if $(\tau, \cdot, \cdot) \in \mathcal{T}$, ignore
7 Return σ

VERO$(m, \mathcal{P}, \sigma)$:
8 Return $\mathsf{Ver}_{\mathsf{sk}_1}(m, \mathcal{P}, \sigma)$

AVERO$((m'_j, \mathcal{P}'_j)_{\forall j \in U}, (\mathsf{ek}'_j)_{\forall j \in U \setminus \{1\}}, \sigma')$: // $U \subseteq P, (\mathsf{ek}'_j)_{\forall j \in U \setminus \{1\}} \subseteq (\mathsf{ek}_j)_{\forall j \in P}$
9 Return $\mathsf{AggVer}((m'_j, \mathcal{P}'_j)_{\forall j \in U}, (\mathsf{sk}_1, \mathsf{sk}_j)_{\forall j \in U \setminus \{1\}}, \sigma')$

FINALIZE(IN):
10 if $IN = \left((m^*_j, \mathcal{P}^*_j)_{\forall j \in U}, (\mathsf{ek}^*_j)_{\forall j \in U \setminus \{1\}}, \sigma^*\right)$
11 Return $\mathsf{Check}_A((m^*_j, \mathcal{P}^*_j)_{\forall j \in U}, (\mathsf{ek}^*_j)_{\forall j \in U \setminus \{1\}}, \sigma^*))$
12 else if $IN = (m^*, \mathcal{P}^*, \sigma^*)$
13 Return $\mathsf{Check}_V(m^*, \mathcal{P}^*, \sigma^*)$

Fig. 1: Game defining required security game for Θ

OUTPUT We say that $\mathcal{A}$ wins, i.e. that the output of $\mathbf{G}^{\text{HA-UF-CMA}}_{\Theta,\mathcal{A}} = 1$ iff FINALIZE returns 1. This happens if either $\mathsf{Check}_A = 1$ or $\mathsf{Check}_V = 1$.

We now describe the Check_A and Check_V algorithms that we use to define the notion of a forgery. These algorithms capture what is meant by a valid aggregate forgery and a valid forgery respectively. As mentioned previously an adversary can easily produce valid MACs and aggregate MACs on inputs for which he has already seen valid MACs, thus these algorithms check whether a claimed forgery was really for something new. To define these formally, we turn to the notion of well-defined program with respect to a table $\mathcal{T}$ as defined in Section 3.3.

ALGORITHM Check_A For any input $\left((m^*_j, \mathcal{P}^*_j)_{\forall j \in U}, (\mathsf{ek}^*_j)_{\forall j \in U \setminus \{1\}}, \sigma^*\right)$, $\mathsf{Check}_A\left((m^*_j, \mathcal{P}^*_j)_{\forall j \in U}, (\mathsf{ek}^*_j)_{\forall j \in U \setminus \{1\}}, \sigma^*\right) = 1$ iff

- $\mathsf{AggVer}((m_j^*, \mathcal{P}_j^*)_{\forall j \in U}, (\mathsf{sk}_j)_{\forall j \in U \setminus \{1\}}, \sigma^*) = 1$ and
- $m_1^* \notin \mathcal{T}$, meaning the forgery is non-trivial, and
- one of the below two events happens:
  - (Type 1 Forgery) $\mathcal{P}_1^*$ is *not* well defined with respect to $\mathcal{T}$. This implies that the adversary has not seen MACs on all the inputs needed to compute $\mathcal{P}_1^*$.
  - (Type 2 Forgery) $\mathcal{P}_1^*$ is well defined with respect to $\mathcal{T}$ but $m_1^* \neq f_1^*\left((m_j)_{(\tau_{1,j}^*, m_j) \in \mathcal{T}}\right)$. This captures the case when m_1^* is not the correct output of the labeled program $\mathcal{P}_1^*$ when executed on previously authenticated messages $(m_j)_{\forall j \in [n]}$.

ALGORITHM Check_V For any input $(m^*, \mathcal{P}^*, \sigma^*)$, $\mathsf{Check}_V\,(m^*, \mathcal{P}^*, \sigma^*) = 1$ iff

- $\mathsf{Ver}_{\mathsf{sk}_1}\,(m^*, \mathcal{P}^*, \sigma^*) = 1$ and
- $m^* \notin \mathcal{T}$, meaning the forgery is non-trivial and
- one of the below two happens
  - (Type 1 Forgery) $\mathcal{P}^*$ is *not* well defined with respect to $\mathcal{T}$
  - (Type 2 Forgery) $\mathcal{P}^*$ is well defined with respect to $\mathcal{T}$ but $m^* \neq f^*\left((m_j)_{(\tau_{1,j}^*, m_j) \in \mathcal{T}}\right)$. This attests that m^* is not the correct output of the labeleld program $\mathcal{P}^*$ when executed on previously authenticated messages $(m_j)_{\forall j \in [n]}$.

DEFINITION We say that a multi-key fully homomorphic aggregate MAC (HA-MAC) scheme is secure if for any PPT adversary $\mathcal{A}$ there exists a negligible function ν such that

$$\mathbf{Adv}^{\text{HA}}_{\Theta,\mathcal{A}}(1^\lambda, 1^n, 1^D, 1^P) = \Pr[\mathbf{G}^{\text{HA-UF-CMA}}_{\Theta,\mathcal{A}}(1^\lambda, 1^n, 1^D, 1^P) \Rightarrow 1] \leq \nu(\lambda) \quad (4)$$

where the probability is over the random coins of $\mathcal{A}$ and KeyGen

Understanding $\mathbf{G}^{\text{HA-UF-CMA}}_{\Theta,\mathcal{A}}$: The game begins with the INITIALIZE procedure in which the challenger creates P (sk, ek) pairs. He gives all of these except the challenge party's sk_1 to $\mathcal{A}$. This allows $\mathcal{A}$ to produce arbitrary MACs for any party other than p_1.

Additionally, $\mathcal{A}$ can make queries to AUTHO to get tags under sk_1 for (m, τ) message label pairs of his choice. The challenger records all such queries in a table $\mathcal{T}$ and answers consistently on repeated queries or if multiple values are requested with the same label.

$\mathcal{A}$ can also make queries to VERO and AVERO on any messages, programs, and MACs. In particular, for the AVERO queries $\mathcal{A}$ can choose any subset of parties to aggregate as long as they include p_1. Computing aggregate MACs without p_1 is trivial since $\mathcal{A}$ has all the other sk's.

Finally, $\mathcal{A}$ outputs in the FINALIZE phase either an attempted forgery of the original MAC or an attempted aggregate forgery. He wins if the produced forgery couldn't be computed by simply running Eval on MACs he has already seen (i.e.,

the ones contained in the table $\mathcal{T}$). The Check_A and Check_V procedures perform this check for AVERO and VERO respectively. Thus, the adversary wins if he is able to produce a valid MAC or aggregate MAC that cannot be computed from the table $\mathcal{T}$.

5 Construction of Multi Key Fully Homomorphic Aggregate MAC for Arithmetic Circuits

We are now ready to present the details of our HA-MAC constructions. Our construction largely follows the construction of Catalano and Fiore [3] with the major differences, that we highlight, to enable aggregation. The details of the construction are given in Figure 2.

Setup*:* As described in Section 3.4, we assume the existence of groups $\mathcal{G}_1$ and $\mathcal{G}_2$ of order p with generators g_1 and g_2 respectively. With an efficiently computable isomorphism ψ from $\mathcal{G}_2$ to $\mathcal{G}_1$ such that $\psi(g_2) = g_1$ and a bilinear map $e : \mathcal{G}_1 \times \mathcal{G}_2 \to \mathcal{G}_T$ such that $e(g_1, g_2) \neq 1$ and the co-CDH problem in $\mathcal{G}_1$ and $\mathcal{G}_2$ is hard. We assume that all of these are public parameters that are available to all of the subsequent algorithms. We additionally assume the existence of a hash function $\mathcal{H}_2 : \{0,1\}^* \to \mathcal{G}_2$ which is modelled as random oracle [36].

KeyGen*:* For each party $l \in P$, we choose a public generator $u_l = g_1^{\gamma_l}$ for a random $\gamma_l \leftarrow\!\!\$\, \mathbb{Z}_p$. Additionally, we choose a random evaluation point $x_{l_0} \leftarrow\!\!\$\, \mathbb{Z}_p$ and a PRF key K_l. The secret key sk_l consists of (x_{l_0}, K_l, γ_l) while the public evaluation key ek_l consists of encodings of powers of x_{l_0} in the exponent. That is ek_l consists of $u_l, u_l^{x_{l_0}}, u_l^{x_{l_0}^2}, \ldots, u_l^{x_{l_0}^D}$ up to the specified depth bound D. We note that our construction differs from [3] in that we include the generator u_l in the public evaluation key, whereas their construction required keeping this value secret.

Auth*:* To authenticate a message m with label τ under user p_l's key, the tag σ is a degree-1 univariate polynomial y such that $y(0) = m$ and $y(x_{l_0}) = r_\tau$ where x_{l_0} is the secret evaluation point in sk_l and $r_\tau = PRF_k(\tau)$. We represent σ as a vector of the coefficients of y: (y_0, y_1).

Eval*:* To compute the homomorphic tag for an evaluation of a depth-D arithmetic circuit f and input tags $\sigma_1, \ldots, \sigma_n$ we follow the procedure described in Section 3.2 to evaluate f over the degree-1 polynomials in the σ's. That is, for every addition gate, we add the input polynomials and for every multiplication gate we do a polynomial multiplication corresponding to performing a convolution on the coefficients.

At the end of this computation, we get a degree-D polynomial y such that $y(0) = f(m_1, m_2, \ldots, m_n)$ and $y(x_0) = f(r_{\tau_1}, r_{\tau_2}, \ldots, r_{\tau_n})$[2]. We let y_k for $k \in \{0, \ldots, D\}$ be the coefficients of y.

[2] In what follows, for ease of presentation we drop the subscript l from all terms since we are only considering the case of player p_l.

Now to compute the homomorphic tag, we essentially evaluate $y(x_0)-y(0)$ in the exponent. Recall that ek contains $h_0 = u, h_1 = u^{x_0}, h_2 = u^{x_0^2}, \ldots, h_D = u^{x_0^D}$. Now, note that $y(x_0) = y_0 + y_1 x_0 + y_2 x_0^2 + \cdots + y_D x_0^D$ and $y(0) = y_0$. So, to evaluate $y(x_0) - y(0)$ in the exponent we compute the product

$$h_1^{y_1} \cdot h_2^{y_2} \cdots h_D^{y_D} = u^{y(x_0)-y(0)}.$$

Ver*:* On input the homomorphic tag Λ, program $\mathcal{P}$, and message m, we first compute $\rho = f(r_{\tau_1}, r_{\tau_2}, \ldots, r_{\tau_n}) = y(x_0)$ where f is the function from $\mathcal{P}$ and $r_{\tau_i} = PRF_K(\tau_i)$ and y is the polynomial computed in Eval. Now recall that for a valid tag $\Lambda = u^{y(x_0)-y(0)}$ where $y(0) = m$, the homomorphic output. So, to verify the homomorphic tag Λ, we just need to check whether $\Lambda \cdot u^m = u^\rho$. The case for verifying an original (i.e., non-homomorphic) tag is similar.

For reasons needed in the proof we deviate from [3]. Meaning, instead we do this verification in the target group $\mathcal{G}_T$ by pairing all terms in this equation with $w \leftarrow \mathcal{H}_2(m)$. Roughly, this is necessary to allow the security reduction to program in the challenge from its challenger. A summary of the proof is given in Section 6.

Aggregate*:* To aggregate homomorphic tags from a set of U users, we take a weighted sum of the tags in the exponent. Specifically, for each message m_i to aggregate, we compute $w_i \leftarrow \mathcal{H}_2(m_i)$. Viewing w_i as $g_2^{b_i}$ for random exponent b_i, we multiply Λ_i by b_i in the exponent by computing $e(\Lambda_i, w_i)$. Finally we multiply the resulting elements in $\mathcal{G}_T$ together to compute a random linear combination of the component tags.

AggVer*:* On input an aggregate mac on any subset of U users, its corresponding messages and evaluation keys, AggVer algorithm uses its secret keys for the users in set U to recompute the aggregate MAC and checks whether it is equal to the input aggregate MAC. It accepts iff the equality holds. Note that since ρ is computed by evaluating f at random points, aggregate verification does not require the verifier to know the original inputs, only the outputs of the homomorphic computations.

5.1 Correctness

We now prove that the construction in Figure 2 satisfies the necessary correctness properties of HA-MAC.

Authentication Correctness. For any $l \in P$, recall that $\rho_l \leftarrow y_0 + y_1 \cdot x_{l_0}$. Let $\gamma_l \leftarrow_\$ \mathbb{Z}_p, b_l \leftarrow_\$ \mathbb{Z}_p$ such that $u_l \leftarrow g_1^{\gamma_l}$ and $w_l \leftarrow g_2^{b_l}$. Therefore we get that:

$$\mathrm{e}(u_l^{x_{l_0} y_1}, w_l) \cdot e(u_l, w_l^{y_0}) = \mathrm{e}(g_1, g_2)^{\gamma_l \cdot b_l (y_0 + x_{l_0} \cdot y_1)} = \mathrm{e}(u_l, w_l)^{\rho_l} \tag{5}$$

Evaluation Correctness. For any $l \in P$, recall that $\rho_l \leftarrow m_l + \Sigma_{k=1}^{D} y_{l,k} \cdot x_{l_0}^k$.

$\underline{\mathsf{KeyGen}(1^\lambda, 1^n, 1^D, 1^P)}$:

$p \leftarrow\!\!\$\ \mathsf{PrimeGen}(1^\lambda)$
$\forall l \in [P]$:
  $x_{l_0}, \gamma_l \leftarrow\!\!\$\ \mathbb{Z}_p^2$, $K_l \leftarrow\!\!\$\ \mathcal{K}$, $u_l \leftarrow g_1^{\gamma_l}$
  $\forall k \in [0, D], h_{l,k} \leftarrow u_l^{x_{l_0}^k}$
  $\mathsf{sk}_l \leftarrow (x_{l_0}, K_l, \gamma_l)$
  $\mathsf{ek}_l \leftarrow (h_{l,0}, h_{l,1}, h_{l,2}, \ldots h_{l,D})$
Return $(\mathsf{ek}_l, \mathsf{sk}_l)_{\forall l \in [P]}$

$\underline{\mathsf{Auth}(\mathsf{sk}, (\tau, m))}$:

$(x_0, K, \gamma) \leftarrow \mathsf{sk}$
$r_\tau \leftarrow PRF_K(\tau)$
$y_0 \leftarrow m; y_1 \leftarrow \frac{r_\tau - m}{x_0} \mod p$
Return $\sigma \leftarrow (y_0, y_1)$

$\underline{\mathsf{Ver}(\mathsf{sk}, m, \mathcal{P}, \Lambda)}$:

$(f, \tau_1, \ldots, \tau_n) \leftarrow \mathcal{P}$
$(x_0, K, \gamma) \leftarrow \mathsf{sk}$, $w \leftarrow \mathcal{H}_2(m)$, $u \leftarrow g_1^\gamma$
$\mathbf{r} \leftarrow PRF_K(\tau_i)_{\forall i \in [n]}$, $\rho \leftarrow f(\mathbf{r})$
If $\Lambda = (y_0, y_1)$ then
return $\mathrm{e}(u^{x_0 \cdot y_1}, w) \cdot \mathrm{e}(u, w^{y_0}) \stackrel{?}{=} \mathrm{e}(u, w^\rho)$
Else Return $\mathrm{e}(u, w^m) \cdot e(\Lambda, w) \stackrel{?}{=} \mathrm{e}(u, w^\rho)$

$\underline{\mathsf{Eval}(\mathsf{ek}, f, (\sigma_i)_{\forall i \in [n]})}$:

$(y_k)_{k \in [0,D]} \leftarrow f((\sigma_i)_{\forall i \in [n]})$
$\Lambda \leftarrow \Pi_{k=1}^D h_k^{y_k}$
Return Λ

$\underline{\mathsf{Aggregate}((\Lambda_l, m_l)_{\forall l \in U})}$:

$w_l \leftarrow \mathcal{H}_2(m_l)$
$\sigma^* \leftarrow \Pi_{\forall l \in U} \mathrm{e}(\Lambda_l, w_l)$
Return σ^*

$\underline{\mathsf{AggVer}((\mathsf{sk}_l, \mathcal{P}_l, m_l)_{\forall l \in U}, \sigma^*)}$:

$\forall l \in U : (f_l, \tau_{l,i})_{\forall i \in [n]} \leftarrow \mathcal{P}_l$, $(x_{l_0}, K_l, \gamma_l) \leftarrow \mathsf{sk}_l$
  $w_l \leftarrow \mathcal{H}_2(m_l), u_l \leftarrow g_1^{\gamma_l}$
  $\mathbf{r}_l \leftarrow PRF_{K_l}(\tau_{l,i})_{\forall l \in U, \forall i \in [n]}$, $\rho_l \leftarrow f_l(\mathbf{r}_l)$
Return $\Pi_{\forall l \in U} \mathrm{e}(u_l, w_l^{\rho_l}) \stackrel{?}{=} \sigma^* \cdot \Pi_{\forall l \in U} \mathrm{e}(u_l, w_l^{m_l})$

Fig. 2: The construction of HA-MAC Θ for bounded depth arithmetic circuits.

Therefore we get that

$$\mathrm{e}(\Lambda_l, w_l) \cdot e(u_l, w_l)^{m_l} = e(g_1^{\gamma_l \cdot \Sigma_{k=1}^D y_{l,k} \cdot x_{l_0}^k}, g_2^{b_l}) \cdot \mathrm{e}(g_1^{\gamma_l}, g_2^{b_l})^{m_l} = e(u_l, w_l)^{\rho_l} \quad (6)$$

Aggregation Correctness. For any subset of users U such that $U \subseteq P$ then for all l in U, let $b_l \leftarrow\!\!\$\ \mathbb{Z}_p$, $\gamma_l \leftarrow\!\!\$\ \mathbb{Z}_p$, hash of message m_l is $w_l \leftarrow g_2^{b_l}$ and generator $u_l \leftarrow g_1^{\gamma_l}$ then we get that:

$$\begin{aligned}
&\sigma^* \cdot (\Pi_{\forall l \in U}\ \mathrm{e}(u_l, w_l^{m_l})) \\
&= \mathrm{e}(g_1, g_2)^{\Sigma_{\forall l \in U}\left(\gamma_l \cdot b_l \cdot (\Sigma_{k=1}^D y_{l,k} \cdot x_{l_0}^k)\right)} \cdot \mathrm{e}(g_1, g_2)^{\Sigma_{\forall l \in U}(\gamma_l \cdot b_l \cdot m_l)} \\
&= \mathrm{e}(g_1, g_2)^{\Sigma_{\forall l \in U}\left(\gamma_l \cdot b_l \cdot \left((\Sigma_{k=1}^D y_{l,k} \cdot x_{l_0}^k) + m_l\right)\right)} = \Pi_{\forall l \in U}\ \mathrm{e}(u_l, w_l)^{\rho_l}
\end{aligned} \quad (7)$$

5.2 Performance and Security

Finally, our HA-MAC scheme achieves the following performance and security:

Efficiency: We note that the size our homomorphic tag and aggregate tag is *succinct* consisting of just one field element independent of the number of parties or complexity of the computed functions. Moreover, the original inputs into the homomorphic computation are not needed to verify the final aggregate signature so they do not need to be communicated.

Security: We say that a P-user HA-MAC scheme is $(t, Q_H, Q_A, Q_V, Q_{AV}, P, \epsilon)$-secure in the random oracle model if for any PPT adversary $\mathcal{A}(t, Q_H, Q_A, Q_V, Q_{AV}, P, \epsilon)$ in the $\mathbf{G}^{\text{HA-UF-CMA}}_{\Theta,\mathcal{A}}$ game (figure 1)

- $\mathcal{A}$ runs in time *at most* t,
- $\mathcal{A}$ makes *at most* Q_H queries to the hash function,
- $\mathcal{A}$ makes *at most* Q_A queries to the authenticate oracle,
- $\mathcal{A}$ makes *at most* Q_V queries to the verification oracle, and
- $\mathcal{A}$ makes *at most* Q_{AV} queries to the aggregate verification oracle then
- $\Pr[\mathbf{G}^{\text{HA-UF-CMA}}_{\Theta,\mathcal{A}} = 1] \leq \epsilon$

where the probability is over the random coins of $\mathcal{A}$ and KeyGen.

We can now formally state the security of our construction.

Theorem 1. *If co-CDH is (t', ϵ') hard over groups $(\mathcal{G}_1, \mathcal{G}_2)$ and PRF is ϵ'' secure then HA-MAC scheme as defined in figure 2 is $(t, Q_H, Q_A, Q_V, Q_{AV}, P, \epsilon)$-secure in the random oracle model for all t, ϵ satisfying*

$$\epsilon < \frac{Q}{2^\lambda} + \frac{DQ}{p - DQ} + \frac{DQ^2}{p} + e^{\frac{3}{P}} PQ \left(e^{\frac{1}{Q}} + 1\right) \epsilon' + \epsilon''$$

and

$$t > t' - c(s + P + Q + PQ + sQ + sPQ)$$

where Q is the maximum of Q_H, Q_A, Q_V, Q_{AV}, c is the maximum time to compute any group operation, s is the size of the depth D function f as in definition 1 and P is the total number of users.

We sketch the intuition behind the proof of this theorem in Section 6. We provide the full proof of security in the supplementary material B.

6 Proof of Security:

To prove the security of our construction, we define a series of games as depicted in figures 3, 4, 5, 6, 7 and 8 in the supplementary material B. We briefly review these games and the intuition behind the proof here, the full proof is in supplementary material B.

Game 0: Our starting point is the real-world security game $\mathbf{G}^{\text{HA-UF-CMA}}_{\Theta,\mathcal{A}}$(Figure 1). Game $\mathbf{G}^{0}_{\Theta,\mathcal{A}}$ (Figure 3) is the same as the real game except for two changes. In every VERO query $(m, \mathcal{P}, \Lambda)$, the challenger uses the probabilistic test of Catalano and Fiore [3](recalled in Proposition 1) to test whether $\mathcal{P}$ is well defined with respect to table $\mathcal{T}$. Similarly, we also use this same test in every AVERO query to check whether $\mathcal{P}'_1$ (player p_1's program) is well defined. Since the test in Proposition 1 is correct with all but negligible probability, this only introduces a negligible difference between the games.

Game 1: Game $\mathbf{G}^{1}_{\Theta,\mathcal{A}}$ (Figure 4) is the same as $\mathbf{G}^{0}_{\Theta,\mathcal{A}}$ except that the PRF in the Auth oracle is replaced by a trully random function(TRF) $\mathcal{R} : \{0,1\}^* \to \mathbb{Z}_p$. Clearly this change is undetectable by the security of the PRF.

Game 2: Game $\mathbf{G}^{2}_{\Theta,\mathcal{A}}$ (Figure 5) is the same as $\mathbf{G}^{1}_{\Theta,\mathcal{A}}$ except the challenger adjusts how it responds to VERO and AVERO queries.

- VERO Queries: For any VERO query $(m, \mathcal{P}, \Lambda)$ such that $\mathcal{P}$ is not well defined in $\mathcal{T}$, the challenger returns reject. For any well-defined $\mathcal{P}$, the challenger acts exactly as in $\mathbf{G}^1$.
  Thus, the only difference between $\mathbf{G}^2$ and $\mathbf{G}^1$ occurs if a Mac for a not well-defined program is accepted in $\mathbf{G}^1$. We argue that for any adversary $\mathcal{A}$ such a *bad* query only occurs with negligible probability. Roughly, making such a query requires finding the correct value of ρ. However, finding ρ requires guessing r_τ for some τ not in $\mathcal{T}$. Since $r_\tau = \mathcal{R}(\tau)$ for a random function $\mathcal{R}$, this can only happen negligibly often.
- AVERO Queries: Similarly, $\mathbf{G}^1$ and $\mathbf{G}^2$ differ if $\mathcal{A}$ can find an aggregate Mac σ' such that the circuit for player p_1 is not well-defined relative to $\mathcal{T}$, but is still accepted in $\mathbf{G}^1$. Here to prove that the probability of $\mathcal{A}$ finding such a σ is negligible, we extract p_1's component of the aggregate (using the sk's of the other parties, which the challenger knows), and argue as above that he must have predicted a value r_τ. The full details of the proof are given in Lemma 3.

Game 3: Game $\mathbf{G}^{3}_{\Theta,\mathcal{A}}$(figure 6) is the same as $\mathbf{G}^{2}_{\Theta,\mathcal{A}}$ except the following change in answering Auth queries. If the random value for some tag τ queried in Auth has previously been used to answer a VERO or AVERO query, then just resample a new, independent value $r_\tau = \mathcal{R}(\tau)$. Since $\mathcal{A}$ can only learn a polynomial number of possible r_τ points, the probability that one of these is sampled is negligible. If this doesn't happen this game is the same as the previous one. The full details are in Lemma 4.

Game 4: Game $\mathbf{G}^{4}_{\Theta,\mathcal{A}}$ (figure 7) differs from $\mathbf{G}^{3}_{\Theta,\mathcal{A}}$ in how the challenger answers VERO queries for programs that are well-defined relative to $\mathcal{T}$. Specifically, for a query $(m, \mathcal{P}, \Lambda)$[3], for every input tag τ_i in $\mathcal{P}$, the challenger finds the corre-

[3] $\mathbf{G}^4$ also handles the case of verification of initial Macs in addition to homomorphically computed Macs, but we omit this case from our discussion here to simplify presentation.

sponding input Mac σ_i and message m_i in $\mathcal{T}$. (Since $\mathcal{P}$ is well-defined relative to $\mathcal{T}$, only wires that have no impact on the output may not have an entry in $\mathcal{T}$, in which case the challenger can just choose a random tag.) Next the challenger uses these tags σ_i to recompute the homomorphic Mac Λ^{***} using Eval and checks if it is equal to the claimed Mac Λ queried by the adversary. The challenger returns accept if and only if they are equal.

This differs from $\mathbf{G}^3$ in that it eliminates the possibility of Type-2 forgeries in VERO queries. In a Type-2 forgery, the adversary somehow produces a VERO query for a well-defined program where the message used for one of the inputs is not equal to the one in $\mathcal{T}$. In this case the recomputed Λ^{***} will not be correct, but the query will still accept in $\mathbf{G}^3$.

We prove that an adversary can only make such a query with non-negligible probability. To do so, we introduce one more change into $\mathbf{G}^4$ in that we change how the challenger answers hash evaluation queries for $\mathcal{H}_2$. Specifically, instead of just returning a random element in $\mathcal{G}_2$, the challenger now samples a random exponent b and returns g_2^b as the random element in $\mathcal{G}_2$ while storing b. This knowledge of the discrete log b, allows the challenger to detect when an adversary makes such a Type-2 forgery query.

Finally, we argue that such a query is negligibly likely by showing a reduction from an adversary making such a query to solving the co-CDH problem in $(\mathcal{G}_1, \mathcal{G}_2)$. To do so, intuitively the challenger roughly does the following. The challenger randomly picks one of the VERO queries and programs the random oracle to return the co-CDH challenge g_2^a as the output of the corresponding $\mathcal{H}_2$ query. We can show that by embedding the other part of the co-CDH challenge (the value $h \in \mathcal{G}_1$) in the evaluation key ek_1, a Type-2 forgery query must allow extraction of the value h^a, thus solving co-CDH. The full proof is given in Lemmas 5–7.

Game 5: Game $\mathbf{G}^5_{\Theta,\mathcal{A}}$ (figure 8) makes a similar change to the one in $\mathbf{G}^4$, but to the AVERO queries. Here, again, the goal is to eliminate Type-2 forgeries in AVERO queries. The added challenge here is that we have to deal with the fact that some (in fact, all but one) of the macs included in an aggregate mac come from malicious parties. We show that the challenger is able to extract the claimed homomorphic Mac for party p_1, and can then compare this to the Mac recomputed from the table $\mathcal{T}$, accepting if and only if they match. Effectively, the challenger can recompute the random linear combination of the adversaries' macs by using $\mathcal{H}_2$ to recompute the weights (in the exponent) for this sum.

With this check in place, the only way that $\mathbf{G}^5$ and $\mathbf{G}^4$ differ is if the adversary queries an aggregate Mac that contains a Type-2 forgery for party 1. However, by a reduction similar to the one described in the previous game, we can show that any adversary that can make such a query must also solve the co-CDH problem on $(\mathcal{G}_1, \mathcal{G}_2)$. The full proof is given in Lemmas 8–10.

Unforgeability: To conclude the proof of unforgeability, we observe that $\Pr[\mathbf{G}^5_{\Theta,\mathcal{A}}] = 0$ because all verification queries and aggregate verification queries for both Type-1 and Type-2 forgeries are answered with 0. Thus, there is no

opportunity for any adversary to win in Game 5, and so for any adversary $\mathcal{A}$ its advantage in $\mathbf{G}^5$ is 0.

7 Conclusion

In this paper, we introduced the concept of a multi-key fully-homomorphic aggregate MAC (HA-MAC) for arithmetic circuits. This primitive enables an untrusted server to produce a short certificate to prove that he has performed correct (disjoint) computations on multiple users' data. The size of this proof is independent of the number of users or the complexity of the performed computations. We give a construction of this primitive based on the co-CDH assumption in the random oracle model.

Our paper leaves open a number of interesting problems for further study. Two immediate questions are removing the reliance on the random oracle and improving the computational efficiency of verification. While the final Mac in our construction is succinct – only 1 field element – verification still requires the verifier to evaluate all functions on random inputs. Very recent work by Anthoine et al. [7] showed how to amortize verification costs for multiple verifications of the same computation in a similar setting. It would be interesting to apply similar techniques to amortize verification costs for our HA-MAC. Further possible improvements to our construction include allowing verification given only an aggregate of the homomorphic outputs rather than requiring all of the outputs to verify an aggregate Mac and eliminating the bounded-degree requirement of our assumption. I.e., can we construct a scheme where the size of the keys required does not grow with the depth of the homomorphic evaluations supported.

Acknowledgements

We would like to thank Adam O'Neill, Ojaswi Acharya, Weiqi Feng for valuable discussions that led to the problem studied here. Arkady Yerukhimovich is supported in part by NSF grants CNS-1955620 and CNS-2144798 (CAREER).

References

1. R. Johnson, D. Molnar, D. X. Song, and D. A. Wagner, "Homomorphic signature schemes," in *Topics in Cryptology - CT-RSA 2002, The Cryptographer's Track at the RSA Conference, 2002, San Jose, CA, USA, February 18-22, 2002, Proceedings*, ser. Lecture Notes in Computer Science, B. Preneel, Ed., vol. 2271. Springer, 2002, pp. 244–262. [Online]. Available: https://doi.org/10.1007/3-540-45760-7_17
2. D. Boneh and D. M. Freeman, "Homomorphic signatures for polynomial functions," in *Advances in Cryptology – EUROCRYPT 2011*, ser. Lecture Notes in Computer Science, K. G. Paterson, Ed., vol. 6632. Tallinn, Estonia: Springer, Heidelberg, Germany, May 15–19, 2011, pp. 149–168.
3. D. Catalano and D. Fiore, "Practical homomorphic macs for arithmetic circuits," in *Advances in Cryptology–EUROCRYPT 2013: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings 32*. Springer, 2013, pp. 336–352.
4. R. Gennaro and D. Wichs, "Fully homomorphic message authenticators," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2013, pp. 301–320.
5. S. Gorbunov, V. Vaikuntanathan, and D. Wichs, "Leveled fully homomorphic signatures from standard lattices," in *47th Annual ACM Symposium on Theory of Computing*, R. A. Servedio and R. Rubinfeld, Eds. Portland, OR, USA: ACM Press, Jun. 14–17, 2015, pp. 469–477.
6. D. Fiore, A. Mitrokotsa, L. Nizzardo, and E. Pagnin, "Multi-key homomorphic authenticators," in *Advances in Cryptology–ASIACRYPT 2016: 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II*. Springer, 2016, pp. 499–530.
7. G. Anthoine, D. Balbás, and D. Fiore, "Fully-succinct multi-key homomorphic signatures from standard assumptions," in *Annual International Cryptology Conference*. Springer, 2024, pp. 317–351.
8. D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "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 22*. Springer, 2003, pp. 416–432.
9. M. Bellare, C. Namprempre, and G. Neven, "Unrestricted aggregate signatures," in *Automata, Languages and Programming: 34th International Colloquium, ICALP 2007, Wrocław, Poland, July 9-13, 2007. Proceedings 34*. Springer, 2007, pp. 411–422.
10. J. Katz and A. Y. Lindell, "Aggregate message authentication codes," in *Cryptographers' Track at the RSA Conference*. Springer, 2008, pp. 155–169.
11. S. Gorbunov and H. Wee, "Digital signatures for consensus," *Cryptology ePrint Archive*, 2019.
12. R. Rivest, "Two new signature schemes, 2001," in *Cambridge seminar*.
13. Y. Desmedt, "Computer security by redefining what a computer is," in *Proceedings on the 1992-1993 Workshop on New Security Paradigms, Sept. 22-24, 1992; and Aug. 3-5, 1993, Little Compton, RI, USA*, J. B. Michael, V. Ashby, and C. Meadows, Eds. ACM, 1993, pp. 160–166. [Online]. Available: https://doi.org/10.1145/283751.283834

14. M. Rückert and D. Schröder, "Aggregate and verifiably encrypted signatures from multilinear maps without random oracles," in *International Conference on Information Security and Assurance*. Springer, 2009, pp. 750–759.
15. B. Waters and D. J. Wu, "Batch arguments for np and more from standard bilinear group assumptions," in *Annual International Cryptology Conference*. Springer, 2022, pp. 433–463.
16. J. Kilian, "A note on efficient zero-knowledge proofs and arguments (extended abstract)," in *24th Annual ACM Symposium on Theory of Computing*. Victoria, BC, Canada: ACM Press, May 4–6, 1992, pp. 723–732.
17. S. Micali, "CS proofs (extended abstracts)," in *35th Annual Symposium on Foundations of Computer Science*. Santa Fe, NM, USA: IEEE Computer Society Press, Nov. 20–22, 1994, pp. 436–453.
18. S. Goldwasser, Y. T. Kalai, and G. N. Rothblum, "Delegating computation: interactive proofs for muggles," in *40th Annual ACM Symposium on Theory of Computing*, R. E. Ladner and C. Dwork, Eds. Victoria, BC, Canada: ACM Press, May 17–20, 2008, pp. 113–122.
19. R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," in *Advances in Cryptology – CRYPTO 2010*, ser. Lecture Notes in Computer Science, T. Rabin, Ed., vol. 6223. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 15–19, 2010, pp. 465–482.
20. K.-M. Chung, Y. Kalai, and S. Vadhan, "Improved delegation of computation using fully homomorphic encryption," in *Advances in Cryptology–CRYPTO 2010: 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings 30*. Springer, 2010, pp. 483–501.
21. B. Applebaum, Y. Ishai, and E. Kushilevitz, "From secrecy to soundness: Efficient verification via secure computation," in *ICALP 2010: 37th International Colloquium on Automata, Languages and Programming, Part I*, ser. Lecture Notes in Computer Science, S. Abramsky, C. Gavoille, C. Kirchner, F. Meyer auf der Heide, and P. G. Spirakis, Eds., vol. 6198. Bordeaux, France: Springer, Heidelberg, Germany, Jul. 6–10, 2010, pp. 152–163.
22. S. Benabbas, R. Gennaro, and Y. Vahlis, "Verifiable delegation of computation over large datasets," in *Advances in Cryptology – CRYPTO 2011*, ser. Lecture Notes in Computer Science, P. Rogaway, Ed., vol. 6841. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 14–18, 2011, pp. 111–131.
23. B. Parno, M. Raykova, and V. Vaikuntanathan, "How to delegate and verify in public: Verifiable computation from attribute-based encryption," in *Theory of Cryptography: 9th Theory of Cryptography Conference, TCC 2012, Taormina, Sicily, Italy, March 19-21, 2012. Proceedings 9*. Springer, 2012, pp. 422–439.
24. D. Fiore and R. Gennaro, "Publicly verifiable delegation of large polynomials and matrix computations, with applications," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 501–512.
25. N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, "From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again," in *ITCS 2012: 3rd Innovations in Theoretical Computer Science*, S. Goldwasser, Ed. Cambridge, MA, USA: Association for Computing Machinery, Jan. 8–10, 2012, pp. 326–349.
26. B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," *Communications of the ACM*, vol. 59, no. 2, pp. 103–112, 2016.

27. R. W. Lai, R. K. Tai, H. W. Wong, and S. S. Chow, "Multi-key homomorphic signatures unforgeable under insider corruption," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2018, pp. 465–492.
28. R. Gennaro, C. Gentry, B. Parno, and M. Raykova, "Quadratic span programs and succinct nizks without pcps," in *Advances in Cryptology–EUROCRYPT 2013: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings 32*. Springer, 2013, pp. 626–645.
29. C. Gentry and D. Wichs, "Separating succinct non-interactive arguments from all falsifiable assumptions," in *Proceedings of the forty-third annual ACM symposium on Theory of computing*, 2011, pp. 99–108.
30. J. Katz and Y. Lindell, *Introduction to modern cryptography: principles and protocols*. Chapman and hall/CRC, 2007.
31. M. Agrawal and R. SaptharishiI, "Classifying polynomials and identity testing." Indian Academy of Sciences, 2009.
32. P. Dutta, P. Dwivedi, and N. Saxena, "Demystifying the border of depth-3 algebraic circuits," in *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2022, pp. 92–103.
33. R. A. DeMillo and R. J. Lipton, "A probabilistic remark on algebraic program testing," *Information processing letters*, vol. 7, no. 4, pp. 193–195, 1978.
34. R. Zippel, "Probabilistic algorithms for sparse polynomials," in *International symposium on symbolic and algebraic manipulation*. Springer, 1979, pp. 216–226.
35. D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," in *International conference on the theory and application of cryptology and information security*. Springer, 2001, pp. 514–532.
36. M. Bellare and P. Rogaway, "The exact security of digital signatures-how to sign with rsa and rabin," in *International conference on the theory and applications of cryptographic techniques*. Springer, 1996, pp. 399–416.
37. ——, "The security of triple encryption and a framework for code-based game-playing proofs," in *Advances in Cryptology – EUROCRYPT 2006*, ser. Lecture Notes in Computer Science, S. Vaudenay, Ed., vol. 4004. St. Petersburg, Russia: Springer, Heidelberg, Germany, May 28 – Jun. 1, 2006, pp. 409–426.
38. D. Boneh and V. Shoup, "A graduate course in applied cryptography," *Draft 0.5*, 2020.

A Deferred Notations

If S is a finite set, then $|S|$ denotes its size. If X is a finite set, we let $x \leftarrow\!\!\$\ X$ denote picking an element of X uniformly at random and assigning it to x. By $1^{|X|}$ we refer to the unary representation of the size of the finite set. By $X \setminus \{i\}$ we mean set X minus the element i of the set. $X = X \bigcup y$ refers to the operation of element y being included in the set X. Algorithms may be randomized unless otherwise indicated. Running time is the worst case, which for an algorithm with access to oracles means across all possible replies from the oracles. We use $\perp$ (bot) as a special symbol to denote rejection, and it is assumed to not be in $\{0,1\}^*$. With reference to the proofs of the lemmas, we note that the superscript notation of m^i refers to the message in i^{th} query to a respective oracle. Whereas in each query, we further note that the subscript notation of m_i for any message $m \in \mathcal{M}$ refers to the message authenticated under key pair $\mathsf{sk}_i, \mathsf{ek}_i$.

GAMES We use the code-based game-playing framework of BR [37]. By $\Pr[\mathrm{G} \Rightarrow y]$ we denote the probability that the execution of game $\mathbf{G}$ results in this output being y, and write just $\Pr[\mathbf{G}]$ for the probability that the execution of game $\mathbf{G}$ results in the output of the execution being the boolean true.

Different games may have procedures (oracles) with the same names. In games, integer variables, set variables, boolean variables, and string variables are assumed initialized, respectively, to 0, the empty set $\emptyset$, the boolean false, and the $\perp$, flags with 0 or 1. For an adversary $\mathcal{A}$ playing game $\mathbf{G}$, we may write another adversary C in the same format as $\mathbf{G}$, with the understanding that C runs this game with $\mathcal{A}$. These games are in the RO Model, with RO being the random oracle available to both schemes algorithms and the adversary $\mathcal{A}$ playing the game. The adversary's queries to an oracle are required to satisfy the conditions listed in the comment next to the oracle name, else the adversary is considered invalid. We henceforth only consider *valid adversaries*.

PROOF We adhere to asymptotic security analysis[38]. A function ν: $\mathbb{N} \to \mathbb{N}$ is *negligible* if for every positive polynomial p: $\mathbb{N} \to \mathbb{R}$ there is a $\lambda_p \in \mathbb{N}$ such that $\nu(\lambda) \leq 1/p(\lambda)$ for all $\lambda \geq \lambda_p$. "PT" stands for "polynomial time," whether for randomized or deterministic algorithms. Specifically "PPT" stands for "probabilistic polynomial time. By 1^λ we denote the unary representation of the integer security parameter $\lambda \in \mathbb{N}$.

Additionally we also analyse concrete security[30]. A concrete approach to computational security quantifies the security of a cryptographic scheme by explicitly bounding the maximum success probability of any randomized adversary running for some specific amount of time. Meaning a scheme is (t, ϵ)-secure if any adversary running for time *at most* t succeeds in breaking the scheme with probability *at most* ϵ for some $t, \epsilon \in \mathbb{N}$.

B Deferred Proofs

To prove the security of our construction, we define a series of games as depicted in Figures 3, 4, 5, 6, 7 and 8. Moreover the construction of two associated reduction adversaries can be found in Figures 9, 10.

For any PPT adversary $\mathcal{A}$ against our HA-MAC scheme Θ, we denote as d as the answer the split out in any Ver or AggVer queries throughout the below games. We capture changes in every subsequent hybrid in *magenta*. We denote with $\mathbf{G}^{\mathrm{i}}$ the event that the i^{th} game $\mathbf{G}^{\mathrm{i}}_{\mathcal{A},\Theta}$ when run with the adversary $\mathcal{A}$ outputs 1. We recall below theorem 1 here for completeness.

Theorem 1. *If co-CDH is (t', ϵ') hard over groups $(\mathcal{G}_1, \mathcal{G}_2)$ and PRF is ϵ'' secure then HA-MAC scheme as defined in Figure 2 is $(t, Q_H, Q_A, Q_V, Q_{AV}, P, \epsilon)$-secure in the random oracle model for all t, ϵ satisfying*

$$\epsilon < \frac{Q}{2^\lambda} + \frac{DQ}{p - DQ} + \frac{DQ^2}{p} + e^{\frac{3}{P}} PQ \left(e^{\frac{1}{Q}} + 1 \right) \epsilon' + \epsilon''$$

and

$$t > t' - c(s + P + Q + PQ + sQ + sPQ)$$

where Q is the maximum of Q_H, Q_A, Q_V, Q_{AV}, c is the maximum time to compute any group operation, s is the size of the depth D function f as in definition 1 and P is the total number of users.

In the remainder of the Section we prove Theorem 1 through a hybrid argument.

Lemma 1. $|Pr[\mathbf{G}^{\text{HA-UF-CMA}}_{\Theta,\mathcal{A}}] - Pr[\mathbf{G}^{0}_{\Theta,\mathcal{A}}]| \leq \frac{Q_V + Q_{AV}}{2^\lambda}$ *where λ is security parameter and Q_V, Q_{AV} is the number of* VERO *queries and* AVERO *Queries made by the adversary $\mathcal{A}$.*

Proof. Note that the only difference between $\mathbf{G}^{\text{HA-UF-CMA}}_{\Theta,\mathcal{A}}$ and $\mathbf{G}^{0}_{\Theta,\mathcal{A}}$ is that in $\mathbf{G}^0$ we use Proposition 1 to check whether programs in VERO and AVERO queries are well formed. By proposition 1, the distinguishing advantage of $\mathcal{A}$ from Ver of the protocol and that of $\mathbf{G}^{0}_{\Theta,\mathcal{A}}$ is upper bounded by $\frac{Q_V}{2^\lambda}$ and that of AggVer of the protocol and that of $\mathbf{G}^{0}_{\Theta,\mathcal{A}}$ is upper bounded by $\frac{Q_{AV}}{2^\lambda}$. Therefore $\mathcal{A}$ distinguishes between real protocol and $\mathbf{G}^{0}_{\Theta,\mathcal{A}}$ with probability at most $\frac{Q_V}{2^\lambda} + \frac{Q_{AV}}{2^\lambda}$ where the probability is taken over the random values for the input labels. □

Lemma 2. $|\Pr[\mathbf{G}^{0}_{\Theta,\mathcal{A}}] - \Pr[\mathbf{G}^{1}_{\Theta,\mathcal{A}}]| \leq \mathbf{Adv}^{PRF}_{\mathrm{D},\mathcal{F}}(\lambda)$ *where* D *is a PRF adversary and $\mathcal{F}$ is the family of functions $\{f | f : \mathbb{F}^n \to \mathbb{F}\}$.*

Proof. This proof can be obtained as a straightforward reduction to the security of the PRF. □

Lemma 3. $|\Pr[\mathbf{G}^{1}_{\Theta,\mathcal{A}}] - \Pr[\mathbf{G}^{2}_{\Theta,\mathcal{A}}]| \leq \frac{Q_V(D+1)}{p - D(Q_V - 1)} + \frac{DQ_{AV}}{p - D(Q_{AV} - 1)}$ *where p is the order of the group, D is the max depth of the circuit and Q_V, Q_{AV} are the number of* VERO *and* AVERO *queries respectively made by the adversary $\mathcal{A}$.*

Proof. Let η be the event that flag $\zeta \leftarrow 1$ and let η_A be the event that flag $\zeta_A \leftarrow 1$ in $\mathbf{G}^2_{\Theta,\mathcal{A}}$. Let η_2 be the event that either event η or event η_A has occurred in $\mathbf{G}^2_{\Theta,\mathcal{A}}$ (Figure 5). $\mathbf{G}^1_{\Theta,\mathcal{A}}$ and $\mathbf{G}^2_{\Theta,\mathcal{A}}$ are identical unless the event η_2 occurs. Meaning the only change is in answering VERO queries where $\mathcal{P}$ is not well defined on $\mathcal{T}$ and in answering AVERO queries where $\mathcal{P}'_1$ is not well defined on $\mathcal{T}$.

We recall the following notation that challenge evaluation key is $\mathsf{ek}_1 \leftarrow (u_1^{x_1^k})_{\forall k \in [0,D]}$ such that the x_1 in the challenge secret key sk_1. We also recall the notation of hash $\mathcal{H}_2$ of any message m as $w \leftarrow g_2^b$ for some $b \leftarrow\!\!\$\, \mathbb{Z}_p$. In case of event η or η_A the challenger provides a different answer to some VERO or AVERO queries respectively. It holds that $\Pr[\mathbf{G}^2_{\Theta,\mathcal{A}} \wedge \neg\eta_2] = \Pr[\mathbf{G}^1_{\Theta,\mathcal{A}}]$, meaning $|\Pr[\mathbf{G}^1_{\Theta,\mathcal{A}}] - \Pr[\mathbf{G}^2_{\Theta,\mathcal{A}}]| \leq \Pr[\eta_2]$. By definition

$$\Pr[\eta_2] \leq \Pr[\eta] + \Pr[\eta_A] \tag{8}$$

VERIFICATION QUERY

For $j \in [Q_V]$, let ξ_j be the event that flag ζ is assigned the value 1 only after the j^{th} verification query and not before that. Clearly, by union bound we have:

$$\Pr[\eta] \leq \sum_{j=1}^{[Q_V]} \Pr[\xi_j] \tag{9}$$

Moreover, by the definition of ξ_j, event $\zeta \leftarrow 1$ did not occur in the previous $j-1$ VERO queries. Therefore we have

$$\Pr[\xi_j] = \Pr[\xi_j \mid \neg\xi_1 \wedge \cdots \wedge \neg\xi_{j-1}] \tag{10}$$

The main part of this proof consists of estimating the probability $\Pr[\xi_j]$, which is taken over the random choices of the values r^{**}_τ sampled by the challenger and for *any* possible values chosen by the adversary. In our analysis, we will consider only verification queries $(m, \mathcal{P}, \sigma)$ such that $\mathcal{P}$ is not well-defined in $\mathcal{T}$, as these are the only queries that may cause setting $\zeta \leftarrow 1$.

Let $(m, \mathcal{P}, \sigma)$ be the j^{th} VERO query for some $j \leftarrow\!\!\$\, [Q_V]$. Depending on whether $\sigma = (y_0, y_1)$ or $\sigma = \Lambda$, we have two possible cases for ξ_j to occur:

- CASE A: Either $z \leftarrow \mathrm{e}(u_1, w_1^{\rho^{**}}) \cdot \mathrm{e}(u_1^{x_{1_0} \cdot y_1}, w_1)^{-1} \cdot \mathrm{e}(u_1, w_1^{y_0})^{-1} = 1$
- CASE B: Or $Z \leftarrow \mathrm{e}(u_1, w_1^{\rho^{**}}) \cdot \mathrm{e}(u_1, w_1^{m})^{-1} \cdot \mathrm{e}(\Lambda, w_1)^{-1} = 1$

We note that in both cases ρ^{**} is computed using at least one value $r^{**}_\tau \in \mathbb{Z}_p$ such that $(\tau, \cdot) \notin \mathcal{T}$. For some $j \in [Q_V]$, let z_j (correspondingly Z_j) be the value computed in the j^{th} VERO query. Let NZ_j be the event $\neg\xi_1 \wedge \cdots \wedge \neg\xi_{j-1}$, and note that for $i \in [j-1]$, $\neg\xi_i$ may mean either $z_i \neq 1$ or $Z_i \neq 1$.

Since the input of the VERO query is either (y_0, y_1) or Λ, we get the following

$$\Pr[\xi_j \mid \neg\xi_1 \wedge \ldots \wedge \neg\xi_{j-1}] \leq \Pr[z_j = 1 \mid NZ_j] + \Pr[Z_j = 1 \mid NZ_j] \tag{11}$$

where the probability is taken over the random choice of r_τ^{**}.

To evaluate the probabilities of equation 11, we observe that ρ^{**} can be thought of a univariate polynomial $\rho^{**} = \alpha(r_\tau^{**})$ in the variable r_τ^{**} such that $\tau \notin \mathcal{T}$ and degree of α is at most D. We recall that we are in the case of $\mathcal{P}$ is not well defined in $\mathcal{T}$. So $\mathcal{P}$ must be a non constant polynomial. We also recall that the value r_τ^{**} was sampled uniformly at random and it was never was used before to produce a tag since $\tau \notin \mathcal{T}$.

Recall that the value of x_{1_0} is fixed at the beginning of the game. We observe that prior to the adversary commencing VERO queries, there exist precisely p tuples $(x_{1_0}, \{r_\tau\}_{\tau\in\mathcal{T}})$ consistent with her view. After the first query if $Z_1 \neq 1$ OR $(z_1 \neq 1)$ then the no of possible values $\{x, \{r_\tau\}_{\tau\in\mathcal{T}}\}$ becomes at least $p - D$ as the roots of a non zero polynomial of degree D are at most D. After the $(j-1)^{th}$ query if $(Z_i)_{\forall i\in[j-1]} \neq 1$ OR $(z_i)_{\forall i\in[j-1]} \neq 1)$ then the number of remaining possible values $(x, \{r_\tau\}_{\tau\in\mathcal{T}})$ is at least $p - D(j-1)$.

Analysis of case A: In this case the ρ^{**} is a one degree polynomial. Conditioned on $NZ_{j-1} \neq 1$ the probability of the adversary $\mathcal{A}$ to guess r_τ^{**} out of the reduced space $p - D(j-1)$ at the j^{th} query cannot be better than $\frac{1}{p-D(j-1)}$

Analysis of case B: In this case the ρ^{**} is a D degree polynomial. Conditioned on $NZ_{j-1} \neq 1$ the probability of the adversary $\mathcal{A}$ to guess the D values of r_τ^{**} out of the reduced space $p - D(j-1)$ at the j^{th} query cannot be better than $\frac{D}{p-D(j-1)}$

Substituting these values into Equations 11, followed by 10 and then Equation 9, we get:

$$\Pr[\eta] \leq \sum_{j=1}^{Q_V} \left(\frac{1}{p - D(j-1)} + \frac{D}{p - D(j-1)} \right) \leq \frac{Q_V(D+1)}{p - D(Q_V - 1)} \tag{12}$$

AGGREGATE VERIFICATION QUERY

For $j_A \in [Q_{AV}]$, let ξ_{j_A} be the event that flag ζ_A is assigned the value 1 only after the j_A^{th} aggregate verification query but not before. Clearly, by union bound we have:

$$\Pr[\eta_A] \leq \sum_{j=1}^{[Q_{AV}]} \Pr[\xi_{j_A}] \tag{13}$$

Moreover, by the definition of ξ_{j_A}, event $\zeta_A \leftarrow 1$ did not occur in the previous $j_A - 1$ AVERO queries. Therefore we have

$$\Pr[\xi_{j_A}] = \Pr[\xi_{j_A} \mid \neg\xi_1 \wedge \cdots \wedge \neg\xi_{j_A-1}] \tag{14}$$

Let NZ_{j_A} be the event $\neg\xi_1 \wedge \cdots \wedge \neg\xi_{j_A-1}$, and note that for $i \in [j_A - 1]$, $\neg\xi_i$ may mean $Z_{A_i} \neq 1$. Since the input of the AVERO query is in the form of homomorphic tag $\mathrm{e}(\Lambda, w)$, we get the following

$$\Pr[\xi_{j_A} \mid \neg\xi_1 \wedge \ldots \wedge \neg\xi_{j_A-1}] \leq \Pr[Z_{A_{j_A}} = 1 \mid NZ_{j_A}] \tag{15}$$

where the probability is taken over the random choice of $r"_{\tau"_{1,i}}$ such that $\tau"_{1,i} \notin \mathcal{T}$ for some $i \in [n]$. The intuition for the below proof is estimating the probability $\Pr[\xi_{j_A}]$ taken over the random choices of $r"_{\tau"_{1,i}}$ such that $\tau"_{1,i} \notin \mathcal{T}$ and over any possible values chosen by the adversary.

We recall that if the event η_A did not happen then answers to AVERO of $\mathbf{G}^1_{\Theta,\mathcal{A}}$ is identical to that of $\mathbf{G}^2_{\Theta,\mathcal{A}}$. The only change is in the way the challenger answers AVERO queries such that program $\mathcal{P}'_1$ is *not* well defined on $\mathcal{T}$ as these are the queries that may cause flag $\zeta_A \leftarrow 1$.

Let $\left((m'_l, \mathcal{P}'_l)_{\forall l \in U}, (\mathsf{ek}'_l)_{\forall l \in U \setminus \{1\}}\right)$ be the j_A^{th} AVERO query. Challenger recomputes the hash of the queried messages as $(w_l)_{\forall l \in U}$. Let $(f'_l, (\tau"_{l,i})_{\forall i \in [n]})$ be the l^{th} program $\mathcal{P}'_l$ for some $l \in U$. Recall that $\mathcal{P}'_1$ is not well defined in $\mathcal{T}$, meaning there must exist at least one $i \in [n]$ such that $\tau"_{1,i} \notin \mathcal{T}$. Therefore $\forall \tau"_{1,i} \notin \mathcal{T}$: challenger computes $r"_{\tau"_{1,i}} \leftarrow \mathcal{R}(\tau"_{1,i})$. Next challenger computes $\rho"_1$ from $\mathcal{T}$ using the internal algorithm on $(r"_{\tau"_{1,i}})_{\forall i \in [n]}$. It then also computes $(\rho"_l)_{\forall l \in U \setminus \{1\}}$ using the internal AggVer algorithm.

Next, given the queried aggregate MAC σ', challenger extracts claimed homomorphic mac of m'_1 under ek_1, sk_1 by using equation 1. We note that the above holds because we are analyzing the AVERO query that is accepted in real but rejected in $\mathbf{G}^2_{\Theta,\mathcal{A}}$. We note that the only change from real to $\mathbf{G}^2_{\Theta,\mathcal{A}}$ is in case of party p_1. This makes that MACs of all other parties must have been valid. Therefore referring to equation 1, we substitute accordingly for parties in $U \setminus \{1\}$ to get the following

$$B \leftarrow \sigma' \cdot \left(\Pi_{\forall l \in U \setminus \{1\}} \mathrm{e}(u_l, w_l^{\rho"_l}) e(u_l, w_l^{m'_l})^{-1}\right)^{-1} \tag{16}$$

Since B is the extracted MAC under $\mathsf{ek}_1, \mathsf{sk}_1$ the following must hold for ξ_{j_A} to occur, conditioned on NZ_{j_A}:

$$Z_A \leftarrow \mathrm{e}(u_1, w_1^{\rho"_1})(B \cdot \mathrm{e}(u_1, w_1^{m'_1})^{-1}) = 1 \tag{17}$$

We recall that B in equation 16 is the claimed extracted mac under $\mathsf{ek}_1, \mathsf{sk}_1$ on message m'_1 as an output of $\mathcal{P}'_1$. We note that after honest extraction of this claimed mac by the internal AggVer algorithm, $\rho"_1$ becomes a degree D polynomial in $r"_{\tau"_{1,i}}$ for some $\tau"_{1,i} \notin \mathcal{T}$. More concretely, to evaluate the probability of equation 14 we observe that honest extraction of the MAC of the first message B allows the interpretation of $\rho"_1$ as a univariate polynomial $\rho"_1 = \alpha(r"_{\tau"_{1,i}}$ such that $r"_{\tau"_{1,i}} \notin \mathcal{T}$ for some $\tau"_{1,i} \notin \mathcal{T}$ where $i \in [n]$. Therefore $\mathcal{P}'_1$ must be a non constant polynomial.

We recall that the value $r"_{\tau"_{1,i}}$ was sampled uniformly at random and it was never used for any prior AVERO query to produce a tag since $\tau"_{1,i} \notin \mathcal{T}$. We recall that x_{1_0} is fixed in the beginning. Before $\mathcal{A}$ made any AVERO query there are exactly P tuples $\{x_{1_0}, \{r"_{\tau"_{1,i}}\}_{\tau"_{1,i} \in \mathcal{T} | \forall i \in [n]}\}$. After the first AVERO query if Z_A as defined in equation 17 is not equal to 1 then the number of possible values of $\{x_{1_0}, \{r"_{\tau"_{1,i}}\}_{\tau"_{1,i} \in \mathcal{T} | \forall i \in [n]}\}$ becomes at least p-D as the roots of a non zero polynomial α of degree D is at most D. After the $(j_A - 1)^{th}$ query if $(Z_{A_i})_{\forall i \in [j-1]} \neq 1$

then the number of remaining values $\{x_{1_0}, \{r"_{\tau"_{1,i}}\}_{\tau"_{1,i} \in \mathcal{T} | \forall i \in [n]}\}$ is at least $p - D(j_A - 1)$.

We recall here that our construction *only* considers aggregation of homomorphically evaluated macs. Therefore $\rho"_1$ is a D degree polynomial. Therefore conditioned on $NZ_{j-1} \neq 1$, the probability that $\mathcal{A}$ guesses the D roots $r"_{\tau"_{1,i}}$ for some $i \in [n]$ out of the reduced space $p - D(j_A - 1)$ at the j_A^{th} query cannot be better than $\frac{D}{p - D(j_A - 1)}$. More concretely consider the equation below

$$\Pr[Z_{A_{j_A}} = 1 \mid NZ_{j_A}] \leq \frac{D}{p - D(j_A - 1)} \tag{18}$$

Substituting the above value in equation 15 and then in equation 14 and then finally in equation 13

$$\Pr[\eta_A] \leq \frac{DQ_{AV}}{p - D(Q_{AV} - 1)} \tag{19}$$

Taking results of equations 12, 19 and substituting in equation 8 concludes the proof of our claim above. □

Lemma 4. $|\Pr[\mathbf{G}^2_{\Theta,\mathcal{A}}] - \Pr[\mathbf{G}^3_{\Theta,\mathcal{A}}]| \leq \frac{DQ_V^2}{p} + \frac{DQ_{AV}^2}{p}$ *where* p, D, Q_V, Q_{AV} *are as defined in lemma 3.*

Proof (Proof of Lemma 4). $\mathbf{G}^2_{\Theta,\mathcal{A}}$ and $\mathbf{G}^3_{\Theta,\mathcal{A}}$ (Figure 6) differ only in how the random evaluation point r_τ is sampled during authentication queries. For each authentication query with label message pair (τ, m), suppose there were Q_V prior verification queries involving the label τ. According to the argument in the previous lemma, the number of possible values for r_τ is at least $p - DQ_V$. More precisely, the two games will only differ if sampling a new y_1^{**} results in one of the DQ_V excluded values, given the condition NZ_{Q_V}, where NZ_{Q_V} means that neither z nor Z is 1 in the preceding $(Q_V - 1)$ queries. The probability of this occurring is $\frac{DQ_V}{p}$. Applying the union bound in this case results in a difference of $\frac{DQ_V^2}{p}$ between the games.

Similarly, consider that Q_{AV} number of AVERO queries were made before AUTHO queries. Also note that from the queried σ', honest extraction of the MAC on m'_1 is computed by the internal AggVer algorithm. Therefore applying similar argument as above yields that the difference between the two games is $\frac{DQ_{AV}^2}{p}$. We recall that $\mathbf{G}^2_{\Theta,\mathcal{A}}$ and $\mathbf{G}^3_{\Theta,\mathcal{A}}$ differs if either the output distribution of VERO or AVERO differs from $\mathbf{G}^2_{\Theta,\mathcal{A}}$ to $\mathbf{G}^3_{\Theta,\mathcal{A}}$. Therefore, over the two scenarios collectively, the two games differ by a probability that is upper bounded by $\frac{DQ_V^2}{p} + \frac{DQ_{AV}^2}{p}$, where the probability is taken over the random sampling of y_1^{**}. □

Indistinguishability of $\mathbf{G}^3_{\Theta,\mathcal{A}}$ *and* $\mathbf{G}^4_{\Theta,\mathcal{A}}$*:* In order to argue indistinguishability of $\mathbf{G}^3_{\Theta,\mathcal{A}}$ and $\mathbf{G}^4_{\Theta,\mathcal{A}}$ we first define a bad even η_4 and prove that as long as η_4 does not happen, the two games are identical. We then argue that η_4 only happens with negligible probability. See Lemmas 5, 6 and 7 and Claims B1–B3 for the proof.

Lemma 5. $\Pr[\mathbf{G}^3_{\Theta,\mathcal{A}}] = Pr[\mathbf{G}^4_{\Theta,\mathcal{A}} \wedge \neg\eta_4]$

Proof. Let η_4 be the event that flag η is set to 1 in $\mathbf{G}^4_{\Theta,\mathcal{A}}$. Conditioned on event η_4 not occurring, we argue that $\Pr[\mathbf{G}^3_{\Theta,\mathcal{A}}] = Pr[\mathbf{G}^4_{\Theta,\mathcal{A}}]$. Note that $\mathbf{G}^3$ and $\mathbf{G}^4$ differ in two ways. The first change is in the way the challenger answers verification queries $(m, \mathcal{P}, \sigma)$ for a program $\mathcal{P}$ that is well defined on $\mathcal{T}$. The second change is that all the hash queries are instead answered by choosing a random exponent $b \leftarrow_\$ \mathbb{Z}_p$ and outputting $w \leftarrow g_2^b$. Since w is a random group element in g_2, this does not change the output distribution. Now we consider a verification query $(m, \mathcal{P}, \sigma)$ where σ is either (y_0, y_1) or Λ, and $\mathcal{P} = (f, (\tau_{1,i})_{\forall i \in [n]})$ is well-defined on $\mathcal{T}$.

Case I: We first examine the scenario where $\forall i \in [n]$, $(\tau_{1,i}, m_{1,i}, \sigma_{1,i}) \in \mathcal{T}$. Challenger retrieves from the $\mathcal{H}$ table $\mathrm{RO}(m)$ as $w \leftarrow g_2^b$ for some $b \leftarrow_\$ \mathbb{Z}_p$. Recall that the challenger calculates σ^{***} using the Eval algorithm under the challenge key ek_1. Also recall that $(u_1^{x_{1_0}^k})_{\forall k \in [0,D]} \leftarrow \mathsf{ek}_1$ where sk_1 contains x_{1_0}. In this context when we evaluate the response given by the challenger, we observe the following:

- Case A ($\sigma = \sigma^{***}$): In the case where $\sigma = \sigma^{***}$, the VERO query accepts in both games.
- Case B: ($\sigma \neq \sigma^{***}$ on input $\sigma = (y_0, y_1)$, i.e., σ is an initial Mac): Let ρ_1 denote the value computed by the internal verification algorithm to validate equation 5. It is important to observe that ρ_1 is the same when executing both $\mathsf{Ver}(\mathsf{sk}_1, m, \mathcal{P}, \sigma)$ and $\mathsf{Ver}(\mathsf{sk}_1, m, \mathcal{P}, \sigma^{***})$ since the same random values have been used in evaluating the function.
  $\mathsf{Ver}(\mathsf{sk}_1, m, \mathcal{P}, \sigma^{***})$= 1 translates to the check $\mathrm{e}(u^{x_{1_0} y_1^{***}}, w_1)\mathrm{e}(u_1, w_1^{y_0^{***}}) = \mathrm{e}(u_1, w_1^{\rho_1})$. $\mathsf{Ver}(\mathsf{sk}_1, m, \mathcal{P}, \sigma)$= 1 translates to the check $\mathrm{e}(u^{x_{1_0} y_1}, w_1)\mathrm{e}(u_1, w_1^{y_0}) = \mathrm{e}(u_1, w_1^{\rho_1})$. Note that RHS of both the above is the same. So returning 1 iff $z \leftarrow \frac{\mathrm{e}(u_1^{y_1 x_1}, w_1)}{\mathrm{e}(u_1^{y_1^{***} x_1}, w_1)} \mathrm{e}(u_1, w_1)^{(y_0 - y_0^{***})} = 1$ is the same as returning the output of $\mathsf{Ver}(m, \mathcal{P}, \sigma)$.
- Case C ($\sigma \neq \sigma^{***}$ on input $\sigma = \Lambda$, i.e., σ is a homomorphic Mac): Let ρ_1 represent the value computed by the Ver algorithm to verify equation 6. It is essential to note that ρ must be identical for both $\mathsf{Ver}(\mathsf{sk}_1, m, \mathcal{P}, \Lambda)$ and $\mathsf{Ver}(\mathsf{sk}_1, m, \mathcal{P}, \Lambda^{***})$ since the same r_τ values are utilized in both scenarios. We also recall that for $\mathsf{Ver}(m, \mathcal{P}, \Lambda)$ to return 1 equation 6 must hold.
  $\mathsf{Ver}(\mathsf{sk}_1, m, \mathcal{P}, \Lambda^{***})$= 1 translates to the check $\mathrm{e}(\Lambda^{***}, w_1)\mathrm{e}(u_1, w_1^{m^{***}}) = \mathrm{e}(u_1, w_1^{\rho_1})$. $\mathsf{Ver}(\mathsf{sk}_1, m, \mathcal{P}, \Lambda)$= 1 translates to the check $\mathrm{e}(\Lambda, w_1)\mathrm{e}(u_1, w_1^{m}) = \mathrm{e}(u_1, w_1^{\rho_1})$. Note that RHS of both the above is the same. So returning 1 iff $Z \leftarrow \frac{\mathrm{e}(\Lambda, w_1)}{\mathrm{e}(\Lambda^{***}, w_1)} \cdot \mathrm{e}(u_1, w_1)^{(m - m^{***})} = 1$ is the same as returning the output of $\mathsf{Ver}(m, \mathcal{P}, \Lambda)$.

Case II: Consider the case where $\mathcal{P}$ is well-defined on $\mathcal{T}$ but there exists an i such that $(\tau_{1,i}, \cdot) \notin T$. By the definition of well defined program we know that keeping the input values of wires labeled with $\tau_{1,i}$ where $(\tau_{1,i}, \cdot) \in T$ for some

$i \in n$ always results in a consistent output from circuit f, irrespective of the values of wires labeled with $\tilde{\tau}_{1,i}$ where $(\tilde{\tau_{1,i}}, \cdot) \notin \mathcal{T}$. Meaning, the value corresponding to the input wire $\tilde{\tau}_{1,i}$ for any $i \in [n]$ is irrelevant when evaluating f. This remains true even during the homomorphic evaluations of f in Eval as well as computing f in the Ver algorithm . This means that for all wires of queried $\mathcal{P}$ with labels such that they are missing from $\mathcal{T}$, the dummy tags chosen for these labels do not contribute to the computation of σ^{***}. The same applies to the computation of f on random values to obtain ρ_1. Therefore, the previous argument for the case when the queried program $\mathcal{P}$ is well defined program with respect to $\mathcal{T}$ and none of the labels in $\mathcal{P}$ is missing from $\mathcal{T}$ applies here as well where there exist labels of $\mathcal{P}$ missing from $\mathcal{T}$.

□

Lemma 6. $\Pr[\eta_4] < e^{\frac{Q_A+Q_V+Q_{AV}}{Q_A+Q_V+PQ_{AV}}}(Q_A + Q_V + PQ_{AV}) \cdot \mathbf{Adv}^{co-CDH}_{\Theta,\mathcal{B}}$ *where* $\mathcal{B}$ *is any PPT adversary against the* (t', ϵ') *security of co-CDH assumption over* $(\mathcal{G}_1, \mathcal{G}_2)$ *and* e *is the base of natural log.*

Proof (Proof of Lemma 6).

If there exists an adversary $\mathcal{A}(t, Q_H, Q_A, Q_V, Q_{AV}, P, \epsilon)$ that makes a forgery query i.e., a query such that event η_4 occurs in VerO, with probability at least ϵ, then we show that there exists an adversary $\mathcal{B}$ that breaks the co-CDH assumption on $(\mathcal{G}_1, \mathcal{G}_2)$ with probability at least ϵ'. By hardness of co-CDH this implies that $\Pr[\eta_4] < \epsilon$.

Using $\mathcal{A}$, we construct an adversary $\mathcal{B}$ (Figure 9) that solves co-CDH in $(\mathcal{G}_1, \mathcal{G}_2)$ with probability at least ϵ' and running time at most t'. By construction we know that every VerO and AVerO query calls the hash function internally. Without loss of generality we also consider that for every AuthO, the hash function is called on the queried message internally. Therefore the following analysis is under the scenario that all incoming messages to any oracle get registered to the $\mathcal{H}$ list prior to the query being made. We recall from section A that superscript refers to the query id and subscript refers to the party id. [4]

At the start $\mathcal{B}$ is given $g_2, g_2^a \in \mathcal{G}_2$, $h \in \mathcal{G}_1$ as its challenge. Its objective is to output h^a. We recall from Section 3.4 that $\mathcal{G}_1$ has a generator g_1 such that $g_1 = \psi(g_2)$. Then, viewing h as $h = g_1^\delta$ for some $\delta \in \mathbb{Z}_p$ we can reinterpret $\mathcal{B}$'s goal as given g_2, g_2^a, g_1, and g_1^δ output $g_1^{\delta a}$.

$\mathcal{B}$ simulates the challenger of $\mathbf{G}^4_{\Theta,\mathcal{A}}$ and interacts with $\mathcal{A}$ as follows. $\mathcal{B}$ starts by sampling its own key pairs for all P users by simulating KeyGen, except for the challenge user party p_1. For p_1 $\mathcal{B}$ chooses $x_{1_0} \leftarrow_\$ \mathbb{Z}_p$ and $K_1 \leftarrow_\$ \mathcal{K}$ and sets $sk_1 = (x_{1_0}, K_1)$ and $ek_1 = (h^{x^k_{1_0}})_{\forall k \in [0,D]}$ where h is the value from the co-CDH challenge. $\mathcal{B}$ returns ek_1 and the key pairs (ek_i, sk_i) for all the other parties. Note that the output distribution of Initialize as seen by $\mathcal{A}$ when run by $\mathcal{B}$ is identical to the distribution that is seen by $\mathcal{A}$ when run by challenger of $\mathbf{G}^4_{\Theta,\mathcal{A}}$.

[4] In what follows in lemmas 6 through 7, for ease of presentation we drop the subscript 1 of coin c since VerO is under $\mathsf{ek}_1, \mathsf{sk}_1$ by definition. Moreover we also drop the superscript unless otherwise needed.

At any time algorithm $\mathcal{A}$ can query the RO with a message y. In order to respond to the query, $\mathcal{B}$ maintains a list of tuples $\langle y, w, b, c\rangle$ called the $\mathcal{H}$-list which is initially empty. When $\mathcal{A}$ queries the RO at a point $y \in \{0,1\}^*$, algorithm $\mathcal{B}$ responds as follows:

- If the query y already appears on the $\mathcal{H}$-list, $\mathcal{B}$ retrieves corresponding entry and returns w back to $\mathcal{A}$.
- Otherwise $\mathcal{B}$ tosses a random coin $c \in \{0,1\}$ such that $\Pr[c=0] = \frac{1}{Q_A+Q_V+PQ_{AV}}$. Then, $\mathcal{B}$ samples $b \leftarrow\!\!\$\, \mathbb{Z}_p$. If $c=0$, $\mathcal{B}$ answers $\mathcal{A}$ with $w = g_2^{a+b}$ where g_2^a is part of the co-CDH challenge, and b is the sampled exponent. Else if $c=1$ $\mathcal{B}$ answers $\mathcal{A}$ with $w = g_2^b$. It then stores the tuple $< y, w, b, c >$ in the $\mathcal{H}$-list.

We observe that in either case, w is just a random element in $\mathcal{G}_2$. Thus, w is independent of the bit c.

To respond to AuthO, VerO, and AVerO queries, $\mathcal{B}$ retrieves the corresponding tuple $\langle m, w, b, c\rangle$ from the $\mathcal{H}$-list. Then, as long as $c=1$, he simulates the answers exactly as in $\mathbf{G}^4_{\Theta,\mathcal{A}}$. If $c=0$, then $\mathcal{B}$ stop responding to oracle queries and proceeds to the output step i.e., Out procedure.

Now, lets consider the first query such that $c=0$ in the corresponding $\mathcal{H}$-tuple. If such a query never occurs, then $\mathcal{B}$ aborts and outputs fail. Let this query be a VerO query with input $(m^*, \mathcal{P}^*, \sigma^*)$. $\mathcal{B}$ checks whether the event η_4 occurs on this query, and if not, he aborts and outputs fail. Note that if the event η_4 occurs then this query is one that will be accepted in $\mathbf{G}^3_{\Theta,\mathcal{A}}$ but is rejected in $\mathbf{G}^4_{\Theta,\mathcal{A}}$, we call this a *forgery query*. In the case of such a forgery query occurring with $c=0$, $\mathcal{B}$ runs the Out procedure to recover h^a solving co-CDH.

We now describe how Out recovers h^a on such a forgery query. Recall that since $c=0$, we program $\mathrm{RO}(m^*) = w = g_2^{a+b}$. We now consider the two possible cases for this query:

- $\mathcal{A}$ submitted $\sigma^* = \Lambda^*$ (i.e., σ^* is a homomorphic Mac): Recall that we are in the case where $\mathcal{P}^* = (f^*, (\tau^*_{1,i})_{\forall i\in[n]})$ is well-defined on $\mathcal{T}$. Additionally, we assume that $\forall i \in [n], (\tau^*_{1,i}, m_{1,i}, \sigma_{1,i}) \in \mathcal{T}$, i.e., the values on all input wires have been authenticated. Note that if this is not the case and there are some input wires not in $\mathcal{T}$, by the fact that $\mathcal{P}^*$ is well-defined, we know that such input wires have no impact on the output of f^*. So, $\mathcal{B}$ can just choose random labels for these wires. Now, $\mathcal{B}$ computes the polynomial coefficients $y_{1,k} \leftarrow f^*(\sigma_{(1,i)_{\forall i\in[n]}})$ where $\sigma_{1,i} \leftarrow \mathcal{T}(\tau^*_{1,i})$ and uses ek_1 to recompute the Mac $\bar{\Lambda} \leftarrow h^{\sum_{k=1}^{D} y_{1,k} x_{10}^k}$ and message $\bar{m} = f^*(m_{1,1}, \ldots, m_{1,n})$. $\mathcal{B}$ also computes ρ^* for $\mathcal{P}^*$ using his internal Ver algorithm under sk_1.
  Since η_4 occurs on this query, we know two things. First, $(m^*, \mathcal{P}^*, \sigma^*)$ correctly verifies in $\mathbf{G}^3_{\Theta,\mathcal{A}}$, and second, the ratio between the claimed Mac Λ^* and recomputed Mac $\bar{\Lambda}$ satisfies the relationship necessary to trigger η_4. Specifically, since we set $u_1 = h$ in the ek, and since $(m^*, \mathcal{P}^*, \sigma^*)$ correctly verifies, we know that

$$\mathrm{e}(\Lambda^*, w) = \mathrm{e}(h, w)^{(\rho^* - m^*)} \tag{20}$$

From the fact that η_4 has occurred, we know that

$$\mathrm{e}(\bar{\Lambda}, w) = \mathrm{e}(\Lambda^*, w) \cdot \mathrm{e}(g_1, w)^{(m^* - \bar{m})} \tag{21}$$

Now, since we set $w = g_2^{a+b}$, combining terms we get that

$$\mathrm{e}(h^{(a+b)((m^* - \rho^*) + \sum_{k=1}^{D} y_{1,k} x_{1_0}^k)}, g_2) = \mathrm{e}(g_1, g_2^{(a+b)(m^* - \bar{m})}) \tag{22}$$

So,

$$h^{a+b} = \psi\big(w^{(m^* - \bar{m})}\big)^{(m^* - \rho^* + \sum_{k=1}^{D} y_{1,k} x_{1_0}^k)^{-1}} \tag{23}$$

$\mathcal{B}$ then outputs $h^a = h^{a+b} \cdot \big(h^b\big)^{-1}$

– Or $\mathcal{A}$ submitted $\sigma^* = (y_0^*, y_1^*)$: We recall that we are in the case of non-trivial forgery $m^* \notin \mathcal{T}$. Therefore using similar analysis as above, in order to satisfy equation 5, the following must hold.

$$\mathrm{e}(h^{(a+b)x_{1_0}(\bar{y_1} - y_1^*)}), g_2) = \mathrm{e}(g_1, g_2^{(a+b)(y_0^* - \bar{y}_0)}) \tag{24}$$

It is straightforward to see that $\mathcal{B}$ correctly returns the solution h^a of co-CDH as

$$\psi\big(g_2^{(a+b)(y_0^* - \bar{y}_0)}\big)^{(x_{1_0}(\bar{y}_1 - y_1^*))^{-1}} \cdot \big(h^b\big)^{-1}.$$

Now we need to show that $\mathcal{B}$ as described above solves the given instance of the co-CDH problem in $(\mathcal{G}_1, \mathcal{G}_2)$ with probability at least ϵ'. In order to prove that we need to analyze the following events. Consider the following events needed for $\mathcal{B}$ to succeed: (i) Let $\mathcal{E}_1$ be the event that $\mathcal{B}$ does not abort in Auth queries. (ii) Let $\mathcal{E}_2$ be the event that $\mathcal{B}$ does not abort in Ver oracle. (iii) Let $\mathcal{E}_3$ be the event that $\mathcal{B}$ does not abort in AggVer oracle. (iv)Let $\mathcal{E}_4$ be the event that valid forgery query has taken place. (v)Let $\mathcal{E}_5$ be the event that event $\mathcal{E}_4$ has taken place and in addition $c = 0$ where c is the component of the tuple m_1 on the $\mathcal{H}$ list.

We define $\mathcal{B}$ wins if all these events take place. We note that since $\mathcal{E}_4$ occurs whenever $\mathcal{E}_5$ occurs, we do not need to include $\mathcal{E}_4$ in the final expression.

$$\begin{aligned}&\Pr[\mathcal{E}_1 \wedge \mathcal{E}_2 \wedge \mathcal{E}_3 \wedge \mathcal{E}_5] = \\ &\Pr[\mathcal{E}_1 \wedge \mathcal{E}_2 \wedge \mathcal{E}_3] \cdot \Pr[\mathcal{E}_4 | \mathcal{E}_1 \wedge \mathcal{E}_2 \wedge \mathcal{E}_3] \cdot \Pr[\mathcal{E}_5 | \mathcal{E}_1 \wedge \mathcal{E}_2 \wedge \mathcal{E}_3 \wedge \mathcal{E}_4]\end{aligned} \tag{25}$$

The following claims B1, B2, B3 give the lower bound for each of these terms.

Claim B1 *The probability that algorithm $\mathcal{B}$ does not abort due to the* AuthO, VerO *and* AVerO *queries made by adversary $\mathcal{A}$ in $\mathbf{G}^4_{\Theta,\mathcal{A}}$ is at least $(1 - \frac{1}{Q_A + Q_V + PQ_{AV}})^{Q_A + Q_V + Q_{AV}}$, where Q_A, Q_V, Q_{AV} represents the number of* Auth, Ver *and* AggVer *queries made by $\mathcal{A}$ in $\mathbf{G}^4$. Therefore, $\Pr[\mathcal{E}_1 \wedge \mathcal{E}_2 \wedge \mathcal{E}_3] \geq (1 - \frac{1}{Q_A + Q_V + PQ_{AV}})^{(Q_A + Q_V + Q_{AV})}$.*

Proof (Proof of Claim B1).

Without loss of generality, we assume that $\mathcal{A}$ does not request the VERO on the same message twice. We recall that every call to VERO is also an internal call to RO. Therefore by making the prior assumption we are implicitly also assuming that $\mathcal{A}$ does not request RO on the same message twice. We will show by induction that after $\mathcal{A}$ makes l VERO queries, the probability that $\mathcal{B}$ does not abort is at least $(1 - \frac{1}{Q_A+Q_V+PQ_{AV}})^l$. We define the superscript notation $(m^{(l)}, w^{(l)}, b^{(l)}, c^{(l)})$ with respect to l^{th} query in any of Auth, Ver or AggVer oracles.

The base case is trivial for $l = 0$. Let $(m^{(l)})$ be $\mathcal{A}$'s l-th VERO query, and let $(m^{(l)}, w^{(l)}, b^{(l)}, c^{(l)})$ be the corresponding tuple in the $\mathcal{H}$-list. Before $\mathcal{A}$ issues this query to VERO, the bit $c^{(l)}$ is independent of $\mathcal{A}$'s view. Meaning before $\mathcal{A}$ makes this VERO query the bit $c^{(l)}$ is independent of $\mathcal{H}(m^{(l)})$. After this VERO query is made, the only value potentially dependent on $c^{(l)}$ that could have been given to $\mathcal{A}$ is $\mathcal{H}(m^{(l)})$ which have the same distribution whether $c^{(l)} = 0$ or $c^{(l)} = 1$. This is because $\mathcal{H}(m^{(l)})$ is a random element in $\mathcal{G}_2$ in both the cases. Therefore, the probability that this query causes $\mathcal{B}$ to abort is at most $\frac{1}{Q_A+Q_V+PQ_{AV}}$ because of the independence from the bit c. Therefore by inductive hypothesis, the probability that $\mathcal{B}$ does not abort after this query is at least $(1 - \frac{1}{Q_A+Q_V+PQ_{AV}})^l$.

Since $\mathcal{A}$ makes at most Q_V VERO queries, the probability that $\mathcal{B}$ does not abort due to all VERO queries is at least $(1 - \frac{1}{Q_A+Q_V+PQ_{AV}})^{Q_V}$. Similarly,the probability that $\mathcal{B}$ does not abort due to any of the Auth queries is at least $(1-\frac{1}{Q_A+Q_V+PQ_{AV}})^{Q_A}$. Also, the probability that $\mathcal{B}$ does not abort due to any of the AggVer queries is at least $(1-\frac{1}{Q_A+Q_V+PQ_{AV}})^{Q_{AV}}$. Multiplying this concludes our claim that probability that $\mathcal{B}$ did not abort in Auth, Ver and AggVer queries. Thus, we have $\Pr[\mathcal{E}_1 \wedge \mathcal{E}_2 \wedge \mathcal{E}_3] \geq (1 - \frac{1}{Q_A+Q_V+PQ_{AV}})^{Q_A+Q_V+Q_{AV}}$.

□

Claim B2 *If algorithm $\mathcal{B}$ does not abort due to the queries made by $\mathcal{A}$ in $\mathbf{G}^4_{\Theta,\mathcal{A}}$, then the view of algorithm $\mathcal{A}$ in $\mathbf{G}^4_{\Theta,\mathcal{A}}$ is identical to its view when run by $\mathcal{B}$. Hence,* $\Pr[\mathcal{E}_4|\mathcal{E}_1 \wedge \mathcal{E}_2 \wedge \mathcal{E}_3] \geq \epsilon$

Proof (Proof of Claim B2). The evaluation keys and the secret keys provided to $\mathcal{A}$ when played by $\mathcal{B}$ have the same distribution as the evaluation keys and secret keys generated by KeyGen in $\mathbf{G}^4_{\Theta,\mathcal{A}}$. Additionally, the responses to hash queries are uniformly and independently distributed in $\mathcal{G}_2$. Since $\mathcal{B}$ does not abort due to $\mathcal{A}$'s authentication, verification, and aggregate verification queries, all responses to those queries are valid. Therefore, the probability of event $\mathcal{E}_4$ given events $\mathcal{E}_1$, $\mathcal{E}_2$, and $\mathcal{E}_3$ have occurred is at least ϵ. In other words probability that $\mathcal{A}$ made a query such that it is accepted in $\mathbf{G}^3_{\Theta,\mathcal{A}}$ and rejected in $\mathbf{G}^4_{\Theta,\mathcal{A}}$ for a well defined program is negligible So, by contradiction it is concluded that $\Pr[\mathcal{E}_4|\mathcal{E}_1 \wedge \mathcal{E}_2 \wedge \mathcal{E}_3] \geq \epsilon$. □

For convenience we recall here that valid forgery refers to case of $\mathcal{A}$ making Ver query with $(m^*, \mathcal{P}^*, \sigma^*)$ such that $\Pr[\eta_4] \geq \epsilon$.

Claim B3 *The probability of algorithm $\mathcal{B}$ not aborting after $\mathcal{A}$ outputs a valid and non-trivial* VERO *forgery query in $\mathbf{G}^4_{\Theta,\mathcal{A}}$ is at least: $\frac{1}{Q_A+Q_V+PQ_{AV}}$. Hence, $\Pr[\mathcal{E}_5|\mathcal{E}_1 \wedge \mathcal{E}_2 \wedge \mathcal{E}_3] \geq \frac{1}{Q_A+Q_V+PQ_{AV}}$.*

Proof (Proof of Claim B3). Given that the events $\mathcal{E}_1$, $\mathcal{E}_2$, $\mathcal{E}_3$ and $\mathcal{E}_4$ have occurred, and $\mathcal{A}$ has produced a valid and non-trivial forgery query at $c = 0$. Let $\langle m^*, w, b, c\rangle$ be the tuple corresponding to m^* on the $\mathcal{H}$-list. Algorithm $\mathcal{B}$ will abort unless $\mathcal{A}$ generates a forgery query at $c = 0$. We note that w is independent of the c bit. Since the forgery query is non-trivial, $\mathcal{A}$ cannot have asked for a MAC on m^* under key sk_1^*. Therefore $\mathcal{A}$ cannot have any information about the value c. In the forged query, $c = 0$ occurs with probability $\frac{1}{(Q_A+Q_V+PQ_{AV})}$. Therefore $\Pr[\mathcal{E}_5|\mathcal{E}_1 \wedge \mathcal{E}_2 \wedge \mathcal{E}_3 \wedge \mathcal{E}_4] \geq \frac{1}{(Q_A+Q_V+PQ_{AV})}$ as required.

We substitute back the results of claims B1, B2, B3 into equation 25 of Lemma 6. By contradiction algorithm $\mathcal{B}$ breaks co-CDH with probability at least ϵ'. Hence we concretely get the following

$$(1 - \frac{1}{(Q_A + Q_V + PQ_{AV})})^{(Q_A+Q_V+Q_{AV})} \cdot \epsilon \cdot \frac{1}{(Q_A + Q_V + PQ_{AV})} \geq \epsilon' \quad (26)$$

Using taylor series approximation $(1 - \frac{1}{x})^x \geq \frac{1}{e}$ on equation 26 we get the following

$$\epsilon \geq e^{\frac{Q_A+Q_V+Q_{AV}}{Q_A+Q_V+PQ_{AV}}} (Q_A + Q_V + PQ_{AV})\epsilon'$$

Therefore the following concludes the proof of lemma 6

$$\Pr[\eta_4] < e^{\frac{Q_A+Q_V+Q_{AV}}{Q_A+Q_V+PQ_{AV}}} (Q_A + Q_V + PQ_{AV})\epsilon' \quad (27)$$

□
□

Lemma 7. *Let $(\mathcal{G}_1, \mathcal{G}_2)$ be a bilinear group pair for co-Diffie-Hellman. For any PPT adversary $\mathcal{A}(t, Q_H, Q_A, Q_V, Q_{AV}, \epsilon)$ that makes a* VERO *forgery query in game $\mathbf{G}^4_{\Theta,\mathcal{A}}$ with probability at least ϵ and running time at most t then there exists another PPT adversary $\mathcal{B}$ against co-CDH assumption with probability at least ϵ' and running time at most t' such that the following two inequalities hold*

$$\epsilon' > \frac{\epsilon}{e^{\frac{Q_A+Q_V+Q_{AV}}{Q_A+Q_V+PQ_{AV}}} (Q_A + Q_V + PQ_{AV})} \quad (28)$$

$$t' < t + c_{\mathbb{Z}} (Q_A + |f| (Q_V + PQ_{AV} + 2) + 1) + c_{\mathcal{G}_1} Q_V + c_{\mathcal{G}_2} (Q_H + Q_A + 3Q_V + (3P)Q_{AV} + 1) + c_{\mathcal{G}_T} (Q_V + Q_{AV} + 1) + c_{\mathrm{e}} (3Q_V + PQ_{AV} + 3)$$

Proof (Proof of lemma 7).

By the proof of lemmas 6 and Claims till B3, we have proven by contradiction that if an adversary $\mathcal{A}$ who breaks Θ by making a VERO forgery query in $\mathbf{G}^4_{\Theta,\mathcal{A}}$ with probability at least ϵ then there exists adversary $\mathcal{B}$ that breaks co-CDH with probability at least ϵ' such that $\epsilon' > \frac{\epsilon}{e^{\frac{Q_A+Q_V+Q_{AV}}{Q_A+Q_V+PQ_{AV}}}(Q_A+Q_V+PQ_{AV})}$. Now it remains to show, given that if there exists $\mathcal{A}$ that takes time at most t to break Θ then there exists $\mathcal{B}$ that takes time at most t' to break co-CDH.

The total running time of $\mathcal{B}$ consists of three main components: (1) The running time of algorithm $\mathcal{A}$, since $\mathcal{B}$ invokes $\mathcal{A}$ to receive queries, (2) The time to respond to $(Q_H + Q_A + Q_V + PQ_{AV})$ hash queries, each requiring an exponentiation in $\mathcal{G}_2$, (3) The time to answer Q_A Auth queries, Q_V Ver queries, Q_{AV} AggVer queries (4) the time to transform $\mathcal{A}$'s final forgery into the co-CDH solution. We recall Figure 2 for the analysis below.

Let $c_{\mathcal{G}_1}$, $c_{\mathcal{G}_2}$, $c_{\mathcal{G}_T}$, $c_{\mathbb{Z}}$, c_{e} denote the time for group operations in $\mathcal{G}_1$, $\mathcal{G}_2$, $\mathcal{G}_T$, $\mathbb{Z}_p$, compute map e. For simplicity PRF computation cost is not there in the analysis.

- $(Q_H+Q_A+Q_V+PQ_{AV})$ Hash queries: each hash query has 1 exponentiation in $\mathcal{G}_2$. RO cost = $c_{\mathcal{G}_2}(Q_H + Q_A + Q_V + PQ_{AV})$
- Q_A Auth queries: Each auth query makes 2 operations in $\mathbb{Z}_p$. AUTHO cost = $2c_z(Q_A)$
- Q_V Ver queries: Each VERO query does $|f|$ no of operations in $\mathbb{Z}_p$ where f is the function submitted. Depending on whether the Ver oracle gets queried on initial MAC or homomorphic MAC, either 2 exponentiations in $\mathcal{G}_2$ +3 e operations + 1 mult in $\mathcal{G}_T$ are performed OR 2 exponentiations in $\mathcal{G}_2$ + 3 e operations + 1 mult in $\mathcal{G}_T$ + 1 exponentiations in $\mathcal{G}_1$. Taking the costlier operation for concrete calculation we get that total VERO cost = $Q_V(c_z|f| + 2c_{\mathcal{G}_2} + 3c_{\mathrm{e}} + c_{\mathcal{G}_T} + c_{\mathcal{G}_1})$.
- Q_{AV} AggVer queries: Each AVERO query has $P|f|$ no of operations in $\mathbb{Z}_p$ + $2P$ exponentiations in $\mathcal{G}_2$ + P maps + (1+2P) mult in $\mathcal{G}_T$. Therefore total AVERO cost = $Q_{AV}(c_{\mathbb{Z}}(P|f|) + c_{\mathcal{G}_2}(2P) + c_{\mathrm{e}}P + c_{\mathcal{G}_T})$.
- Forgery: Operations needed to calculate solution to co-CDH from the forgery of homomorphic MAC under the honest key is $|f|$ operations in $\mathbb{Z}_p$, 1 operation in $\mathbb{Z}_p$, 1 operation in $\mathcal{G}_2$, 3 map operations, 1 mult in $\mathcal{G}_T$. Total cost for forgery inversion is $c_{\mathcal{G}_2} + 3c_{\mathrm{e}} + c_{\mathbb{Z}}(2|f| + 1) + c_{\mathcal{G}_T}$.

Summing these components, the total running time of $\mathcal{B}$ is at most :
$t + c_{\mathbb{Z}}\left(2Q_A + |f|\left(Q_V + PQ_{AV} + 2\right) + 1\right) + c_{\mathcal{G}_1}Q_V +$
$c_{\mathcal{G}_2}\left(Q_H + Q_A + 3Q_V + (3P)Q_{AV} + 1\right) + c_{\mathcal{G}_T}\left(Q_V + Q_{AV} + 1\right) +$
$c_{\mathrm{e}}\left(3Q_V + PQ_{AV} + 3\right)$.
This concludes the proof of lemma 7. □

Indistinguishability of $\mathbf{G}^4_{\Theta,\mathcal{A}}$ and $\mathbf{G}^5_{\Theta,\mathcal{A}}$: In order to prove indistinguishability of $\mathbf{G}^4_{\Theta,\mathcal{A}}$ and $\mathbf{G}^5_{\Theta,\mathcal{A}}$, we again define an event η_5 such that the two games are identical unless η_5 occurs. We then show that η_5 occurs with negligible probability. See lemmas 8–10 for the proofs.

Lemma 8. $\Pr[\mathbf{G}^4_{\Theta,\mathcal{A}}] \equiv \Pr[\mathbf{G}^5_{\Theta,\mathcal{A}} \wedge \neg\eta_5]$

Proof. Let η_5 represent the event where flag β_A is assigned 1 in $\mathbf{G}^5_{\Theta,\mathcal{A}}$ of Figure 8. If the event η_5 does not occur, we claim that $\mathbf{G}^4_{\Theta,\mathcal{A}}$ is identical to $\mathbf{G}^5_{\Theta,\mathcal{A}}$. The only distinction lies in how the challenger responds to the AVERO query $((m'_j, \mathcal{P}'_j)_{\forall j\in U}, (\mathsf{ek}'_j)_{\forall j\in U\setminus\{1\}}), \sigma')$ such that $\mathcal{P}'_1$ is well defined on $\mathcal{T}$.

Let $((m'_j, \mathcal{P}'_j)_{\forall j\in U}, (\mathsf{ek}'_j)_{\forall j\in U\setminus\{1\}}, \sigma')$ be a AVERO query such that $\mathcal{P}'_1$ is well defined in $\mathcal{T}$ where $\mathcal{P}'_1 = \left(f'_1, (\tau'_{1,i})_{\forall i\in[n]}\right)$ where we recall that n is the arity of the program $\mathcal{P}$.

CASE 1: First let us consider the case when $\forall i \in [n]$, $(\tau'_{1,i}, m'_{1,i}, \sigma'_{1,i}) \in \mathcal{T}$ st $m'_1 = f'_1\left((m'_{1,i})_{\forall i\in[n]}\right)$. This means that a tag has already been generated for all the initial messages that formed the homomorphic message m'_1. Recall that challenger computes $\bar{\Lambda}_1$ by running Eval algorithm on f'_1 and $(\sigma'_{1,i})_{\forall i\in[n]}$ under ek_1. Let $(\rho_l)_{\forall l\in U}$ be the value computed by the AggVer algorithm. Challenger retrieves the outputs of RO for each of $(m'_j)_{\forall j\in U}$. It thus obtains the $|U|$ corresponding tuples $\langle m'_i, w_i, b_i\rangle_{\forall i\in U}$ from the $\mathcal{H}$ list where $w_i = g_2^{b_i}$. If $m'_i \notin \mathcal{H}$ then challenger samples $b_i \leftarrow_{\$} \mathbb{Z}_p$, computes $w_i = g_2^{b_i}$. It then stores all these in $\mathcal{H}$. Using these he extracts the claimed MAC on message m'_1 as the output of $\mathcal{P}'_1$. It uses equation 16 for this. Let us call this claimed MAC as B. He also computes $\mathrm{e}(\bar{\Lambda}_1, w_1)$ where $w_1 = g_2^{b_1}$. So it is equivalent to $\mathrm{e}(\bar{\Lambda}_1{}^{b_1}, g_2)$.

- CASE A: if $\mathrm{e}(\bar{\Lambda}_1{}^{b_1}, g_2) = B$ then the answer is correct by the correctness of the scheme. Therefore σ' is accepted.
- CASE B: If $\mathrm{e}(\bar{\Lambda}_1{}^{b_1}, g_2) \neq B$: The challenger returns reject and computes $(\rho_l)_{\forall l\in U}$ by the internal AggVer algorithm to check equation 7. Also recall that $\forall l \in U$ evaluation key is $(u_l^{x^k_{l_0}})_{\forall k\in[0,D]} \leftarrow \mathsf{ek}_l$ where x_{l_0} is contained in the sk_l. We reconstruct back the recomputed aggregate MAC $\bar{\sigma}'$ as $\bar{\sigma}' \leftarrow \mathrm{e}(\bar{\Lambda}_1{}^{b_1}, g_2)\cdot\left(\Pi_{\forall l\in U\setminus\{1\}}\mathrm{e}(u_l, w_l^{\rho_l})e(u_l, w_l^{m'_l})^{-1}\right)$. Intuitively observe that $(\rho_l)_{\forall l\in U}$ is the same while running both $\mathsf{AggVer}(\ldots, \bar{\sigma}')$ on the recomputed HA-MAC and $\mathsf{AggVer}(\ldots, \sigma')$ on the claim HA-MAC. By the correctness of $\bar{\sigma}'$, the equality check of $\mathsf{AggVer}(\ldots, \bar{\sigma}') = 1$ translates to the check of whether the following equality holds
$e(\bar{\Lambda}_1, w_1)\left(\Pi_{\forall l\in U\setminus\{1\}}\mathrm{e}(u_l, w_l^{\rho_l})e(u_l, w_l^{m'_l})^{-1}\right) = \left(\Pi_{\forall l\in U}\mathrm{e}(u_l, w_l^{\rho_l})e(u_l, w_l^{m'_l})^{-1}\right)$.
Cancelling on both side give us the following

$$\mathsf{AggVer}(\ldots, \bar{\sigma}') = 1 \implies \mathrm{e}(\bar{\Lambda}_1, w_1)\mathrm{e}(u_1, w_1^{n\bar{m}_1}) = \mathrm{e}(u_1, w_1^{\rho_1}) \tag{29}$$

Similarly by the correctness of σ', the equality check of $\mathsf{AggVer}(\ldots, \sigma') = 1$ translates to checking whether the following equality holds
$B\left(\Pi_{\forall l\in U\setminus\{1\}}\mathrm{e}(u_l, w_l^{\rho_l})e(u_l, w_l^{m'_l})^{-1}\right) = \left(\Pi_{\forall l\in U}\mathrm{e}(u_l, w_l^{\rho_l})e(u_l, w_l^{m'_l})^{-1}\right)$. Cancelling on both side give us the following

$$\mathsf{AggVer}(\ldots, \sigma) = 1 \implies B\mathrm{e}(u_1, w_1^{m'_1}) = \mathrm{e}(u_1, w_1^{\rho_1}) \tag{30}$$

Since RHS of both equations 29 and 30 is the same hence $B\mathrm{e}(u_1, w_1^{m'_1})^{-1} = \mathrm{e}(\bar{\Lambda}_1, w_1)\mathrm{e}(u_1, w_1^{\bar{m}_1})^{-1}$ must hold. Hence returning 1 iff $Z_A \leftarrow \frac{B}{\mathrm{e}(\bar{\Lambda}_1{}^{b_1}, g_2)}\mathrm{e}(u_1^{b_1}, g_2)^{(m'_1 - \bar{m}_1)} = 1$ is the same as returning the output of AggVer on input σ' where $w_1 \leftarrow g_2^{b_1}$.

CASE II:We now consider the case of $\mathcal{P}'_1$ is well defined on $\mathcal{T}$ but $\exists i \in [n]$ such that $\tau'_{1,i} \notin \mathcal{T}$. By the definition of well defined program this means that if we fix the input values of all wires labelled with $\tau'_{1,i}$ where $(\tau'_{1,i}, \cdot, \cdot) \in \mathcal{T}$ then the circuit f'_1 always returns the same output irrespective of the values of the wires labelled $\tau'_{1,i}$ such that $\tau'_{1,i} \notin \mathcal{T}$ for some $i \in [n]$. Recall from section 3.2 that the circuit f'_1 is once computed during Eval algorithm to get $\bar{\Lambda}_1$ as well as when f'_1 is evaluated during Ver algorithm to compute ρ_1. The earlier statement means that for both of these operations on f'_1 these labels $\tau'_{1,i}$ such that $\tau'_{1,i} \notin \mathcal{T}$ for some $i \in [n]$ do not affect the output of the circuit f'_1. This means that for all of these labels the dummy tags chosen do not contribute to the computation of $\bar{\Lambda}_1$ as well as in the computation of ρ_1. This asserts that the above analysis of CASE 1 holds in the CASE II too. □

Lemma 9. $\Pr[\eta_5] < e^{\frac{Q_A+Q_V+Q_{AV}+P}{Q_A+Q_V+PQ_{AV}}} \cdot (Q_A + Q_V + PQ_{AV} - 1) \cdot \mathbf{Adv}^{co-CDH}_{\Theta,\mathrm{C}}$ *where* C *is any PPT adversary against the* (t', ϵ') *security of co-CDH over* $(\mathcal{G}_1, \mathcal{G}_2)$ *and* e *is the base of the natural log.*

Proof (Proof of Lemma 9).

If there exists an adversary $\mathcal{A}(t, Q_H, Q_A, Q_V, Q_{AV}, P, \epsilon)$ that makes a forgery query that is a query that triggers the event η_5 in AVERO with probability at least ϵ. Then we show that there exists an adversary C that breaks the co-CDH with probability at least ϵ'. Because co-CDH is a computationally hard problem, the above implies that $\Pr[\eta_5] < \epsilon$.

In other words, using $\mathcal{A}$ we show the construction of adversary C(Figure 10) that solves co-CDH in $(\mathcal{G}_1, \mathcal{G}_2)$ with probability at least ϵ' and running time at most t'. C simulates the game $\mathbf{G}^5_{\Theta,\mathcal{A}}$ and answers queires of $\mathcal{A}$ as follows. For concrete details of the construction of C we recall the construction of $\mathcal{B}$ from the proof of lemma 6. We also recall from the proof of Lemma 6 that the below analysis is under the scenario that all incoming messages to any oracle gets registered to the $\mathcal{H}$ list i.e., gets a c coin allocated to it. [5]

In the start, C upon given $g_2, g_2^a \in \mathcal{G}_2, h \in \mathcal{G}_1$ such that $\psi(g_2) = g_1$, the goal of C can be interpreted as that of outputting h^a. We note below the key distinctions in the construction of C from that of $\mathcal{B}$.

The change in the AVERO oracle from $\mathbf{G}^4_{\Theta,\mathcal{A}}$ to $\mathbf{G}^5_{\Theta,\mathcal{A}}$ is as follows. When $c_1 = 1$, C uses the keys he generated and simulates answers to the AUTHO, VERO, AVERO identical to $\mathbf{G}^5_{\Theta,\mathcal{A}}$. Meaning conditioned on the bit c_1 is not set to 0 the output distribution of all three of these oracles are distributed identical

[5] In what follows in lemmas 9 through 10, we retain the subscript i of coin c for authentication under $\mathsf{ek}_i, \mathsf{sk}_i$ for some $i \in P$. However we drop the superscript denoting the query number unless otherwise needed.

to $\mathbf{G}^5_{\Theta,\mathcal{A}}$. In other words, particularly the output distribution of AVERO as seen by $\mathcal{A}$ when run by C is the same as seen by $\mathcal{A}$ when run by the challenger of $\mathbf{G}^5_{\Theta,\mathcal{A}}$ given that the message for which the c bit is set is not queried for any party $(p_l)_{\forall l \in U}$.

When $c_1 = 0$, C aborts in AUTHO, VERO, AVERO. In the case where $\mathcal{A}$ does not make the forgery query such that event η_5 occurs with probability at least ϵ then $\mathcal{A}$ halted and conceded failure. In this case C halts and concedes failure too. Next, consider the scenario that $\mathcal{A}$ made the query in AVERO such that event η_5 occurs with probability at least ϵ. Let $((m^*_j, \mathcal{P}^*_j)_{\forall j \in U}, (\mathsf{ek}^*_j)_{\forall j \in U \setminus \{1\}}, \sigma^*)$ be that query triggering event η_5. We note that this is the query that gets accepted in $\mathbf{G}^4_{\Theta,\mathcal{A}}$ but not in $\mathbf{G}^5_{\Theta,\mathcal{A}}$. Therefore we call this a *forgery query*. Therefore it can be interpreted as equivalent to $\mathcal{A}$ halted with success.

C now proceeds if $c_1 = 0$ and $\forall i \in U \setminus \{1\}, c_i = 1$. Otherwise C declares failure and halts. We recall that at $c_1 = 0$ hash of message m^*_1 is $w_1 = g_2^{a+b_1}$ and hash of message $(m^*_i)_{\forall i \in U \setminus \{1\}}$ is $w_i = g_2^{b_i}$ for some $b_i \leftarrow\!\!\$\ \mathbb{Z}_p$.

Now we analyze how the C runs in the OUT procedure to extract the correct answer to be returned to his own challenger given that event η_5 has happened. We recall that σ^* is in the format of Λ^* denoting a homomorphically evaluated Mac. On input $((m^*_j, \mathcal{P}^*_j)_{\forall j \in U}, (\mathsf{ek}^*_j)_{\forall j \in U \setminus \{1\}}, \sigma^*)$ to OUT, it proceeds as follows. $\mathcal{A}$ must not have queried the AUTHO on m^*_1. Algorithm C now runs its own hash algorithm at each m^*_i $\forall i \in U$ obtaining $|U|$ corresponding tuples $\langle m^*_i, w_i, b_i, c_i \rangle$ on the $\mathcal{H}$ list.

Using the internal AggVer algorithm, C also computes the $(\rho_i)_{\forall i \in U}$. The claimed aggregate MAC σ^* must satisfy equation 7 as recalled below

$$\sigma^* = \Pi_{\forall i \in U}\, \mathrm{e}(u_i, w_i)^{\rho_i - m^*_i} \tag{31}$$

For each $i > 1$, $\mathrm{e}(\Lambda^*_i, w_i) = \mathrm{e}(u_i, w_i)^{\rho_i - m^*_i}$ must hold according to equation 1. This is because we are analyzing the AVERO query that was accepted in $\mathbf{G}^4_{\Theta,\mathcal{A}}$.

Since event η_5 has occurred one of the things we know is that the MAC for each of the users in U must verify in $\mathbf{G}^4_{\Theta,\mathcal{A}}$. Therefore for each $i > 1$, C substitutes the MACs as $\mathrm{e}(u_i, w_i)^{\rho_i - m^*_i}$. This is because Λ^*_i is a valid homomorphic mac on m^*_i whose hash is w_i and whose evaluation key component is u_i. C extracts mac B under challenge key pair and hash w_1 as $\mathrm{e}(\Lambda^*_1, w_1)$

$$\mathrm{e}(\Lambda^*_1, w_1) = \sigma^* \cdot \left(\Pi_{\forall i \in U \setminus \{1\}}\, \mathrm{e}\left(\Lambda^*_i, w_i\right)\right)^{-1} = \sigma^* \cdot \left(\Pi_{\forall i \in U \setminus \{1\}}\, \mathrm{e}\left(u_i, w_i\right)^{(\rho_i - m^*_i)}\right)^{-1} \tag{32}$$

We recall that $\mathrm{e}(\Lambda^*_1, w_1)$ is the claimed mac on m^*_1 submitted by $\mathcal{A}$ under $(h^{x^k_{1_0}})_{\forall k \in D} \leftarrow \mathsf{ek}^*_1$ as the challenge evaluation key where x_{1_0} is in sk^*_1 using $g_2^{a+b_1}$ as the hash of message m^*_1.

Recall that $\mathcal{P}^*_1$ is well defined in $\mathcal{T}$ such that $\forall i \in [n], (\tau^*_{1,i}, \cdot, \cdot) \in \mathcal{T}$. Let $\forall i \in [n], (\tau^*_{1,i}, m_{1,i}, \sigma_{1,i})$ be the corresponding entries in $\mathcal{T}$, such that $m^*_1 = f^*_1\left((m_{1,i})_{\forall i \in [n]}\right)$. Challenger computes $(y_{1,k})_{\forall k \in [0,D]}$ using the internal Eval algorithm, meaning it computes $(y_{1,k})_{k \in [0,D]} \leftarrow f^*_1\left((\sigma_{1,i})_{\forall i \in [n]}\right)$. Eventually he

computes $\mathrm{e}(\bar{\Lambda}_1, w_1)$ as $\mathrm{e}(h^{\Sigma_{k=1}^{D} y_{1,k} x^k}, g_2^{a+b_1})$. For party p_1 he also computes $\bar{m}_1$ as the homomorphic message as output of f_1^* on input the messages from the table for the submitted input labels.

Because event η_5 has occurred in $\mathbf{G}^5_{\Theta,\mathcal{A}}$ we further know that the ratio between claimed MAC under $\mathsf{ek}_1^*, \mathsf{sk}_1^*$ that is Λ_1^* and the corresponding recomputed MAC $\bar{\Lambda}_1$ fulfill the conditions required to initiate event η_5. From Figure 8 we recall below the expression that holds given that event η_5 has occurred.

$$\mathrm{e}(\bar{\Lambda}_1, w_1) = \mathrm{e}(\Lambda_1^*, w_1)\mathrm{e}(g_1, w_1)^{m_1^* - \bar{m}_1} \tag{33}$$

Moreover, conditioned that $c_1 = 0$, we recall hash of message m_1^* is $w_1 = g_2^{a+b_1}$. The equation 33 then becomes the following

$$\mathrm{e}(h^{(a+b_1)\left(\Sigma_{k=1}^{D} x_{1_0}^k - \rho_1^* + m_1^*\right)}, g_2) = \mathrm{e}(g_1, g_2^{(a+b_1)(m_1^* - \bar{m}_1)}) \tag{34}$$

Using equation 3 , C correctly returns solution to his co-CDH challenge (h^a) as

$$(\psi(g_2^{(a+b_1)})^{(m_1^* - \bar{m}_1)})^{\left(m_1^* - \rho_1 + (\Sigma_{k=1}^{D} y_{1,k} x_{1_0}^k)\right)^{-1}} \cdot (h^{b_1})^{-1} \tag{35}$$

After executing the Out procedure C aborts, so does $\mathcal{A}$. We note that in the event that forgery query has not been made η_5 has not occurred. We further prove in Claim B6 that view of $\mathcal{A}$ remains consistent in this case.

Now it remains to show that C solves the given instance of the co-CDH problem in $(\mathcal{G}_1, \mathcal{G}_2)$ with probability at least ϵ'. In order to do so consider the following events.

- Let $\mathcal{E}_1$ be the event that C does not abort in Auth queries.
- Let $\mathcal{E}_2$ be the event that C does not abort in Ver oracle.
- Let $\mathcal{E}_3$ be the event that C does not abort in AggVer oracle.
- Let $\mathcal{E}_4$ be the event that valid forgery query has taken place.
- Let $\mathcal{E}_5$ be the event that event $\mathcal{E}_4$ has taken place and $c_1 = 0$ and $\forall l \in U \setminus \{1\}$, $c_l = 1$ where c_l is the component of the tuple m_l on the $\mathcal{H}$ list.

We define C wins if all these events take place.

$$\begin{aligned}&\Pr[\mathcal{E}_1 \wedge \mathcal{E}_2 \wedge \mathcal{E}_3 \wedge \mathcal{E}_5] = \\ &\Pr[\mathcal{E}_1 \wedge \mathcal{E}_2 \wedge \mathcal{E}_3] \cdot \Pr[\mathcal{E}_4 | \mathcal{E}_1 \wedge \mathcal{E}_2 \wedge \mathcal{E}_3] \cdot \Pr[\mathcal{E}_5 | \mathcal{E}_1 \wedge \mathcal{E}_2 \wedge \mathcal{E}_3 \wedge \mathcal{E}_4].\end{aligned} \tag{36}$$

The following Claims B4, B5, B6 give the lower bound for each of these terms.

Claim B4 *The probability that algorithm* C *does not abort due to the* Auth, Ver *and* AggVer *queries made by any adversary* $\mathcal{A}$ *in* $\mathbf{G}^5_{\Theta,\mathcal{A}}$ *is at least* $(1 - \frac{1}{Q_A + Q_V + PQ_{AV}})^{Q_A + Q_V + Q_{AV}}$*, where* Q_A, Q_V, Q_{AV} *represents the number of* AuthO, VerO *and* AVerO *queries made by* $\mathcal{A}$ *and* P *represents the total number of users. Therefore,* $\Pr[\mathcal{E}_1 \wedge \mathcal{E}_2 \wedge \mathcal{E}_3] \geq (1 - \frac{1}{Q_A + Q_V + PQ_{AV}})^{(Q_A + Q_V + Q_{AV})}$.

Proof (Proof of Claim B4).

Assume without loss of generality that $\mathcal{A}$ does not request AVERO query on the same message twice. We recall that AVERO calls RO by construction. So we are also assuming that RO is not queried on the same message twice. We will show by induction that after $\mathcal{A}$ makes l AVERO queries the probability that C does not abort due to the AVERO queries is at least $\left(1 - \frac{1}{Q_A+Q_V+PQ_{AV}}\right)^l$. The base case for $l = 0$ is trivial. We define the superscript notation $(m^{(l)}, w^{(l)}, b^{(l)}, c^{(l)})$ with respect to l^{th} query in any of Auth, Ver or AggVer oracles. We recall that the following proof is similar to proof of Claim B1 and we include here for completeness.

Let $(m_j^{(l)}, \mathcal{P}_j^{(l)}, \sigma'^{(l)})_{\forall j \in U}$ be $\mathcal{A}$'s l^{th} aggregate verification query and let $(m_1^{(l)}, w_1^{(l)}, b_1^{(l)}, c_1^{(l)})$ be the corresponding tuple on the $\mathcal{H}$ list for the message $m_1^{(l)}$ that is authenticated under challenge key $\mathsf{ek}_1^*, \mathsf{sk}_1^*$. [6] Before $\mathcal{A}$ makes this query to the AVERO, the bit $c^{(l)}$ is independent of $\mathcal{A}$'s view. The only value potentially dependent on $c^{(l)}$ that could have been given to $\mathcal{A}$ is $\mathcal{H}(m_1^{(l)})$ which has the same distribution whether $c^{(l)} = 0$ or $c^{(l)} = 1$. This is because the output of RO in both cases is a random element in $\mathcal{G}_2$. Therefore, the probability that this query causes C to abort is at least $\frac{1}{Q_A+Q_V+PQ_{AV}}$. Using inductive hypothesis and independence of bit c, the probability that C does not abort after this query is at least $(1 - \frac{1}{Q_A+Q_V+PQ_{AV}})^l$. Since $\mathcal{A}$ makes at most Q_{AV} AVERO queries, the probability that C does not abort due to all AVERO queries is at least $(1 - \frac{1}{Q_A+Q_V+PQ_{AV}})^{Q_{AV}}$.

We recall here again that we are arguing unforgeability of AVERO oracle. This necessarily encompasses VERO and AUTHO. This is because the message for which the bit c is set to 0 can be queried to any of these oracles and thereby used to forge AVERO. Therefore we consider the following now.

Analogously,the probability that C does not abort due to any of the VERO queries is at least $(1 - \frac{1}{Q_A+Q_V+PQ_{AV}})^{Q_V}$. Also, the probability that C does not abort due to any of the Auth queries is at least $(1 - \frac{1}{Q_A+Q_V+PQ_{AV}})^{Q_A}$. Multiplying this concludes our claim that probability that C did not abort in Auth, Ver and AggVer queries. Thus, we have
$\Pr[\mathcal{E}_1 \wedge \mathcal{E}_2 \wedge \mathcal{E}_3] \geq (1 - \frac{1}{Q_A+Q_V+PQ_{AV}})^{Q_A+Q_V+Q_{AV}}$.

□

Claim B5 *If algorithm* C *does not abort due to the queries made by* $\mathcal{A}$ *in* $\mathbf{G}^5_{\Theta,\mathcal{A}}$*, then the view of algorithm* $\mathcal{A}$ *in the protocol is identical to its view when run by* C*. Hence,* $\Pr[\mathcal{E}_4|\mathcal{E}_1 \wedge \mathcal{E}_2 \wedge \mathcal{E}_3] \geq \epsilon$.

Proof (Proof of Claim B5). The evaluation keys and the secret keys provided to $\mathcal{A}$ in $\mathbf{G}^5_{\Theta,\mathcal{A}}$(Figure 8) has the same distribution as that of the keys given to $\mathcal{A}$ when played by C(Figure 10). Moreover, the responses to hash queries as seen by $\mathcal{A}$ when run by C are uniformly and independently distributed in $\mathcal{G}_2$. Therefore it is the same as the distribution seen by $\mathcal{A}$ when it runs in the real protocol.

[6] What follows ahead, for simplicity we are omitting the subscript 1 of coin of l^{th} AVERO query $c^{(l)}$.

Since C does not abort due to $\mathcal{A}$'s authentication, verification, and aggregate verification queries, all responses to those queries are valid. In other words, prior to this, $\mathcal{A}$ did not make the forgery query in any of the oracles. Therefore, the probability of event $\mathcal{E}_4$ (forgery) given events $\mathcal{E}_1$, $\mathcal{E}_2$, and $\mathcal{E}_3$ have occurred is at least ϵ. In other words the probability that $\mathcal{A}$ made a forgery query in AVERO such that it was accepted in $\mathbf{G}^4_{\Theta,\mathcal{A}}$ but rejected in $\mathbf{G}^5_{\Theta,\mathcal{A}}$ where $\mathcal{P}'_1$ is well defined on $\mathcal{T}$ is ϵ. More concretely, $\Pr[\mathcal{E}_4|\mathcal{E}_1 \wedge \mathcal{E}_2 \wedge \mathcal{E}_3] \geq \epsilon$. □

Claim B6 *The probability of algorithm* C *not aborting after* $\mathcal{A}$ *outputs a valid and nontrivial agggregate verification forgery query in* $\mathbf{G}^5_{\Theta,\mathcal{A}}$ *is at least:*
$\left(1 - \frac{1}{Q_A+Q_V+PQ_{AV}}\right)^{P-1} \frac{1}{Q_A+Q_V+PQ_{AV}}$.
Meaning, $\Pr[\mathcal{E}_5|\mathcal{E}_1 \wedge \mathcal{E}_2 \wedge \mathcal{E}_3 \wedge \mathcal{E}_4] \geq \left(1 - \frac{1}{Q_A+Q_V+PQ_{AV}}\right)^{P-1} \frac{1}{Q_A+Q_V+PQ_{AV}}$.

Proof (Proof of Claim B6). [7]

We are given that the events $\mathcal{E}_1$, $\mathcal{E}_2$, $\mathcal{E}_3$ and $\mathcal{E}_4$ have occurred, and $\mathcal{A}$ has produced a valid and non-trivial forgery query such that $c_1 = 0$ and $\forall i \in U \backslash \{1\}, c_i = 1$. Let $((m^*_j, \mathcal{P}^*_j)_{\forall j \in U\backslash\{1\}}, (\mathsf{ek}^*_j)_{\forall j \in U\backslash\{1\}}, \sigma^*)$ be the forgery query, meaning query triggering event η_5. We note that $\forall i \in U \setminus \{1\}, c_i$ is independent of each other. For each m^*_i, where $i \in U \setminus \{1\}$, let $\langle m^*_i, w_i, b_i, c_i\rangle$ be the corresponding tuple retrieved from the $\mathcal{H}$-list. So, output distribution of RO is independent of programmed bit c.

Algorithm C will abort unless $\mathcal{A}$ generates a forgery such that $c_1 = 0$ and for $i > 1$, $c_i = 1$. For concrete analysis we use P instead of $|U|$. Since the forgery is non-trivial we know that $\mathcal{A}$ could not have queried AUTHO, VERO, AVERO on m^*_1 under ek^*_1 and sk^*_1. Recall that for a well defined program $\mathcal{P}^*_1$ there must exist $m_1, \ldots, m_n \in \mathcal{T}$ such that $m^*_1 = f^*_1(m_1, \ldots, m_n)$ under the challenge evaluation key ek^*_1. Nevertheless, $\mathcal{A}$ has no information about the value of c_1. This is because $c_1 = 0$ occurs with probability $\frac{1}{(Q_A+Q_V+PQ_{AV})}$.

For each $i > 1$, two cases arise for $\mathcal{A}$. If $\mathcal{A}$ requested a authenticator under key ek^*_1 on m^*_i or queried VERO on input m^*_i then $\Pr[c_i = 1] = 1$. On the other hand if $\mathcal{A}$ did not query the RO or AUTHO or VERO on m^*_i for some $i \in P$ then $\Pr[c_i = 1] = (1 - \frac{1}{(Q_A+Q_V+PQ_{AV})})$. The probability that $c_i = 1$, $\forall i \in [2, P]$, is at least $(1 - \frac{1}{(Q_A+Q_V+PQ_{AV})})^{P-1}$. Therefore, $\Pr[\mathcal{E}_5|\mathcal{E}_1 \wedge \mathcal{E}_2 \wedge \mathcal{E}_3 \wedge \mathcal{E}_4] \geq (1 - \frac{1}{(Q_A+Q_V+PQ_{AV})})^{P-1} \cdot \frac{1}{(Q_A+Q_V+PQ_{AV})}$, as required.

We substitute back the results of Claims B4, B5, B6 into equation 36 of Lemma 9. By contradiction algorithm C breaks co-CDH with probability at least ϵ'. Hence we concretely get the following

$$\left(1 - \frac{1}{Q_A + Q_V + PQ_{AV}}\right)^{Q_A+Q_V+Q_{AV}+P-1} \cdot \frac{1}{Q_A + Q_V + PQ_{AV}} \cdot \epsilon \geq \epsilon' \quad (37)$$

[7] In the following proofs, for simplicity of presentation, we further drop from coin c the superscript l denoting the l^{th} aggregate verification oracle query.

Using taylor series approximation $(1-\frac{1}{x})^x \geq \frac{1}{e}$ we get the following

$$\frac{\epsilon}{e^{\frac{Q_A+Q_V+Q_{AV}+P}{Q_A+Q_V+PQ_{AV}}} \cdot (Q_A+Q_V+PQ_{AV}-1)} \geq \epsilon' \tag{38}$$

Therefore the following concludes the proof of lemma 9.

$$\Pr[\eta_5] < e^{\frac{Q_A+Q_V+Q_{AV}+P}{Q_A+Q_V+PQ_{AV}}} \cdot (Q_A+Q_V+PQ_{AV}-1) \cdot \mathbf{Adv}^{co-CDH}_{\Theta,\mathcal{A}}$$

□
□

Lemma 10. *Let $(\mathcal{G}_1, \mathcal{G}_2)$ be a bilinear group pair for co-Diffie-Hellman. For any PPT adversary $\mathcal{A}(t, Q_H, Q_V, Q_A, Q_{AV}, \epsilon)$ that makes an* AVERO *forgery query in the $\mathbf{G}^5_{\Theta,\mathcal{A}}$ with probability at least ϵ and running time at most t then there exists another PPT adversary* C *against co-CDH assumption with probability at least ϵ' and running time at most t' such that the following two inequalities hold*

$$\epsilon' > \frac{\epsilon}{e^{\frac{Q_A+Q_V+Q_{AV}+P}{Q_A+Q_V+PQ_{AV}}} \cdot (Q_A+Q_V+PQ_{AV}-1)}$$

$$\begin{aligned} t' < t + c_{\mathcal{G}_1}[Q_V] + c_{\mathcal{G}_2}[Q_H + Q_A + 3Q_V + Q_{AV}(3P) + 2P + 2] + \\ c_{\mathcal{G}_T}[2Q_{AV} + 5P + 3] + c_{\mathbb{Z}}[3Q_{AV} + |f|Q_V + P|f|Q_{AV} + 1] + \\ c_{\mathrm{e}}[3Q_V + PQ_{AV} + 2P + 3)] \end{aligned}$$

Proof (Proof of lemma 10).

By Lemma 9 and Claims till B6, we have proven that given an adversary $\mathcal{A}$ who breaks Θ by making an aggregate verification oracle forgery query in $\mathbf{G}^5_{\Theta,\mathcal{A}}$ with probability at least ϵ then there exists adversary C that breaks co-CDH with probability at least ϵ' such that $\epsilon' > \frac{\epsilon}{e^{\frac{Q_A+Q_V+Q_{AV}+P}{Q_A+Q_V+PQ_{AV}}} \cdot (Q_A+Q_V+PQ_{AV}-1)}$. Now it remains to show, given that there exists $\mathcal{A}$ that takes time at most t to break Θ then there exists C that takes time at most t' to break co-CDH.

Referring to Figure 2 the total running time of C consists of three main components: (1) The running time of algorithm $\mathcal{A}$, since C invokes $\mathcal{A}$ to receive queries, (2) The time to respond to $(Q_H+Q_A+Q_V+PQ_{AV})$ hash queries, each requiring an exponentiation in $\mathcal{G}_2$, (3) The time to answer Q_A Auth queries, Q_V Ver queries, Q_{AV} AggVer queries (4) the time to transform $\mathcal{A}$'s final forgery into the co-CDH solution.

Let $c_{\mathcal{G}_1}$, $c_{\mathcal{G}_2}$, $c_{\mathcal{G}_T}$, $c_{\mathbb{Z}}$, c_{e} denote the time for group operations in $\mathcal{G}_1$, $\mathcal{G}_2$, $\mathcal{G}_T$, $\mathbb{Z}_p$, compute map e. For simplicity PRF computation cost is not there in the analysis.

- $(Q_H+Q_A+Q_V+PQ_{AV})$ Hash queries: each hash query has 1 exponentiation in $\mathcal{G}_2$. RO cost = $c_{\mathcal{G}_2}(Q_H+Q_A+Q_V+PQ_{AV})$
- Q_A Auth queries: Each auth query makes 2 operations in $\mathbb{Z}_p$. AUTHO cost = $3c_z(Q_A)$

- Q_V Ver queries: Each VERO query does $|f|$ no of operations in $\mathbb{Z}_p$ where f is the function submitted. Depending on whether the Ver oracle gets queried on initial MAC or homomorphic MAC, either 2 exponentiations in $\mathcal{G}_2$ +3 e operations + 1 multiplication in $\mathcal{G}_T$ are performed OR 2 exponentiations in $\mathcal{G}_2$ + 3 e operations + 1 multiplication in $\mathcal{G}_T$ + 1 exponentiations in $\mathcal{G}_1$. Taking the costlier operation for concrete calculation we get that total VERO cost = $Q_V(c_z|f| + 2c_{\mathcal{G}_2} + 3c_{\mathrm{e}} + c_{\mathcal{G}_T} + c_{\mathcal{G}_1})$.
- Q_{AV} AggVer queries: Each AVERO query has $P|f|$ no of operations in $\mathbb{Z}_p$ + $2P$ exponentiations in $\mathcal{G}_2$ + P maps + $(1 + 2P)$ multiplications in $\mathcal{G}_T$. Therefore total AVERO cost = $Q_{AV}(c_{\mathbb{Z}}(P|f|) + c_{\mathcal{G}_2}(2P) + c_{\mathrm{e}}P + c_{\mathcal{G}_T})$.
- Forgery: The calculation of B takes $2P$ exponentiations in $\mathcal{G}_2$, $(1+P)$ inversions on $\mathcal{G}_T$, P multiplications in $\mathcal{G}_T$. This becomes $c_{\mathcal{G}_2}2P+(1+P)c_{\mathcal{G}_T}+(2+P)\mathcal{G}_T$. Moreover it needs $P|f|$ operations in $\mathbb{Z}_p$ for evaluating the function at chosen random evaluation points. Next it needs 1 exponentiation in $\mathcal{G}_2$ + 1 map for the conditional check. Then to compute the solution to challenge it needs 1 operation in $\mathbb{Z}_p$, 1 exponentiation in $\mathcal{G}_2$, 1 map computation. Total cost for forgery inversion is $(2P + 2)c_{\mathcal{G}_2} + (2P + 3)c_{\mathcal{G}_T} + (2P + 3)c_{\mathrm{e}}$.

Summing these components, the total running time of C is at most: $t + c_{\mathcal{G}_1}[Q_V] + c_{\mathcal{G}_2}[Q_H + Q_A + 3Q_V + Q_{AV}(3P) + 2P + 2] + c_{\mathcal{G}_T}[2(Q_{AV}) + 5P + 3)] + c_{\mathbb{Z}}[2Q_A + |f|Q_V + P|f|Q_{AV} + 1] + c_{\mathrm{e}}[3Q_V + P(Q_{AV}) + 2P + 3)]$. This concludes the proof of lemma 10. This concludes the proof that for any PPT adversary $\mathcal{A}$ in $\mathbf{G}^5_{\Theta,\mathcal{A}}$, aggregate verification oracle queries are unforgeable except with probability at least ϵ (as computed through Lemma 5 and Claims till B6) and running time at most t (as computed in Lemma 10). □

B.1 Summary

To conclude the proof of the theorem, we first sum up the results of all the lemmas and express ϵ as below

$$\mathbf{Adv}^{\mathrm{HA}}_{\Theta,\mathcal{A}}(1^\lambda, 1^n, 1^D, 1^P) < \frac{Q_V + Q_{AV}}{2^\lambda} + \mathbf{Adv}^{PRF}_{\mathrm{D},\mathcal{F}}(\lambda) + \frac{Q_V(D+1)}{p - D(Q_V - 1)} + \frac{DQ_{AV}}{p - D(Q_{AV} - 1)} + \frac{DQ_V^2}{p} + \frac{DQ_{AV}^2}{p} + e^{\frac{Q_A+Q_V+Q_{AV}}{Q_A+Q_V+PQ_{AV}}}(Q_A + Q_V + PQ_{AV}) \cdot \mathbf{Adv}^{co-CDH}_{\Theta,\mathcal{B}} e^{\frac{Q_A+Q_V+Q_{AV}+P}{Q_A+Q_V+PQ_{AV}}} \cdot (Q_A + Q_V + PQ_{AV} - 1) \cdot \mathbf{Adv}^{co-CDH}_{\Theta,\mathrm{C}}$$

In the above we perform the following substitutions (i)Replace by Q the maximum of Q_H, Q_A, Q_V, Q_{AV} (ii) $\mathbf{Adv}^{co-CDH}_{\Theta,\mathcal{B}}(1^\lambda, 1^n, 1^D, 1^P)$ and $\mathbf{Adv}^{co-CDH}_{\Theta,\mathrm{C}}(1^\lambda, 1^n, 1^D, 1^P)$ are both replaced by ϵ' as defined prior (iii) Replacing $\mathbf{Adv}^{PRF}_{\mathrm{D},\mathcal{F}}(\lambda)$ as ϵ'' (iii) losing constant coefficients (iv)$P + 2 \approx P$ (vv) Substitute $\mathbf{Adv}^{\mathrm{HA}}_{\Theta,\mathcal{A}}(1^\lambda, 1^n, 1^D, 1^P)$ as ϵ. Hence we get the following precise ex-

pression

$$\epsilon < \frac{Q}{2^{\lambda}} + \frac{DQ}{p - DQ} + \frac{DQ^2}{p} + e^{\frac{3}{P}} PQ \left(e^{\frac{1}{Q}} + 1 \right) \epsilon' + \epsilon'' \tag{39}$$

In order to get the exact expression for running time t of any ppt adversary against Θ, we sum up the lemmas 7 and 10 above and perform the following substitutions (i)maximum of $c_{\mathbb{Z}}, c_{\mathcal{G}_1}, c_{\mathcal{G}_2}, c_{\mathcal{G}_T}, c_{\mathrm{e}}$ as c (ii)Replace by Q the maximum of Q_H, Q_A, Q_V, Q_{AV} (iii) size of f is replaced by s. Therefore we get the below:

$$\begin{aligned} &t + c(Q + sQ + sPQ + s + 1) + cQ + c(Q + PQ) + cQ + c(Q + PQ) + cQ + \\ &c(Q + PQ + P) + c(Q + P) + c(Q + sQ + sPQ) + c(Q + PQ + P) < t' \\ &\implies t + c(s + Q + P + sQ + PQ + sPQ) > t' \end{aligned} \tag{40}$$

Since both D, P and Q is $poly(\lambda)$ and ϵ' and ϵ'' are negligible as defined in equation 4 therefore equation 39 evaluates to negligible. Meaning $\mathcal{A}$ has at most negligible advantage of breaking the unforgeability of our construction Θ. Similarly from equation 40 the total running time of $\mathcal{A}$ evaluates to approximately running time of solving co-CDH. This conlcudes the proof of theorem 1.

Game $\mathbf{G}^0_{\Theta,\mathcal{A}}(1^\lambda, 1^n, 1^D, 1^P)$

INITIALIZE$(1^\lambda, 1^n, 1^D, 1^P)$:
1 $\mathcal{T} \leftarrow \emptyset$
2 $(\mathsf{sk}_l, \mathsf{ek}_l)_{\forall l \in P} \leftarrow\!\!\$\ \mathsf{KeyGen}(1^\lambda, 1^n, 1^D, 1^P)$
3 Return ek_1, $(\mathsf{ek}_l, \mathsf{sk}_l)_{\forall l \in [2,P]}$

RO(y):
4 same as real

AUTHO(τ, m):
5 if $(\tau, m, \cdot) \in \mathcal{T}$, $\sigma \leftarrow \mathcal{T}(\tau, m, \cdot)$
6 if $(\tau, m, \cdot) \notin \mathcal{T}$,$\sigma \leftarrow \mathsf{Auth}_{\mathsf{sk}_1}(\tau, m)$; $\mathcal{T} = \mathcal{T} \cup (\tau, m, \sigma)$
7 if $(\tau, \cdot, \cdot) \notin \mathcal{T}$, ignore
8 Return σ

VERO$(m, \mathcal{P}, \sigma)$:
9 sample $\lambda + 1$ tuples $\{U_i\}_{i=0}^{\lambda}$ st $U_{i_{\forall i \in [0,\lambda]}} \leftarrow\!\!\$\ \mathbb{Z}_p^n$
10 Return 1 iff $\forall i \in [1, \lambda], f(U_0) = f(U_i)$

AVERO$((m'_j, \mathcal{P}'_j)_{\forall j \in U}, (\mathsf{ek}'_j)_{\forall j \in U \setminus \{1\}}, \sigma')$:
11 sample $\lambda + 1$ tuples $\{U_i\}_{i=0}^{\lambda}$ st $U_{i_{\forall i \in [0,\lambda]}} \leftarrow\!\!\$\ \mathbb{Z}_p^n$
12 Return 1 iff $\forall i \in [1, \lambda], f'_1(U_0) = f'_1(U_i)$

FINALIZE(IN):
13 if $IN = (m^*_j, \mathcal{P}^*_j)_{\forall j \in U}, (\mathsf{ek}^*_j)_{\forall j \in U \setminus \{1\}}, \sigma^*)$
14 Return $\mathsf{Check}_A((m^*_j, \mathcal{P}^*_j)_{\forall j \in U}, (\mathsf{ek}^*_j)_{\forall j \in U \setminus \{1\}}, \sigma^*))$
15 else if $IN = (m^*, \mathcal{P}^*, \sigma^*)$
16 Return $\mathsf{Check}_V(m^*, \mathcal{P}^*, \sigma^*)$

Fig. 3: Game 0 for security proof for Θ

Game $\mathbf{G}^1_{\Theta,\mathcal{A}}(1^\lambda, 1^n, 1^D, 1^P)$

INITIALIZE$(1^\lambda, 1^n, 1^D, 1^P)$:
1 $\mathcal{T} \leftarrow \emptyset$
2 $(\mathsf{sk}_l, \mathsf{ek}_l)_{\forall l \in P} \leftarrow\!\!\$\ \mathsf{KeyGen}(1^\lambda, 1^n, 1^D, 1^P)$
3 Return ek_1, $(\mathsf{ek}_l, \mathsf{sk}_l)_{\forall l \in [2,P]}$

RO(y):
4 same as before

AUTHO(τ, m):
5 if $(\tau, m, \cdot) \in \mathcal{T}$, $\sigma \leftarrow \mathcal{T}(\tau, m, \cdot)$
6 if $(\tau, m, \cdot) \notin \mathcal{T}$,$\sigma \leftarrow \mathsf{Auth}_{\mathsf{sk}}(\tau, m); \mathcal{T} = \mathcal{T} \cup (\tau, m, \sigma)$
7 In Auth, Replace PRF with TRF $\mathcal{R}$
8 if $(\tau, \cdot, \cdot) \notin \mathcal{T}$, ignore
9 Return σ

VERO$(m, \mathcal{P}, \sigma)$:
10 same as before

AVERO$((m'_j, \mathcal{P}'_j)_{\forall j \in U}, (\mathsf{ek}'_j)_{\forall j \in U \setminus \{1\}}, \sigma')$:
11 same as before

FINALIZE(IN):
12 same as before

Fig. 4: Game 1 for security proof for Θ

Game $\mathbf{G}^2_{\Theta,\mathcal{A}}(1^\lambda, 1^n, 1^D, 1^P)$

INITIALIZE$(1^\lambda, 1^n, 1^D, 1^P)$:

$\mathcal{T}, \mathcal{S}, \mathcal{S}_A \leftarrow \emptyset$, $\zeta, \zeta_A \leftarrow 0$

$(\mathsf{sk}_l, \mathsf{ek}_l)_{\forall l \in P} \leftarrow\$ \mathsf{KeyGen}(1^\lambda, 1^n, 1^D, 1^P)$

$(x_{1_0}, K_1, x_1) \leftarrow \mathsf{sk}_1$, $u_1 \leftarrow g_1^{x_1}$, $(u_1^{x_{1_0}^k})_{\forall k \in [0,D]} \leftarrow \mathsf{ek}_1$

Return ek_1, $(\mathsf{ek}_l, \mathsf{sk}_l)_{\forall l \in [2,P]}$

RO(y):

same as before

AUTHO(τ, m):

same as before

VERO$(m, \mathcal{P}, \sigma)$:

$w_1 \leftarrow H_2(m)$, $(f, \tau_i)_{\forall i \in n} \leftarrow \mathcal{P}$

if $\mathcal{P}$ is well defined on $\mathcal{T}$, do same as $\mathbf{G}^1_{\Theta,\mathcal{A}}$

if $\mathcal{P}$ not well defined on $\mathcal{T}$: $d \leftarrow 0$

  $\forall \tau_i \notin \mathcal{T}$: $\mathbf{r}^{**}_{\tau_i} \leftarrow \mathcal{R}(\tau_i)$; $\mathcal{S} = \mathcal{S} \cup \mathbf{r}^{**}_{\tau_i}$

  $\forall \tau_i \in \mathcal{T}$: $\mathbf{r}^{**}_{\tau_i} \leftarrow \mathcal{T}(\tau_i, \cdot, \cdot)$; $\mathcal{S} = \mathcal{S} \cup \mathbf{r}^{**}_{\tau_i}$

  $\rho^{**} \leftarrow f(\mathcal{S})$

  if $\sigma = (y_0, y_1)$

    $z \leftarrow \mathrm{e}(u_1, w_1^{\rho^{**}}) \cdot \mathrm{e}(u_1^{x_1 \cdot y_1}, v_2)^{-1} \cdot \mathrm{e}(u_1, w_1^{y_0})^{-1}$

    if $z = 1 \bmod p : \zeta \leftarrow 1$

  else if $\sigma = \Lambda$; $Z \leftarrow \mathrm{e}(u_1, w_1^{\rho^{**}}) \cdot \mathrm{e}(u_1, w_1^m)^{-1} \cdot \mathrm{e}(\Lambda, w_1)^{-1}$

    if $Z = 1 : \zeta \leftarrow 1$

Return d

AVERO$((m'_j, \mathcal{P}'_j)_{\forall j \in U}, (\mathsf{ek}'_j)_{\forall j \in U \setminus \{1\}}, \sigma')$:

$w_l \leftarrow H_2(m'_l)$, $(f"_l, \tau"_{l,i})_{\forall l \in U, \forall i \in n} \leftarrow \mathcal{P}'_l$

if $\mathcal{P}'_1$ is well defined on $\mathcal{T}$, do same as $\mathbf{G}^1_{\Theta,\mathcal{A}}$

if $\mathcal{P}'_1$ is not well defined on $\mathcal{T}$: $d \leftarrow 0$

  $\forall \tau"_{1,i} \notin \mathcal{T}$: $\mathbf{r}"_{\tau"_{1,i}} \leftarrow \mathcal{R}(\tau"_{1,i})$; $\mathcal{S}_A = \mathcal{S}_A \cup \mathbf{r}"_{\tau"_{1,i}}; i \in [n]$

  $\forall \tau"_{1,i} \in \mathcal{T}$: $\mathbf{r}"_{\tau"_{1,i}} \leftarrow \mathcal{T}(\tau"_{1,i})$; $\mathcal{S}_A = \mathcal{S}_A \cup \mathbf{r}"_{\tau"_{1,i}}; i \in [n]$

  $\rho"_1 \leftarrow f"_1(\mathcal{S}_A)$

  $\forall l \in U \setminus \{1\}$

    $\rho"_l \leftarrow f"_l((r"_{\tau_{l,i}})_{\forall i \in [n]})$ $,(r"_{\tau"_{l,i}})_{\forall i \in [n]} \leftarrow PRF_{K_l}(\tau"_{l,i})_{\forall i \in [n]}$

  $B \leftarrow \sigma' \cdot \left(\Pi_{\forall l \in U \setminus \{1\}} \mathrm{e}(u_l, w_l^{\rho"_l}) \cdot \mathrm{e}(u_l, w_l^{m'_l})^{-1} \right)^{-1}$

  $Z_A \leftarrow \mathrm{e}(u_1, w_1^{\rho"_1}) e(u_1, w_1^{m'_1})^{-1} \cdot B^{-1}$

  if $Z_A = 1 : \zeta_A \leftarrow 1$

Return d

FINALIZE(IN):

same as before

Fig. 5: Game 2 for security proof for Θ

Game $\mathbf{G}^3_{\Theta,\mathcal{A}}(1^\lambda, 1^n, 1^D, 1^P)$

INITIALIZE$(1^\lambda, 1^n, 1^D, 1^P)$:
1 $\mathcal{T}, \mathcal{S}, \mathcal{S}_A \leftarrow \emptyset$, $\zeta, \zeta_A \leftarrow 0$
2 $(\mathsf{sk}_l, \mathsf{ek}_l)_{\forall l \in P} \leftarrow\!\!\$\ \mathsf{KeyGen}(1^\lambda, 1^n, 1^D, 1^P)$
3 $(x_{1_0}, K_1, x_1) \leftarrow \mathsf{sk}_1$, $u_1 \leftarrow g_1^{x_1}$, $(u_1^{x_{1_0}^k})_{\forall k \in [0,D]} \leftarrow \mathsf{ek}_1$
4 Return ek_1, $(\mathsf{ek}_l, \mathsf{sk}_l)_{\forall l \in [2,P]}$

RO(y):
5 same as before

AUTHO(τ, m):
6 if $(\tau, m, \cdot) \in \mathcal{T}$, $\sigma \leftarrow \mathcal{T}(\tau, m, \cdot)$
7 if $(\tau, m, \cdot) \notin \mathcal{T}$,$\sigma \leftarrow \mathsf{Auth}_{\mathsf{sk}_1}(\tau, m)$ using $\mathcal{R}$;
8 $(y_0, y_1) \leftarrow \sigma$: if $y_1 \in \mathcal{S}_A \cup \mathcal{S}$
9 $y_1^{**} \leftarrow\!\!\$\ Z_p$; $\sigma \leftarrow y_0, y_1^{**}$
10 $\mathcal{T} = \mathcal{T} \cup (\tau, m, \sigma)$
11 if $(\tau, \cdot, \cdot) \notin \mathcal{T}$, ignore
12 Return σ

VERO$(m, \mathcal{P}, \sigma)$:
13 same as before

AVERO$((m'_j, \mathcal{P}'_j)_{\forall j \in U}, (\mathsf{ek}'_j)_{\forall j \in U \setminus \{1\}}, \sigma')$:
14 same as before

FINALIZE(IN):
15 same as before

Fig. 6: Game 3 for security proof for Θ

Game $\mathbf{G}^4_{\Theta,\mathcal{A}}(1^\lambda, 1^n, 1^D, 1^P)$

INITIALIZE$(1^\lambda, 1^n, 1^D, 1^P)$:

$\mathcal{T}, \mathcal{S} \leftarrow \emptyset$, $\zeta \leftarrow 0$, $\eta \leftarrow 0, \mathsf{DM} \leftarrow \emptyset$

$(\mathsf{sk}_l, \mathsf{ek}_l)_{\forall l \in P} \leftarrow\!\!\$\ \mathsf{KeyGen}(1^\lambda, 1^n, 1^D, 1^P)$

$(x_{1_0}, K_1, x_1) \leftarrow \mathsf{sk}_1$, $u_1 \leftarrow g_1^{x_1}$, $(u_1^{x_{1_0}^k})_{\forall k \in [0,D]} \leftarrow \mathsf{ek}_1$

Return ek_1, $(\mathsf{ek}_l, \mathsf{sk}_l)_{\forall l \in [2,P]}$

RO(y):

if $y \in \mathcal{H}$, $(y, w, b) \leftarrow \mathcal{H}[y]$ else:

  $b \leftarrow\!\!\$\ \mathbb{Z}_p$, $w \leftarrow g_2^b$, $\mathcal{H} = \mathcal{H} \cup (y, w, b)$

Return w

AUTHO(τ, m):

same as before

VERO$(m, \mathcal{P}, \sigma)$:

if $\mathcal{P}$ not well defined in $\mathcal{T}$, do same as before.

if $\mathcal{P}$ is well defined on $\mathcal{T}$:

  $\forall (\tau_i, \cdot, \cdot) \notin \mathcal{T} : \hat{\sigma}_i \leftarrow\!\!\$\ \mathbb{Z}_p^2$; $\mathsf{DM} = \mathsf{DM} \cup \hat{\sigma}_i$

  $\forall (\tau_i, \cdot, \cdot) \in \mathcal{T} : \hat{\sigma}_i \leftarrow\!\!\$\ \mathcal{T}(\tau_i, \cdot, \cdot); \mathsf{DM} = \mathsf{DM} \cup \hat{\sigma}_i$

  $\sigma^{***} \leftarrow \mathsf{Eval}(\mathsf{ek}_1, f, \mathsf{DM})$, $m^{***} \leftarrow f\left((\hat{\sigma}[0]_i)_{\forall i \in n}\right))$

  if $\sigma^{***} = \sigma : d \leftarrow 1$ else $d \leftarrow 0$

  if $((y_0^{***}, y_1^{***}) \leftarrow \sigma^{***}) \neq ((y_0, y_1) \leftarrow \sigma)$

    $z \leftarrow \frac{\mathrm{e}(u_1^{y_1 x_1}, g_2^b)}{\mathrm{e}(u_1^{y_1^{***} x_1}, g_2^b)} \mathrm{e}(g_1, g_2^b)^{(y_0 - y_0^{***})}$

    if $z = 1$, $\eta \leftarrow 1$

  if $(\Lambda^{***} \leftarrow \sigma^{***}) \neq (\Lambda \leftarrow \sigma)$

    $Z \leftarrow \frac{\mathrm{e}(\Lambda, g_2^b)}{\mathrm{e}(\Lambda^{***}, g_2^b)} \cdot \mathrm{e}(g_1, g_2^b)^{(m - m^{***})}$

    if $Z = 1$, $\eta \leftarrow 1$

Return d

AVERO$((m'_j, \mathcal{P}'_j)_{\forall j \in U}, (\mathsf{ek}'_j)_{\forall j \in U \setminus \{1\}}, \sigma')$:

same as before

FINALIZE(IN):

same as before

Fig. 7: Game 4 for security proof for Θ

Game $\mathbf{G}^5_{\Theta,\mathcal{A}}(1^\lambda, 1^n, 1^D, 1^P)$

INITIALIZE$(1^\lambda, 1^n, 1^D, 1^P)$:

$\zeta \leftarrow 0$ $\eta \leftarrow 0$, $\beta_A \leftarrow 0, \mathcal{C}_1 \leftarrow \emptyset$

$(\mathsf{sk}_l, \mathsf{ek}_l)_{\forall l \in P} \leftarrow\!\!\$\ \mathsf{KeyGen}(1^\lambda, 1^n, 1^D, 1^P)$

$(x_{1_0}, K_1, x_1) \leftarrow \mathsf{sk}_1$, $u_1 \leftarrow g_1^{x_1}$, $(u_1^{x_{1_0}^k})_{\forall k \in [0,D]} \leftarrow \mathsf{ek}_1$

Return ek_1, $(\mathsf{ek}_l, \mathsf{sk}_l)_{\forall l \in [2,P]}$

RO(y):

same as before

AUTHO(τ, m):

same as before

VERO$(m, \mathcal{P}, \sigma)$:

same as before

AVERO$((m'_j, \mathcal{P}'_j)_{\forall j \in U}, (\mathsf{ek}'_j)_{\forall j \in U \setminus \{1\}}, \sigma')$:

if $\mathcal{P}'_1$ not well defined in $\mathcal{T}$, do same as before.

if $\mathcal{P}'_1$ is well defined on $\mathcal{T}$:

  $\forall \tau'_i \in \mathcal{P}'_1$: if $\tau'_{1,i} \notin \mathcal{T} : \bar{\sigma} \leftarrow\!\!\$\ \mathbb{Z}_p^2; \mathcal{C}_1 = \mathcal{C}_1 \cup \bar{\sigma}$

    if $\tau'_{1,i} \in \mathcal{T} : \bar{\sigma} \leftarrow \mathcal{T}(\tau'_{1,i}, \cdot, \cdot); \mathcal{C}_1 = \mathcal{C}_1 \cup \bar{\sigma}$

  $\bar{\Lambda}_1 \leftarrow \mathsf{Eval}(\mathsf{ek}_1, f'_1, \mathcal{C}_1)$, $\bar{m}_1 \leftarrow f'_1(\mathcal{C}_1[0])$

  $B \leftarrow \sigma' \cdot \left(\Pi_{\forall l \in U \setminus \{1\}} \left(\mathrm{e}(u_l, w_l^{\rho_l}) \cdot \mathrm{e}(u_l, w_l^{m'_l})^{-1} \right) \right)^{-1}$

  if $\mathrm{e}(\bar{\Lambda}_1, g_2^{b_1}) = B$, $d \leftarrow 1$

  if $\mathrm{e}(\bar{\Lambda}_1, g_2^{b_1}) \neq B$, $d \leftarrow 0$

    $Z_A \leftarrow \frac{B}{\mathrm{e}(\bar{\Lambda}_1, g_2^{b_1})} \mathrm{e}(g_1, g_2^{b_1})^{m'_1 - \bar{m}_1}$

    if $Z_A = 1$, $\beta_A \leftarrow 1$

Return d

FINALIZE(IN):

same as before

Fig. 8: Game 5 for security proof for Θ

Adv $\mathcal{B}_{\Theta,\mathcal{A}}(1^\lambda, 1^n, 1^D, 1^P)$

INITIALIZE$(1^\lambda, 1^n, 1^D, 1^P, (g_1, h) \in \mathcal{G}_1, (g_2, g_2^a) \in \mathcal{G}_2)$:

$K_1 \leftarrow\!\!\$\ \mathcal{K}\ x_{1_0} \leftarrow\!\!\$\ \mathbb{Z}_p.$, $\mathsf{ek}_1^* \leftarrow (h^{x_{1_0}^k})_{\forall k\in[0,D]}$, $\mathsf{sk}_1^* \leftarrow (x_{1_0}, K_1)$

$\forall l \in [2,P] : K_l \leftarrow\!\!\$\ \mathcal{K}, (x_{l_0}, x_l) \leftarrow\!\!\$\ \mathbb{Z}_p^2, u_l \leftarrow g_1^{x_l}$

$\mathsf{ek}_l \leftarrow (u_l^{x_{l_0}^k})_{\forall k\in[0,D]}, \mathsf{sk}_l \leftarrow (x_{l_0}, K_l, x_l)$

Return ek_1^*, $(\mathsf{sk}_l, \mathsf{ek}_l)_{\forall l\in[2,P]}$

RO(y):

if $y \in \mathcal{H}$, $(y, w, b, c) \leftarrow \mathcal{H}[y]$ else:

  $c \leftarrow \{0,1\}$, $b \leftarrow\!\!\$\ \mathbb{Z}_p$, if $c = 0$: $w \leftarrow g_2^{a+b}$

  else if $c = 1$: $w \leftarrow g_2^b$

$\mathcal{H} = \mathcal{H} \cup (y, w, b, c)$

Return w

AUTHO(τ, m):

if $c = 0$: Return $\perp$

else same as $\mathbf{G}^4_{\Theta,\mathcal{A}}$

VERO$(m, \mathcal{P}, \sigma)$:

if $c = 0$: Return $\perp$

  Return OUT$(m, \mathcal{P}, \sigma)$

else same as $\mathbf{G}^4_{\Theta,\mathcal{A}}$

AVERO$((m'_1, \ldots, m'_P), (\mathcal{P}'_1, \ldots, \mathcal{P}'_P), (\mathsf{ek}'_2, \ldots, \mathsf{ek}'_P), \sigma')$:

if $c = 0$: Return $\perp$

else same as $\mathbf{G}^4_{\Theta,\mathcal{A}}$

OUT$(m^*, \mathcal{P}^*, \sigma^*)$: // $m^* \notin \mathcal{T}$

  $\rho^* \leftarrow f^*\left((r_{\tau^*_{1,i}})_{\forall i\in n}\right)$, $(r_{\tau^*_{1,i}})_{\forall i\in[n]} \leftarrow PRF_{K_1}(\tau^*_{1,i})_{\forall i\in[n]}$

  $(y_{1,k})_{\forall k\in[0,D]} \leftarrow f^*\left((\sigma_{1,i})_{\forall i\in[n]}\right)$, $(\sigma_{1,i})_{\forall i\in[n]} \leftarrow \mathcal{T}((\tau^*_{1,i})_{\forall i\in[n]}, \cdot, \cdot)$

  $\bar{m} \leftarrow f_1^*\left((m_{1,i})_{\forall i\in[n]}\right)$, $(m_{1,i})_{\forall i\in[n]} \leftarrow \mathcal{T}((\tau^*_{1,i})_{\forall i\in[n]}, \cdot)$

  if $\Lambda^* \leftarrow \sigma^*$:

    Return $\left(\psi(g^{(a+b)})^{(m^*-\bar{m})}\right)^{(m^*-\rho^*+\Sigma_{k=1}^D y_{1,k}x_{1_0}^k)^{-1}} \cdot \left(h^b\right)^{-1}$

  if $(y_0^*, y_1^*) \leftarrow \sigma^*$:

    Return $\left(\psi(g_2^{(a+b)})^{(y_0^*-y_{1,0})}\right)^{\left(x_{1_0}(y_{1,1}-y_1^*)\right)^{-1}} \cdot \left(h^b\right)^{-1}$

FINALIZE(IN): // $(\cdot, m^*, \cdot) \notin \mathcal{T}$

if $c = 0$, Return $\perp$

else same as $\mathbf{G}^4_{\Theta,\mathcal{A}}$

Fig. 9: Adversary $\mathcal{B}$ against co-CDH assumption

Adv $\mathrm{C}_{\Theta,\mathcal{A}}(1^\lambda, 1^n, 1^D, 1^P)$

$\textsc{Initialize}(1^\lambda, 1^n, 1^D, 1^P, (g_1, h) \in \mathcal{G}_1, (g_2, g_2^a) \in \mathcal{G}_2)$:

$K_1 \leftarrow\$ \mathcal{K}$, $x_{1_0} \leftarrow\$ \mathbb{Z}_p$, $\mathsf{ek}_1^* \leftarrow (h^{x_{1_0}^k})_{\forall k \in [0,D]}$, $\mathsf{sk}_1^* \leftarrow x_{1_0}, K_1$

$\forall l \in [2, P] : K_l \leftarrow\$ \mathcal{K}, (x_{l_0}, x_l) \leftarrow\$ \mathbb{Z}_p^2, u_l \leftarrow g_1^{x_l}$

$\mathsf{ek}_l \leftarrow (u_l^{x_{l_0}^k})_{\forall k \in [0,D]}, \mathsf{sk}_l \leftarrow (x_{l_0}, K_l, x_l)$

Return ek_1^*, $(\mathsf{sk}_l, \mathsf{ek}_l)_{\forall l \in [2,P]}$

$\mathrm{RO}(y)$:

if $y \in \mathcal{H}$, $(y, w, b, c) \leftarrow \mathcal{H}[y]$ else:

  $c \leftarrow \{0, 1\}$, $b \leftarrow\$ \mathbb{Z}_p$, if $c = 0$: $w \leftarrow g_2^{a+b}$

  else if $c = 1$: $w \leftarrow g_2^b$

$\mathcal{H} = \mathcal{H} \cup (y, w, b, c)$

Return w

$\textsc{AuthO}(\tau, m)$:

if $c_1 = 0$: Return $\perp$

else same as $\mathbf{G}^5_{\Theta,\mathcal{A}}$

$\textsc{VerO}(m, \mathcal{P}, \sigma)$:

if $c_1 = 0$: Return $\perp$

else same as $\mathbf{G}^5_{\Theta,\mathcal{A}}$

$\textsc{AVerO}((m'_j, \mathcal{P}'_j)_{\forall j \in U}, (\mathsf{ek}'_j)_{\forall j \in U \setminus \{1\}}, \sigma')$:

if $c_1 = 0$: Return $\perp$

  Return $\textsc{Out}((m'_j, \mathcal{P}'_j)_{\forall j \in U}, (\mathsf{ek}'_j)_{\forall j \in U \setminus \{1\}}, \sigma')$

else same as $\mathbf{G}^5_{\Theta,\mathcal{A}}$

$\textsc{Out}((m^*_j, \mathcal{P}^*_j)_{\forall j \in U}, (\mathsf{ek}^*_j)_{\forall j \in U \setminus \{1\}}, \sigma^*)$: // $m_1^* \notin \mathcal{T}$

if $c_1 = 0$ and $\forall l \in U \setminus \{1\}$ st $c_l = 1$:

  $\forall l \in U, \rho_l \leftarrow f_l^*\left((r_{\tau^*_{l,i}})_{\forall i \in [n]}\right)$, $(r_{\tau^*_{l,i}})_{\forall i \in [n]} \leftarrow PRF_{K_l}(\tau^*_{l,i})_{\forall i \in [n]}$

  compute $B \leftarrow \left(\sigma^* \cdot \left(\Pi_{\forall l \in U \setminus \{1\}} e(u_l, w_l^{\rho_l}) \cdot e(u_l, w_l^{m_l^*})^{-1}\right)^{-1}\right)$

  $(y_{1,k})_{\forall k \in [0,D]} \leftarrow f_1^*\left((\sigma_{1,i})_{\forall i \in [n]}\right)$, $(\sigma_{1,i})_{\forall i \in [n]} \leftarrow \mathcal{T}((\tau^*_{1,i})_{\forall i \in [n]}, \cdot)$

  $\bar{m}_1 \leftarrow f_1^*\left((m_{1,i})_{\forall i \in [n]}\right)$, $(m_{1,i})_{\forall i \in [n]} \leftarrow \mathcal{T}((\tau^*_{1,i})_{\forall i \in [n]}, \cdot)$

  Return $(\psi(g_2^{(a+b_1)})^{(m_1^* - \bar{m}_1)})^{\left(m_1^* - \rho_1 + (\Sigma_{k=1}^D y_{1,k} x_{1_0}^k)\right)^{-1}} \cdot (h^{b_1})^{-1}$

$\textsc{Finalize}(IN)$: // $(\cdot, m_1^*, \cdot) \notin \mathcal{T}$

if $c_1 = 0$ Return $\perp$

else same as $\mathbf{G}^5_{\Theta,\mathcal{A}}$

Fig. 10: Adversary C against co-CDH assumption