Online-Offline Homomorphic Signatures for Polynomial Functions

Kaoutar Elkhiyaoui, Melek Önen, and Refik Molva

EURECOM, Sophia Antipolis, France
{elkhiyao, onen, molva}@eurecom.fr

Abstract. The advent of cloud computing has given rise to a plethora of work on verifiable delegation of computation. Homomorphic signatures are a powerful tool that can be tailored for verifiable computation, as long as they are efficiently verifiable. The main advantages of homomorphic signatures are twofold: (i) *public verifiability:* Any third party can verify the correctness of the delegated computation; (ii) *statelessness:* The verifier is not required to have access to the dataset on which the computation was performed. Thus in this paper, we design a homomorphic signature suitable for multivariate polynomials of bounded degree, and which draws upon the algebraic properties of *eigenvectors* and *leveled multilinear maps*. The proposed signature yields an efficient verification process (in an amortized sense) and supports offline-online signing. Furthermore, our signature is provably secure and its size grows only linearly with the degree of the evaluated polynomial.

1 Introduction

The problem of verifiable computation has attracted increasing interest with the rise of cloud computing. Thanks to the various computational and financial advantages of cloud technology, companies are keen to delegate their computation tasks to powerful servers. Yet, since such servers are considered to be potentially malicious, one major challenge is to empower cloud customers to efficiently verify the correctness of the requested computations. Homomorphic signatures are one of the cryptographic tools that perfectly address this challenge, so long as their underlying verification algorithm is efficient (in the amortized sense) and can be used by lightweight clients.

While all existing homomorphic signature designs rely on the use of either polynomials [11] or lattices [5, 18], in this paper, we propose a new approach that leverages the algebraic properties of *eigenvectors* to achieve homomorphism in signatures. Within this new solution, a homomorphic signature is mapped to a $(2, 2)$-matrix admitting a "predefined" eigenvalue with respect to some secret vector $\boldsymbol{u}$ ($\boldsymbol{u}$ defines the signing key). This mapping can be easily shown to be homomorphic: The eigenvalue of the sum (product resp.) of two matrices is equal to the sum (product resp.) of the eigenvalues of each matrix. More specifically, our homomorphic signature encodes each component of the $(2, 2)$-matrix in the exponent. Since such an encoding preserves the addition operation, this signature is additively homomorphic. On the other hand, in order to support multiplication, the proposed solution uses a *leveled multilinear map* [13]. Thanks to this

new design, the signature verification is more efficient and the size of the signature is reduced compared to [11].

Additionally, to the best of our knowledge, our solution is the first homomorphic signature that features an online/offline signing procedure. By pre-computing "offline" signatures over a pre-defined random dataset, the actual "online" signature operation becomes much cheaper.

The major contributions of this paper are:

- A new original primitive for homomorphic signatures which combines the use of eigenvectors with leveled multilinear maps to support multivariate polynomials.
- The solution is provably secure under the multilinear Diffie-Hellman inversion (MDHI) assumption [6]: Similarly to existing work, we first propose a weakly sound solution that we later transform into an adaptively sound one. Our transformation consists of replacing the message to be-signed with the evaluation of a one-degree polynomial at a secret point.
- Because the signature is mapped to a matrix, the size of the weakly sound signature is constant. Moreover, our adaptively sound solution results in signatures whose size grows only linearly in the degree of the evaluated polynomial.
- Similarly to previous work, our signature enables online/offline verification. Furthermore, it actually supports *online/offline signatures* as well: Namely, our solution allows the signer to pre-compute signatures of a random dataset (offline), and whenever the "to-be-signed" dataset is generated, the signer applies cheap transformations over the pre-computed signatures.

The rest of the paper is organized as follows: Section 2 formalizes the definition and the security properties of a homomorphic signature. The main building blocks of the proposed solution are presented in Section 3. Section 4 introduces a first version of our solution and proves its security using a weak unforgeability experiment. This solution is further transformed in Section 5 to provide adaptive security. Finally, section 6 reviews related work.

2 Background

2.1 Multi-labeled Programs

We first recall the definition of labeled programs, which by tying program inputs to predefined labels enable the construction of homomorphic signatures (cf. [5, 14]).

Definition 1. *A labeled program $\mathcal{P}$ evaluating an n-variate function $f : \mathbb{I}^n \rightarrow \mathbb{I}$, is defined by a tuple $(f, \tau_1, \tau_2, ..., \tau_n)$, where $\tau_i \in \{0,1\}^*$ is the label associated with the i^{th} variable of function f (i.e. the i^{th} input of program $\mathcal{P}$).*

Given labeled programs $\mathcal{P}_1, ..., \mathcal{P}_t$ and a function $g : \mathbb{I}^t \rightarrow \mathbb{I}$, we define the composed (labeled) program $\mathcal{P}^C = g(\mathcal{P}_1, ..., \mathcal{P}_t)$ as the evaluation of function g on the outputs of programs $\mathcal{P}_1, ..., \mathcal{P}_t$. In this case, the labeled inputs of program $\mathcal{P}^C$ correspond to the distinctly *labeled inputs of programs $\mathcal{P}_1, ..., \mathcal{P}_t$. Namely, the inputs of $\mathcal{P}_1, ..., \mathcal{P}_t$ associated with the same label will form one single input for the composed program $\mathcal{P}^C$.*

If we denote $\mathcal{I}_\tau$ the identity program associated with the canonical identity function $\mathcal{I}$ and label $\tau \in \{0,1\}^*$, then any labeled program $\mathcal{P} = (f, \tau_1, \tau_2, ..., \tau_n)$ can be expressed as the composition of identity programs $\mathcal{I}_{\tau_1}, ..., \mathcal{I}_{\tau_n}$ and function f, i.e. $\mathcal{P} = f(\mathcal{I}_{\tau_1}, ..., \mathcal{I}_{\tau_n})$.

Similarly to the work of [11], in this paper, we focus on *multi-labeled programs* which give way to the construction of efficiently verifiable homomorphic signatures. In a nutshell, a multi-labeled program assigns label not only to program inputs, but also to the dataset to which these inputs belong (cf. Definition 2).

Definition 2. *A multi-labeled program $\mathcal{P}_\Delta$ is defined by a pair of dataset identifier $\Delta \in \{0,1\}^*$ and a labeled program $\mathcal{P}$. As such, $\mathcal{P}_\Delta$ indicates that program $\mathcal{P}$ operates on inputs from dataset Δ.*

As labeled programs, multi-labeled programs associated with the same dataset identifier support composition. Notably, given multi-labeled programs $(\mathcal{P}_1, \Delta), ..., (\mathcal{P}_t, \Delta)$ sharing the same dataset identifier Δ, and a function $g : \mathbb{I}^t \to \mathbb{I}$, we define the composed multi-labeled program $\mathcal{P}^C_\Delta = g\left((\mathcal{P}_1, \Delta), ..., (\mathcal{P}_t, \Delta)\right)$ by the pair $(\mathcal{P}^C, \Delta)$, where $\mathcal{P}^C$ is the composed program $g(\mathcal{P}_1, ..., \mathcal{P}_t)$.

Moreover, we define the multi-labeled identity program $\mathcal{I}_{(\Delta,\tau)}$ by the pair $(\mathcal{I}_\tau, \Delta)$. Consequently, if $\mathcal{P}_\Delta = (\mathcal{P}, \Delta)$, where $\mathcal{P}$ is the labeled program defined as $(f, \tau_1, ..., \tau_n)$, then $\mathcal{P}_\Delta$ can be expressed as the composition of function f and the multi-labeled identities $\mathcal{I}_{(\Delta,\tau_1)}, ..., \mathcal{I}_{(\Delta,\tau_n)}$, i.e. $\mathcal{P}_\Delta = (f(\mathcal{I}_{\tau_1}, ..., \mathcal{I}_{\tau_n}), \Delta)$.

2.2 Efficient Homomorphic Signatures for Multi-labeled Programs

According to the work of [11], a homomorphic signature for multi-labeled programs consists of the following algorithms:

- $\mathsf{KeyGen}(1^\kappa, \mathcal{L}) \to (\mathsf{SK}, \mathsf{VK}, \mathsf{param})$: It is a randomized algorithm that takes as input a security parameter κ and the label space $\mathcal{L}$, and outputs a secret key SK, the matching public verification key VK, and a set of public parameters param describing the set of admissible inputs $\mathbb{I}$ and the set of admissible functions $\mathcal{F}$.
- $\mathsf{Sign}(\mathsf{SK}, \Delta, \tau, m) \to \sigma$: On inputs of secret key SK, a dataset identifier Δ, a label $\tau \in \mathcal{L}$, and an input message $m \in \mathbb{I}$, algorithm Sign outputs a signature σ. By definition, signature σ authenticates m as the output of the identity program $(\mathcal{I}_\tau, \Delta)$.
- $\mathsf{Eval}(\mathsf{VK}, f, \boldsymbol{\sigma}) \to \sigma$: On input of verification key VK, an n-variate function $f : \mathbb{I}^n \to \mathbb{I}$, and a vector $\boldsymbol{\sigma} = (\sigma^{(1)}, ..., \sigma^{(n)})$ of n signatures, algorithm Eval outputs a new signature σ. If each signature $\sigma^{(i)}$ authenticates a message $m^{(i)}$ as the output of a multi-labeled program $(\mathcal{P}_i, \Delta)$, then by definition, signature σ authenticates the output of the composed program $(f(\mathcal{P}_1, ..., \mathcal{P}_n), \Delta)$.
- $\mathsf{Verify}(\mathsf{VK}, \mathcal{P}_\Delta, m, \sigma) \to b$: It is a deterministic algorithm that takes as inputs public verification key VK, a multi-labeled program $\mathcal{P}_\Delta = (\mathcal{P}, \Delta)$, a message $m \in \mathbb{I}$, and a signature σ. It accordingly verifies using signature σ, whether m is the output of program $\mathcal{P}$ when executed on previously authenticated labeled messages belonging to dataset Δ; and it outputs $b = 1$, if it decides that m was computed as the output of program $\mathcal{P}$, and $b = 0$ otherwise.

For a homomorphic signature to be secure, it should ensure the properties of **correctness** and **soundness**.

Correctness Correctness in homomorphic signatures is captured through two requirements. The first one is *authentication correctness* which ensures that the output of a correct execution of algorithm Sign is always accepted by algorithm Verify. The second requirement is *evaluation correctness* which assures that algorithm Eval always yields outputs that are accepted by algorithm Verify.

Authentication Correctness We say that a homomorphic signature satisfies authentication correctness, if for any input $m \in \mathbb{I}$, any label $\tau \in \mathcal{L}$ and any dataset Δ, the signature σ produced by algorithm $\mathsf{Sign}(\mathsf{SK}, \Delta, \tau, m)$ will correctly authenticate m as the output of identity program $\mathcal{I}_{(\Delta,\tau)} = (\mathcal{I}_\tau, \Delta)$.

Definition 3. *A homomorphic signature provides authentication correctness* **iff***:*

For any tuple of keys and public parameters $(\mathsf{SK}, \mathsf{VK}, \mathsf{param}) \leftarrow \mathsf{KeyGen}(1^\kappa, \mathcal{L})$ *and any signature* $\sigma \leftarrow \mathsf{Sign}(\mathsf{SK}, \Delta, \tau, m)$*:*

$$\Pr[\mathsf{Verify}(\mathsf{VK}, \mathcal{I}_{(\Delta,\tau)}, m, \sigma) \rightarrow 1] = 1$$

Evaluation Correctness Intuitively, evaluation correctness ensures that given an n-variate function f and a vector of n signatures $\boldsymbol{\sigma} = (\sigma^{(1)}, ..., \sigma^{(n)})$, such that each signature $\sigma^{(i)}$ authenticates a message $m^{(i)}$ as the output of a multi-labeled program $(\mathcal{P}_i, \Delta)$, algorithm Eval generates a signature σ that correctly authenticates the output m of the composed program $(f(\mathcal{P}_1, ..., \mathcal{P}_n), \Delta)$ executed on inputs $m^{(1)}, ..., m^{(n)}$. Notably, algorithm Verify will accept σ as a valid signature of message m, with respect to multi-labeled program $(f(\mathcal{P}_1, ..., \mathcal{P}_n), \Delta)$.

Definition 4. *A homomorphic signature ensures evaluation correctness* **iff***:*

For any tuple of keys and public parameters $(\mathsf{SK}, \mathsf{VK}, \mathsf{param}) \leftarrow \mathsf{KeyGen}(1^\kappa, \mathcal{L})$*, and any set of tuples* $\{(\mathcal{P}_i, \Delta), m^{(i)}, \sigma^{(i)}\}_{i=1}^n$ *such that* $\mathsf{Verify}(\mathsf{VK}, (\mathcal{P}_i, \Delta), m^{(i)}, \sigma^{(i)}) \rightarrow 1$*, when we denote* m *the output of the composed program* $\mathcal{P}_\Delta^C = (f(\mathcal{P}_1, ..., \mathcal{P}_n), \Delta)$ *executed on inputs* $(m^{(1)}, ..., m^{(n)})$ *(i.e.* $m = f(m^{(1)}, ..., m^{(n)})$*) and* $\boldsymbol{\sigma}$ *the vector* $(\sigma^{(1)}, ..., \sigma^{(n)})$*, we get the following equality:*

$$\Pr[\mathsf{Verify}(\mathsf{VK}, \mathcal{P}_\Delta^C, m, \sigma) \rightarrow 1 \mid \mathsf{Eval}(\mathsf{VK}, f, \boldsymbol{\sigma}) \rightarrow \sigma] = 1$$

Soundness We say that a homomorphic scheme is sound, if the only way to make algorithm Verify accept a tuple $(\mathcal{P}_\Delta, m, \sigma)$ comprising a mutli-labeled program $\mathcal{P}_\Delta$ evaluating an n-variate function f, a message m and a signature σ (i.e. $\mathsf{Verify}(\mathsf{VK}, \mathcal{P}_\Delta, m, \sigma) \rightarrow 1$), is by computing m as the outcome of an execution of program $\mathcal{P}_\Delta$ on some inputs $m^{(1)}, ..., m^{(n)}$ belonging to dataset Δ, and having σ equal the output of Eval when called with function f and vector $\boldsymbol{\sigma} = (\sigma^{(1)}, ..., \sigma^{(n)})$ composed of the signatures authenticating messages $m^{(i)}$.

Algorithm 1: The unforgeability experiment of homomorphic signatures

$(\mathsf{VK}, \mathsf{param}) \leftarrow \mathcal{O}_{\mathsf{KeyGen}}(1^{\kappa}, \mathcal{L})$;
$\mathcal{A}$ can do the following in any interleaved order
$\mathcal{A}$ picks up to s dataset identifiers
$\mathcal{A} \rightarrow \Delta_i$;
For each dataset Δ_i, $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Sign}}$ up to t times
$\mathcal{A} \rightarrow (\tau_{(i,j)}, m^{(i,j)})$; # if $l \neq j$, then $\tau_{(i,j)} \neq \tau_{(i,l)}$
$\sigma^{(i,j)} \leftarrow \mathcal{O}_{\mathsf{Sign}}(\mathsf{VK}, \Delta_i, \tau_{(i,j)}, m^{(i,j)})$;
$\mathcal{A}$ outputs the tuple on which is going to be challenged
$\mathcal{A} \rightarrow (\mathcal{P}_{\Delta}, m, \sigma)$; # $\mathcal{P}_{\Delta} = (\mathcal{P}, \Delta)$
$b \leftarrow \mathsf{Verify}(\mathsf{VK}, \mathcal{P}_{\Delta}, m, \sigma)$;

In accordance with previous work on homomorphic signatures [11], we formalize soundness by way of an *unforgeability experiment*. During this experiment, an adversary $\mathcal{A}$ is allowed not only to run algorithms Verify and Eval, but also to access the output of algorithms KeyGen and Sign through the following oracles:

- $\mathcal{O}_{\mathsf{KeyGen}}$: When queried with a security parameter 1^{κ} and a label space $\mathcal{L}$, this oracle generates a set of public parameters param, a secret key SK and the corresponding verification key VK; and returns the pair $(\mathsf{VK}, \mathsf{param})$.
- $\mathcal{O}_{\mathsf{Sign}}$: When called with a verification key VK, a dataset identifier $\Delta \in \{0,1\}^*$, a label $\tau \in \mathcal{L}$ and a message m, oracle $\mathcal{O}_{\mathsf{Sign}}$ retrieves the secret key SK matching verification key VK, executes algorithm Sign on input $(\mathsf{SK}, \Delta, \tau, m)$, and finally outputs the resulting signature σ.

As depicted in Algorithm 1, adversary $\mathcal{A}$ enters the unforgeability experiment by querying the oracle $\mathcal{O}_{\mathsf{KeyGen}}$ with security parameter κ and label space $\mathcal{L}$. In turn, $\mathcal{O}_{\mathsf{KeyGen}}$ outputs a verification key VK and a set of public parameters param that will be used throughout the experiment. Later, adversary $\mathcal{A}$ *adaptively* picks s dataset identifiers Δ_i. For each dataset identifier Δ_i, adversary $\mathcal{A}$ submits t adaptive queries to oracle $\mathcal{O}_{\mathsf{Sign}}$: Namely, adversary $\mathcal{A}$, selects t pairs of label and message $(\tau_{(i,j)}, m^{(i,j)})$ such that for all $l \neq j$, $\tau_{(i,j)} \neq \tau_{(i,l)}$. Notice that in this manner, we take into account the fact that adversary $\mathcal{A}$ can only submit one signature query per pair of dataset identifier Δ and label τ.

Eventually, adversary $\mathcal{A}$ produces a tuple of multi-labeled program $\mathcal{P}_{\Delta} = (\mathcal{P}, \Delta)$, message m and signature σ. To conclude the experiment, adversary $\mathcal{A}$ executes algorithm Verify on the challenge tuple $(\mathcal{P}_{\Delta}, m, \sigma)$.

Without loss of generality, we denote b the output of this execution of algorithm Verify, and we assume that program $\mathcal{P}$ evaluates an n-variate function f and is associated with labels $(\tau_1, ..., \tau_n)$.

Consequently, we say that adversary $\mathcal{A}$ succeeds in breaking the unforgeability experiment, if $b = 1$ and one of the following conditions holds:

- Adversary $\mathcal{A}$ never submitted a query involving dataset identifier Δ to oracle $\mathcal{O}_{\mathsf{Sign}}$. In this case, we say that the tuple $(\mathcal{P}_{\Delta}, m, \sigma)$ returned by adversary $\mathcal{A}$ is a forgery of **Type I**.
- Adversary $\mathcal{A}$ submitted signature queries to oracle $\mathcal{O}_{\mathsf{Sign}}$ for dataset identifier Δ and pairs $(\tau_j, m^{(j)})$, $1 \leq j \leq n$, that is: $\Delta = \Delta_i$ for some $i \in \{1, ..., s\}$, and $\{(\tau_j, m^{(j)})\}_{j=1}^{n} \subset \{(\tau_{(i,j)}, m^{(i,j)}\}_{j=1}^{t}$. Yet, m is not the correct output of the labeled program $\mathcal{P}$ when executed on messages $\{m^{(j)}\}_{j=1}^{n}$ (i.e. $m \neq f(m^{(1)}, ..., m^{(n)})$). In such a case, we say that adversary $\mathcal{A}$ provides a forgery of **Type II**.
- Adversary $\mathcal{A}$ submitted signature queries to oracle $\mathcal{O}_{\mathsf{Sign}}$ for dataset identifier Δ, namely: $\Delta = \Delta_i$ for some $i \in \{1, ..., s\}$, however, $\{\tau_1, ..., \tau_n\} \not\subset \{\tau_{(i,1)}, ..., \tau_{(i,t)}\}$. This case corresponds to a forgery of **Type III**.

Definition 5. *Let $\Pi^{\mathcal{A}}_{(s,t)}$ denote the probability that adversary $\mathcal{A}$ succeeds in the unforgeability experiment.*

A homomorphic signature scheme is (s,t)-sound **iff***,*

$$\Pi^{\mathcal{A}}_{(s,t)} \leq \epsilon(\kappa)$$

where κ is the security parameter and ϵ is a negligible function.

Efficiency In addition to the classical security properties of correctness and soundness, a homomorphic signature should be efficient as well. Namely, in accordance with previous work [11, 18], a homomorphic signature should be *succinct* and support *efficient verification*.

Succinctness Succinctness entails that the size of the signature of a program should not depend on the number of inputs to that program. More precisely, we consider a homomorphic signature succinct **if**, for a fixed security parameter κ, the output size of algorithm Eval for any function f does not depend on the size of the inputs of f.

Efficient Verification This property can be implemented by dividing the verification algorithm into two phases. An *offline* phase, during which the verifier is provided with verification key VK and a labeled program $\mathcal{P}$ so as to compute a *concise* verification key $\mathsf{VK}_{\mathcal{P}}$. The computed key is then used in an *online* phase, to verify signatures involving program $\mathcal{P}$ and any dataset Δ *efficiently*. In the context of this paper, "efficiently" means that the cost of verifying signatures is less than computing program $\mathcal{P}$, and that concise verification key $\mathsf{VK}_{\mathcal{P}}$ is reused indefinitely. This implies that the cost of computing the concise key $\mathsf{VK}_{\mathcal{P}}$ is *amortized* over the unlimited number of verifications that one can carry out for program $\mathcal{P}$ and different datasets.

Formally, efficient verification is achieved by dividing the algorithm Verify into two sub-algorithms:

- $\mathsf{OffVerify}(\mathsf{VK}, \mathcal{P}) \rightarrow \mathsf{VK}_{\mathcal{P}}$: This algorithm takes as inputs a verification key VK and a description of labeled program $\mathcal{P}$, and computes a concise verification key $\mathsf{VK}_{\mathcal{P}}$ which will be used later to verify signatures related to program $\mathcal{P}$.

– $\mathsf{OnVerify}(\mathsf{VK}_{\mathcal{P}}, \Delta, m, \sigma) \rightarrow b$: It is a deterministic algorithm, which given a concise verification key $\mathsf{VK}_{\mathcal{P}}$, a data set identifier Δ, a message $m \in \mathbb{I}$ and a signature σ, outputs a bit $b \in \{0, 1\}$ such that: $b = 1$, if algorithm $\mathsf{OnVerify}$ decides that m is the correct output of multi-labeled program $(\mathcal{P}, \Delta)$. If not, then $b = 0$.

3 Preliminaries

The starting point of our proposal is the fact that if a vector $\boldsymbol{u}$ is an eigenvector of a matrix M, then there exists a scalar λ such that $M\boldsymbol{u} = \lambda\boldsymbol{u}$. In light of this equality, we can easily show that for any pair of matrices $M^{(1)}, M^{(2)}$ admitting vector $\boldsymbol{u}$ as an eigenvector and having λ_1 and λ_2 as the corresponding eigenvalues, the following equalities hold:

– $(M^{(1)} + M^{(2)})\boldsymbol{u} = (\lambda_1 + \lambda_2)\boldsymbol{u}$, meaning that $\lambda_1 + \lambda_2$ is the eigenvalue of matrix $M^{(1)} + M^{(2)}$ associated with eigenvector $\boldsymbol{u}$;
– similarly, $M^{(1)}M^{(2)}\boldsymbol{u} = M^{(2)}M^{(1)}\boldsymbol{u} = (\lambda_1\lambda_2)\boldsymbol{u}$, which entails that $\lambda_1\lambda_2$ is the eigenvalue of matrices $M^{(1)}M^{(2)}$ and $M^{(2)}M^{(1)}$, associated with eigenvector $\boldsymbol{u}$;

On account of these two observations, we map the homomorphic signature of a pair of message m and label τ to a matrix

$$M = \begin{bmatrix} m & \gamma \\ 0 & \lambda \end{bmatrix}$$

such that λ is computed as function of τ, and γ is generated in such a way that λ is the eigenvalue of matrix M associated with a secret vector $\boldsymbol{u} = (x, y)$.

Therefore, the homomorphic signature of $m^{(1)} + m^{(2)}$ is mapped to matrix $M^{(1)} + M^{(2)}$, and the verification of such a signature consists of checking that $\lambda_1 + \lambda_2$ is the eigenvalue of matrix $M^{(1)} + M^{(2)}$ associated with eigenvector $\boldsymbol{u}$. Along the same lines, the homomorphic signature of $m^{(1)}m^{(2)}$ is mapped to matrix $M^{(1)}M^{(2)}$, and the verification of this signature is performed by verifying whether $\lambda_1\lambda_2$ is the eigenvalue of matrix $M^{(1)}M^{(2)}$ associated with vector $\boldsymbol{u}$. More precisely, we define the homomorphic signature of a pair (m, τ) as a tuple (m, Λ_1, Γ_1), where $\Lambda_1 = g_1^{\lambda}$ and $\Gamma_1 = g_1^{\gamma}$. It follows that the signature of $m^{(1)} + m^{(2)}$ is straightforward and defined as:

$$(m^{(1)} + m^{(2)}, \Lambda_1^{(1)}\Lambda_1^{(2)}, \Gamma_1^{(1)}\Gamma_1^{(2)})$$

whereas the signature of $m^{(1)}m^{(2)}$ involves a *a leveled multilinear map* e and corresponds to:

$$(m^{(1)}m^{(2)}, e(\Lambda_1^{(1)}, \Lambda_1^{(2)}), e(g_1^{m^{(1)}}, \Gamma_1^{(2)})e(\Gamma_1^{(1)}, \Lambda_1^{(2)}))$$

We note that this homomorphic signature is weakly sound against **Type II** forgeries. Namely, it is secure against adversaries that issue their signature queries before receiving the public verification key (cf. Section 4.3). In order to make this signature *adaptively* sound against **Type II** forgeries, the signer is required to do the following whenever she wants to sign a pair (m, τ): (i) Generate a random number θ_1 and evaluate polynomial $t(z) = \theta_1 z + m$ at a secret point α; (ii) sign $(t(\alpha), \tau)$ using the weakly

secure signature to get the tuple $(t(\alpha), \Lambda_1, \Gamma_1)$; (iii) set the homomorphic signature of (m, τ) to $([m, \theta_1], \Lambda_1, \Gamma_1)$.

Finally, the signer thwarts **Type I** forgeries by signing dataset identifiers using a *digital signature*, while she counters **Type III** forgeries by using *aggregate signatures* to authenticate input labels.

Before moving to the description of our signature, we first provide a short overview on multilinear maps and aggregate signatures.

3.1 Leveled Multilinear Maps

Definition 6. *Let $\mathbb{G}_1$, $\mathbb{G}_2$, ...,and $\mathbb{G}_d$ be d groups of large prime order p, generated on input of a security parameter 1^κ ($p > 2^\kappa$) and d. Let P_i be a canonical generator of group $\mathbb{G}_i$.*

A d-leveled multilinear map is a set of bilinear maps $e_{i,j} : \mathbb{G}_i \times \mathbb{G}_j \rightarrow \mathbb{G}_{i+j}$ whereby $i \geq 1$, $j \geq 1$ and $i + j \leq d$, with the following property:

$$\forall \alpha, \beta \in \mathbb{F}_p, e_{i,j}(P_i^\alpha, P_j^\beta) = P_{i+j}^{\alpha\beta}$$

For the sake of better readability, we omit the indices i and j from $e_{i,j}$. We also assume that all generators P_i are computed from P_1 using bilinear maps repeatedly: $P_i = e(P_1, P_{i-1})$.

We hereby state the assumption on which the security of our solution relies.

Definition 7 (MDH Inversion Assumption [6]). *Let $\mathbb{G}_1$, $\mathbb{G}_2$, ...,$\mathbb{G}_{d+1}$ be $d+1$ groups of large prime order p, generated on input of a security parameter 1^κ ($p > 2^\kappa$) and d. Let P_1 be a canonical generator of group $\mathbb{G}_1$ and e a $(d+1)$-leveled multilinear map, i.e. $\forall 1 \leq i \leq d, P_{i+1} = e(P_1, P_i)$.*

We say that multilinear Diffie-Hellman inversion (MDHI) assumption holds, if given $(P_1, P_1^\alpha) \in \mathbb{G}_1 \times \mathbb{G}_1$, where $\alpha \in \mathbb{F}_p^*$, the probability of finding $P_{d+1}^{\alpha^{-1}}$ is negligible, that is:

$$\Pr[\mathcal{A} \rightarrow P_{d+1}^{\alpha^{-1}} | (P_1, P_1^\alpha)] \leq \epsilon(\kappa)$$

where ϵ is a negligible function.

3.2 Aggregate Signatures

We provide herein a quick overview of a *simplified* variant of the aggregate signature proposed in [7]. Similarly to the signature of [7], this simplified variant comprises four algorithms:

$\mathsf{KeyGen}_{\mathsf{Agg}}(1^\kappa) \rightarrow (\mathsf{SK}_{\mathsf{Agg}}, \mathsf{PK}_{\mathsf{Agg}}, \mathsf{param}_{\mathsf{Agg}})$ On input of a security parameter 1^κ, $\mathsf{KeyGen}_{\mathsf{Agg}}$ proceeds as follows:

- It picks two groups $\mathbb{G}$ and $\mathbb{G}_T$ of a large prime order p that admit a bilinear pairing $\hat{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$.
- It selects a secret key $\mathsf{SK}_{\mathsf{Agg}} \in \mathbb{F}_p^*$, chooses a random generator P of group $\mathbb{G}$, and sets public key $\mathsf{PK}_{\mathsf{Agg}}$ to $P^{\mathsf{SK}_{\mathsf{Agg}}}$.

– It selects a cryptographic hash function $H : \{0,1\}^* \to \mathbb{G}$ and defines $\mathsf{param}_{\mathsf{Agg}}$ as $(p, H, \mathbb{G}, \mathbb{G}_T, \hat{e}, P)$.

$\mathsf{Sign}_{\mathsf{Agg}}(\mathsf{SK}_{\mathsf{Agg}}, \mu) \to \Psi$ Given a message $\mu \in \{0,1\}^*$, $\mathsf{Sign}_{\mathsf{Agg}}$ outputs a signature $\Psi = H(\mu)^{\mathsf{SK}_{\mathsf{Agg}}}$.

$\mathsf{Aggregate}_{\mathsf{Agg}}(\boldsymbol{\Psi}) \to \Psi$ Given a vector of n signatures $\boldsymbol{\Psi} = (\Psi^{(1)}, ..., \Psi^{(n)})$, $\mathsf{Aggregate}_{\mathsf{Agg}}$ outputs aggregate signature $\Psi = \prod_{i=1}^{n} \Psi^{(i)}$.

$\mathsf{Verify}_{\mathsf{Agg}}(\mathsf{PK}_{\mathsf{Agg}}, \boldsymbol{\mu}, \Psi) \to b \in \{0,1\}$ When provided with public key $\mathsf{PK}_{\mathsf{Agg}}$, a vector of messages $\boldsymbol{\mu} = (\mu^{(1)}, ..., \mu^{(n)})$, and an aggregate signature Ψ, $\mathsf{Verify}_{\mathsf{Agg}}$ checks whether the following equality holds:

$$\hat{e}(\Psi, P) = \hat{e}(\prod_{i=1}^{n} H(\mu^{(i)}), \mathsf{PK}_{\mathsf{Agg}})$$

If so, then $\mathsf{Verify}_{\mathsf{Agg}}$ accepts the aggregate signature and accordingly outputs $b = 1$; otherwise, it rejects the signature and outputs $b = 0$.

Using a similar argument to Boneh et al. [7]'s, one can easily show that this variant of aggregate signatures is adaptively secure in the *random oracle model*, under the co-CDH assumption in $\mathbb{G}$.

4 A Weakly Secure Homomorphic Signature

The homomorphic signature we propose in this paper is suitable for programs that evaluate multivariate polynomials over a finite field $\mathbb{F}_p$. Such programs can be expressed using arithmetic circuits. An arithmetic circuit is composed of addition and multiplication gates, such that each addition (multiplication resp.) gate takes two inputs and returns the sum (the product resp.) of these two inputs. Furthermore, we recall that arithmetic circuits have a measure called *degree* that is assigned to the inputs/outputs of their gates. Namely, constants in the circuit are assigned a degree 0, whereas initial inputs of the circuits has a degree 1. Moreover, the degree of an addition gate (resp. multiplication gate) is defined as the maximum of the degree of its inputs (resp. the sum of its inputs). Accordingly, the *degree* of an arithmetic circuit is defined as the degree of the output gate of the circuit. This entails that the degree of an arithmetic circuit evaluating a d-degree polynomial is d.

4.1 Description

In this section, we provide the detailed description of the algorithms underlying our weakly secure homomorphic signature.

$\mathsf{KeyGen}(1^\kappa, d, \mathcal{L}) \to (\mathsf{SK}, \mathsf{VK}, \mathsf{param})$ Given a security parameter 1^κ, an upper-bound d of the degree of circuits supported by the signature, and a set of admissible labels [1] $\mathcal{L} = \{\tau_1, ..., \tau_N\}$ in $\{0,1\}^*$, algorithm KeyGen proceeds as follows:

[1] Since $|\mathcal{L}| = N$, the size of datasets supported by our signature cannot exceed N.

- It selects $(d+1)$-leveled linear groups $\mathbb{G}_1, ..., \mathbb{G}_{d+1}$ of prime order p. We denote elements lying in group $\mathbb{G}_i$ with capital letters and subscript i. Moreover, for any element $P_1 \in \mathbb{G}_1$, we denote by P_i the element $e(P_1, P_{i-1}) \in \mathbb{G}_i$, $2 \leq i \leq d+1$, and namely, for all $i, j \geq 1$ and $i + j \leq d + 1$, we let $e(P_i, P_j) = P_{i+j}$.
- It selects a random key $K \in \mathbb{F}_p^*$ and a keyed hash function $F : \mathbb{F}_p^* \times \{0,1\}^* \times \{0,1\}^* \to \mathbb{F}_p^*$.
- It picks a random generator P_1 in $\mathbb{G}_1$, generates N random numbers $\lambda_i = F(K, \tau_i, 0)$, $1 \leq i \leq N$ and computes $\Lambda_1^{(\tau_i)} = P_1^{\lambda_i}$.
- It selects a pair of secret and public keys $(\mathsf{SK}_{\mathsf{Dig}}, \mathsf{PK}_{\mathsf{Dig}})$ for a digital signature Σ_{Dig}.
- It selects two groups $\mathbb{G}$ and $\mathbb{G}_T$ of the same prime order p that admit a bilinear pairing $\hat{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. Then it picks a random generator P of $\mathbb{G}$ and a cryptographic hash function $H : \{0,1\}^* \to \mathbb{G}$. This generator and cryptographic hash function will be used to implement an aggregate signature Σ_{Agg} as depicted in Section 3.2.
- Finally, it outputs the following:

$$\begin{aligned} \mathsf{SK} &= (F, K, \mathsf{SK}_{\mathsf{Dig}}) \\ \mathsf{VK} &= (\mathsf{PK}_{\mathsf{Dig}}, \{(\tau_i, \Lambda_1^{(\tau_i)})\}_{i=1}^N) \\ \mathsf{param} &= (p, \{\tau_i\}_{i=1}^N, H, e, \hat{e}, \{\mathbb{G}_i\}_{i=1}^{d+1}, \mathbb{G}, \mathbb{G}_T, P_1, P) \end{aligned}$$

  Note that public parameters param implicitly specify the set of admissible functions $\mathcal{F}$ and the set of admissible inputs $\mathbb{I}$, which are respectively the set of n-variate functions ($n \leq N$) that can be implemented using circuits of degree $k \leq d$, and the finite field $\mathbb{F}_p$.

$\mathsf{Sign}(\mathsf{SK}, \Delta, \tau, m) \to \sigma$ In a nutshell, algorithm Sign computes three types of signatures: A digital signature Σ_{Dig} to counter **Type I** forgery, an aggregate signature Σ_{Agg} to circumvent **Type III** forgery, and finally, a homomorphic signature Σ_{Hom} to preclude **Type II** forgery. Indeed, given secret key $\mathsf{SK} = (F, K, \mathsf{SK}_{\mathsf{Dig}})$, dataset identifier Δ, a label $\tau \in \mathcal{L}$, and a message $m \in \mathbb{F}_p$, algorithm Sign runs four subroutines:

- $\Delta\mathsf{KeyGen}(F, K, \Delta)$: Given keyed hash function F, secret key K, and dataset identifier Δ, $\Delta\mathsf{KeyGen}$ proceeds as follows:
  - It generates a secret key $\mathsf{SK}_{\mathsf{Agg}} \in \mathbb{F}_p^*$ for aggregate signature Σ_{Agg} by computing $F(K, \Delta, 1)$. Once secret key $\mathsf{SK}_{\mathsf{Agg}}$ is produced, $\Delta\mathsf{KeyGen}$ computes the corresponding public key $\mathsf{PK}_{\mathsf{Agg}} = P^{\mathsf{SK}_{\mathsf{Agg}}}$.
  - It computes $F(K, \Delta, 2)$ and $F(K, \Delta, 3)$ to generate the secret key $\mathsf{SK}_{\mathsf{Hom}} = (x, y) \in \mathbb{F}_p^* \times \mathbb{F}_p^*$ for a homomorphic signature Σ_{Hom}. After the generation of this secret key, $\Delta\mathsf{KeyGen}$ computes $X_1 = {P_1}^x$ and $Y_1 = {P_1}^y$ and defines the public key of homomorphic signature Σ_{Hom} as $\mathsf{PK}_{\mathsf{Hom}} = (X_1, Y_1)$.
- $\Sigma_{\mathsf{Dig}}(\mathsf{SK}_{\mathsf{Dig}}, \Delta, \mathsf{PK}_{\mathsf{Agg}}, \mathsf{PK}_{\mathsf{Hom}})$: This subroutine takes as inputs secret key $\mathsf{SK}_{\mathsf{Dig}}$, dataset identifier Δ, public key $\mathsf{PK}_{\mathsf{Agg}}$, and public key $\mathsf{PK}_{\mathsf{Hom}}$, and computes a digital signature Ω of tuple $(\Delta, \mathsf{PK}_{\mathsf{Agg}}, \mathsf{PK}_{\mathsf{Hom}})$.
- $\Sigma_{\mathsf{Agg}}(F, K, \mathsf{SK}_{\mathsf{Agg}}, \tau)$: This subroutine computes $\Lambda_1 = P_1^{\lambda}$ given $\lambda = F(K, \tau, 0)$, and uses secret key $\mathsf{SK}_{\mathsf{Agg}}$ to compute signature $\Psi = H(\Lambda_1)^{\mathsf{SK}_{\mathsf{Agg}}} \in \mathbb{G}$.

– $\Sigma_{\mathsf{Hom}}(\mathsf{SK}_{\mathsf{Hom}}, \Lambda_1, m)$: Given $\mathsf{SK}_{\mathsf{Hom}} = (x, y)$, this subroutine computes

$$\Gamma_1 = \left(\frac{\Lambda_1}{{P_1}^m}\right)^{xy^{-1}}$$

and finally, defines the homomorphic signature of tuple (m, τ, Δ) as $\Upsilon = (m, \Lambda_1, \Gamma_1)$.

Remark 1. Assume that $\Lambda_1 = {P_1}^{\lambda}$ and $\Gamma_1 = {P_1}^{\gamma}$, for $(\lambda, \gamma) \in \mathbb{F}_p^* \times \mathbb{F}_p$.

Since $\Gamma_1 = \left(\frac{\Lambda_1}{{P_1}^m}\right)^{xy^{-1}}$, we can easily show that λ is an eigenvalue of matrix

$$M = \begin{bmatrix} m & \gamma \\ 0 & \lambda \end{bmatrix}$$

associated with vector $\boldsymbol{u} = (x, y)$, and consequently, we can deduce the following equality:

$$M\boldsymbol{u} = \lambda\boldsymbol{u}$$

This remark comes in handy when we prove evaluation correctness in the following section.

At the end of its execution, algorithm Sign returns the homomorphic signature $\sigma = (\mathsf{param}_{\Delta}, \Omega, \Psi, \Upsilon)$, where

$$\mathsf{param}_{\Delta} = (\Delta, \mathsf{PK}_{\mathsf{Agg}}, \mathsf{PK}_{\mathsf{Hom}})$$

$\mathsf{Eval}(\mathsf{VK}, f, \boldsymbol{\sigma}) \rightarrow \sigma$ On inputs of public verification key VK, an n-variate function f, and a vector $\boldsymbol{\sigma}$ of n homomorphic signatures $\sigma^{(l)} = (\mathsf{param}_{\Delta}, \Omega, \Psi^{(l)}, \Upsilon^{(l)})$ whereby each signature $\sigma^{(l)}$ authenticates a message $m^{(l)}$ for all $1 \leq l \leq n$, algorithm Eval computes signature $\sigma = (\mathsf{param}_{\Delta}, \Omega, \Psi, \Upsilon)$ in two steps:

Computation of homomorphic signature Υ: Eval computes Υ by evaluating the arithmetic circuit $\mathcal{C}_f$ of function f on input $(\Upsilon^{(1)}, ..., \Upsilon^{(n)})$. This evaluation of circuit $\mathcal{C}_f$ is achieved by running the following subroutines:

– $\mathsf{GateEval}_+(\Upsilon^{(1)}, \Upsilon^{(2)})$[2]: Without loss of generality, we assume that $\Upsilon^{(1)} = (m^{(1)}, \Lambda_i^{(1)}, \Gamma_i^{(1)}) \in \mathbb{F}_p \times \mathbb{G}_i^2$ and $\Upsilon^{(2)} = (m^{(2)}, \Lambda_i^{(2)}, \Gamma_i^{(2)}) \in \mathbb{F}_p \times \mathbb{G}_i^2$. In other words, we assume that $\Upsilon^{(1)}$ and $\Upsilon^{(2)}$ are inputs to $\mathsf{GateEval}_+$ of the same degree i [3].

Algorithm Eval invokes $\mathsf{GateEval}_+$ when it encounters an addition gate. Therefore, given $(\Upsilon^{(1)}, \Upsilon^{(2)})$, $\mathsf{GateEval}_+$ computes $m = m^{(1)} + m^{(2)}$ and

$$\Lambda_i = \Lambda_i^{(1)}\Lambda_i^{(2)} \; ; \; \Gamma_i = \Gamma_i^{(1)}\Gamma_i^{(2)}$$

At the end of its execution, $\mathsf{GateEval}_+$ outputs $\Upsilon = (m, \Lambda_i, \Gamma_i)$.

[2] For ease of exposition, we abuse the notations here and we denote by $\Upsilon^{(1)}$ and $\Upsilon^{(2)}$ the inputs of $\mathsf{GateEval}_{+,\times,c}$.

[3] Note that $\Upsilon^{(1)}$ and $\Upsilon^{(2)}$ could be inputs of different degrees i and j respectively, that is, $\Upsilon^{(2)} = (m^{(2)}, \Lambda_j^{(2)}, \Gamma_j^{(2)}) \in \mathbb{F}_p \times \mathbb{G}_j^2$. Still, if we assume that $j < i$, we can transform $\Upsilon^{(2)}$ into an input of degree i by computing $\Lambda_i^{(2)} = e(\Lambda_j^{(2)}, P_{i-j})$ and $\Gamma_i^{(2)} = e(\Gamma_j^{(2)}, P_{i-j})$.

– $\mathsf{GateEval}_c(\Upsilon^{(1)}, c)$: Algorithm Eval calls $\mathsf{GateEval}_c$ when it wants to evaluate a gate for multiplication by a constant. On inputs of a homomorphic signature $\Upsilon^{(1)} = (m^{(1)}, \Lambda_i^{(1)}, \Gamma_i^{(1)}) \in \mathbb{F}_p \times \mathbb{G}_i^2$ and a constant c, $\mathsf{GateEval}_c$ computes $m = cm^{(1)}$ and

$$\Lambda_i = (\Lambda_i^{(1)})^c \; ; \; \Gamma_i = (\Gamma_i^{(1)})^c$$

Later, $\mathsf{GateEval}_c$ returns $\Upsilon = (m, \Lambda_i, \Gamma_i)$.

– $\mathsf{GateEval}_\times(\Upsilon^{(1)}, \Upsilon^{(2)})$: Without loss of generality, assume that $\Upsilon^{(1)} = (m^{(1)}, \Lambda_i^{(1)}, \Gamma_i^{(1)}) \in \mathbb{F}_p \times \mathbb{G}_i^2$ and $\Upsilon^{(2)} = (m^{(2)}, \Lambda_j^{(2)}, \Gamma_j^{(2)}) \in \mathbb{F}_p \times \mathbb{G}_j^2$.
Algorithm Eval executes $\mathsf{GateEval}_\times$ when it wants to evaluate a multiplication gate. Hence, given $(\Upsilon^{(1)}, \Upsilon^{(2)})$, $\mathsf{GateEval}_\times$ computes $m = m^{(1)}m^{(2)}$ and

$$\Lambda_{i+j} = e(\Lambda_i^{(1)}, \Lambda_j^{(2)})$$
$$\Gamma_{i+j} = e(P_i^{m^{(1)}}, \Gamma_j^{(2)}) e(\Gamma_i^{(1)}, \Lambda_j^{(2)})$$

$\mathsf{GateEval}_\times$ then outputs $\Upsilon = (m, \Lambda_{i+j}, \Gamma_{i+j})$.

Computation of aggregate signature Ψ: Algorithm Eval computes the aggregate signature Ψ corresponding to function f by evaluating a modified circuit $\tilde{\mathcal{C}}_f$ on inputs of aggregate signatures $(\Psi^{(1)}, ..., \Psi^{(n)})$. The modified circuit $\tilde{\mathcal{C}}_f$ is generated from circuit $\mathcal{C}_f$ as follows:

– Each multiplication and addition gate in circuit $\mathcal{C}_f$ is transformed into a multiplication operation in $\mathbb{G}$ in circuit $\tilde{\mathcal{C}}_f$;
– a multiplication by a constant in circuit $\mathcal{C}_f$ is omitted in circuit $\tilde{\mathcal{C}}_f$.

Algorithm Eval concludes its work by outputting

$$\sigma = (\mathsf{param}_\Delta, \Omega, \Psi, \Upsilon)$$

$\mathsf{OffVerify}(\mathsf{VK}, \mathcal{P}) \rightarrow \mathsf{VK}_\mathcal{P}$ This algorithm takes as inputs verification key $\mathsf{VK} = (\mathsf{PK}_{\mathsf{Dig}}, \{(\tau_i, \Lambda_1^{(\tau_i)})\}_{i=1}^N)$ and labeled program $\mathcal{P} = (f, \tau_1, ..., \tau_n)$. Here we assume that function f is implemented as a circuit $\mathcal{C}_f$ of degree k.

Algorithm OffVerify first evaluates the arithmetic circuit $\mathcal{C}_f$ on tuple $(\Lambda_1^{(\tau_1)}, ..., \Lambda_1^{(\tau_n)})$. More precisely, OffVerify transforms $\mathcal{C}_f$ into a circuit $\bar{\mathcal{C}}_f$ that operates on elements from $\mathbb{G}_i$, $1 \leq i \leq d+1$, as follows:

– Addition gates of degree i in circuit $\mathcal{C}_f$ are replaced by multiplications in $\mathbb{G}_i$ in circuit $\bar{\mathcal{C}}_f$;
– gates of degree i for multiplication by a constant in circuit $\mathcal{C}_f$ are transformed in circuit $\bar{\mathcal{C}}_f$ into exponentiations in $\mathbb{G}_i$ by the same constants;
– finally, multiplication gates with inputs of degree i and degree j in circuit $\mathcal{C}_f$ are expressed in circuit $\bar{\mathcal{C}}_f$ as bilinear maps between elements lying in $\mathbb{G}_i$ and $\mathbb{G}_j$.

In the rest of the paper, we denote by $f(\Lambda_1^{(\tau_1)}, ..., \Lambda_1^{(\tau_n)})$ the output of circuit $\bar{\mathcal{C}}_f$ when evaluated on $(\Lambda_1^{(\tau_1)}, ..., \Lambda_1^{(\tau_n)})$.

Next, algorithm $\mathsf{OffVerify}$ evaluates the modified circuit $\tilde{\mathcal{C}}_f$ (cf. Algorithm Eval) on inputs $(H(\Lambda_1^{(\tau_1)}), ..., H(\Lambda_1^{(\tau_n)})) \in \mathbb{G}^n$. This yield an aggregated hash $H_{\mathcal{P}}$.
At the end of its execution, algorithm $\mathsf{OffVerify}$ outputs concise verification key

$$\mathsf{VK}_{\mathcal{P}} = (\mathsf{PK}_{\mathsf{Dig}}, H_{\mathcal{P}}, f(\Lambda_1^{(\tau_1)}, ..., \Lambda_1^{(\tau_n)}))$$

$\mathsf{OnVerify}(\mathsf{VK}_{\mathcal{P}}, \Delta, m, \sigma) \rightarrow b$ On input of concise verification key $\mathsf{VK}_{\mathcal{P}}$, dataset identifier Δ, message $m \in \mathbb{F}_p$, and signature $\sigma = (\mathsf{param}_{\Delta}, \Omega, \Psi, \Upsilon)$, $\mathsf{OnVerify}$ proceeds as follows:

- It parses $\mathsf{VK}_{\mathcal{P}}$ as $(\mathsf{PK}_{\mathsf{Dig}}, H_{\mathcal{P}}, f(\Lambda^{(\tau_1)}, ..., \Lambda^{(\tau_n)}))$, dataset parameters param_{Δ} as $(\Delta, \mathsf{PK}_{\mathsf{Agg}}, \mathsf{PK}_{\mathsf{Hom}})$ and homomorphic signature σ as $(m, \Omega, \Psi, \Upsilon)$.
- Using public key $\mathsf{PK}_{\mathsf{Dig}}$, algorithm $\mathsf{OnVerify}$ checks whether Ω is a valid signature of message param_{Δ}. If not, then $\mathsf{OnVerify}$ returns $b = 0$.
- Otherwise, given public key $\mathsf{PK}_{\mathsf{Agg}}$, algorithm $\mathsf{OnVerify}$ checks whether the following equality holds:

$$\hat{e}(H_{\mathcal{P}}, \mathsf{PK}_{\mathsf{Agg}}) = \hat{e}(\Psi, P) \tag{1}$$

  If the above equality is not satisfied, then $\mathsf{OnVerify}$ returns $b = 0$. Otherwise, it moves on to the next step.
- Since program $\mathcal{P}$ is implemented as a circuit of degree k, algorithm $\mathsf{OnVerify}$ parses homomorphic signature Υ as a tuple $(m, \Lambda_k, \Gamma_k) \in \mathbb{F}_p \times \mathbb{G}_k^2$, and given public key $\mathsf{PK}_{\mathsf{Hom}} = (X_1, Y_1) \in \mathbb{G}_1^2$, it verifies whether the following equalities are true:

$$\Lambda_k = f(\Lambda_1^{(\tau_1)}, ..., \Lambda_1^{(\tau_n)}) \tag{2}$$

$$e(X_1, \Lambda_k) = e(X_1, P_k)^m e(Y_1, \Gamma_k) \tag{3}$$

  If both equalities hold, then $\mathsf{OnVerify}$ outputs $b = 1$; otherwise it outputs $b = 0$.

4.2 Correctness

In this section, we demonstrate that our solution satisfies the properties of authentication correctness and evaluation correctness.

Theorem 1. *The homomorphic signature described above ensures authentication correctness.*

Proof. Let $\sigma = (\mathsf{param}_{\Delta}, \Omega, \Psi, \Upsilon)$ be a signature output by algorithm Sign on input of message m, label τ and dataset identifier Δ, and let $\Lambda_1 = P_1^{\lambda}$, where $\lambda = F(K, \tau, 0)$. This implies the following:

$$\begin{aligned}
\mathsf{param}_{\Delta} &= (\Delta, \mathsf{PK}_{\mathsf{Agg}}, \mathsf{PK}_{\mathsf{Hom}}) \\
\mathsf{PK}_{\mathsf{Agg}} &= P^{\mathsf{SK}_{\mathsf{Agg}}} \\
\mathsf{PK}_{\mathsf{Hom}} &= (X_1, Y_1) = ({P_1}^x, {P_1}^y) \\
\Omega &\leftarrow \Sigma_{\mathsf{Dig}}(\mathsf{SK}, \mathsf{param}_{\Delta})
\end{aligned}$$

$$\Psi = H(\Lambda_1)^{\mathsf{SK}_{\mathsf{Agg}}}$$
$$\Upsilon = (m, \Lambda_1, \Gamma_1)$$
$$\Gamma_1 = \left(\frac{\Lambda_1}{{P_1}^m}\right)^{xy^{-1}}$$

Hence, signature Ω will pass the first verification of $\mathsf{OnVerify}$. Similarly, signature Ψ will verify Equation 1, and be accepted as a valid aggregate signature.

Now moving to the last verification step which consists of checking whether Equations 2 and 3 hold for the pair $(\mathcal{P}, \Upsilon) = ((\mathcal{I}, \tau), \Upsilon)$, where $\mathcal{I}$ is the canonical identity.

Since $\Lambda_1 = \mathcal{I}(\Lambda_1)$, Equation 2 is satisfied.

Additionally, since $\Gamma_1 = \left(\frac{\Lambda_1}{{P_1}^m}\right)^{xy^{-1}}$, we obtain:

$$\begin{aligned} e(Y_1, \Gamma_1) &= e\left({P_1}^y, \left(\frac{\Lambda_1}{{P_1}^m}\right)^{xy^{-1}}\right) \\ &= e\left(P_1, \left(\frac{\Lambda_1}{{P_1}^m}\right)^{x}\right) = e\left({P_1}^x, \frac{\Lambda_1}{{P_1}^m}\right) \\ &= e\left(X_1, \frac{\Lambda_1}{{P_1}^m}\right) = \frac{e(X_1, \Lambda_1)}{e(X_1, P_1)^m} \end{aligned}$$

Meaning that Equation 3 also is satisfied.

Theorem 2. *The homomorphic signature described in Section 4.1 ensures evaluation correctness.*

Proof. Let $\boldsymbol{\sigma} = (\sigma^{(1)}, ..., \sigma^{(n)})$ be a vector of n homomorphic signatures $\sigma^{(l)}$, such that each signature $\sigma^{(l)}$ $(1 \leq l \leq n)$ *successfully* authenticates a message $m^{(l)} \in \mathbb{F}_p$ as the output of some multi-labeled program $(\mathcal{P}_l, \Delta)$.

For ease of exposition, we assume that $\mathcal{P}_l = (f_l, \tau_1, ..., \tau_n)$ for all $1 \leq l \leq n$. That is, each $\mathcal{P}_l$ evaluates an n-variate function f_l and as such, it is associated with n labels $(\tau_1, ..., \tau_n)$. We also assume that function f_l is a a circuit $\mathcal{C}_f$ of degree k, and we let:

$$\sigma^{(l)} = (\mathsf{param}_{\Delta}, \Omega, \Psi^{(l)}, \Upsilon^{(l)})$$
$$\Upsilon^{(l)} = (m^{(l)}, \Lambda_k^{(l)}, \Gamma_k^{(l)}) \in \mathbb{F}_p \times \mathbb{G}_k^2$$

Since $\sigma^{(l)}$ successfully authenticates message $m^{(l)}$, we conclude the following:

- Ω is a valid digital signature of param_{Δ}.
- $\Psi^{(l)}$ is a valid aggregate signature for program $\mathcal{P}_l$, and thereby verifies: $\hat{e}(H_{\mathcal{P}_l}, \mathsf{PK}_{\mathsf{Agg}}) = \hat{e}(\Psi^{(l)}, P)$, where $H_{\mathcal{P}_l}$ is the aggregate hash of program $\mathcal{P}_l$ (cf. Algorithm $\mathsf{OnVerify}$, Section 4.1).
- Finally, $\Upsilon^{(l)}$ is a correct homomorphic signature, namely:

$$\Lambda_k^{(l)} = f_l(\Lambda_1^{(\tau_1)}, ..., \Lambda_1^{(\tau_n)})$$
$$e(X_1, \Lambda_k) = e(X_1, P_k)^{m^{(l)}} e(Y_1, \Gamma_k)$$

In what follows, we show that if algorithm Eval is executed correctly on inputs $(\mathsf{VK}, f, \boldsymbol{\sigma})$ for some function f, then the output $\sigma = (\mathsf{param}_{\Delta}, \Omega, \Psi, \Upsilon)$ returned by algorithm Eval is going to be accepted by algorithm Verify (i.e. the combination of algorithms $\mathsf{OnVerify}$ and $\mathsf{OffVerify}$).

Correctness of digital signature Ω: Since algorithm Eval does not change signature Ω, the latter will always verify as a valid signature for the tuple $\mathsf{param}_{\Delta} = (\Delta, \mathsf{PK}_{\mathsf{Agg}}, \mathsf{PK}_{\mathsf{Hom}})$.

Correctness of aggregate signature Ψ: We remind the reader that aggregate signature Ψ is obtained by evaluating a modified circuit $\tilde{\mathcal{C}}_f$ on input $(\Psi^{(1)}, ..., \Psi^{(n)})$.

Since for all $1 \leq l \leq n$, $\Psi^{(l)}$ is a valid aggregate signature for program $\mathcal{P}_l$, we know that $\Psi^{(l)} = H_{\mathcal{P}_l}^{\mathsf{SK}_{\mathsf{Agg}}}$, which entails:

$$\Psi = \tilde{\mathcal{C}}_f(\Psi^{(1)}, ..., \Psi^{(n)}) = \tilde{\mathcal{C}}_f(H_{\mathcal{P}_1}^{\mathsf{SK}_{\mathsf{Agg}}}, ..., H_{\mathcal{P}_n}^{\mathsf{SK}_{\mathsf{Agg}}})$$

Moreover, since circuit $\tilde{\mathcal{C}}_f$ comprises only multiplication operations, it satisfies the following:

$$\tilde{\mathcal{C}}_f(H_{\mathcal{P}_1}^{\mathsf{SK}_{\mathsf{Agg}}}, ..., H_{\mathcal{P}_n}^{\mathsf{SK}_{\mathsf{Agg}}}) = \tilde{\mathcal{C}}_f(H_{\mathcal{P}_1}, ..., H_{\mathcal{P}_n})^{\mathsf{SK}_{\mathsf{Agg}}}$$

and hence: $\Psi = \tilde{\mathcal{C}}_f(H_{\mathcal{P}_1}, ..., H_{\mathcal{P}_n})^{\mathsf{SK}_{\mathsf{Agg}}} = H_{\mathcal{P}}^{\mathsf{SK}_{\mathsf{Agg}}}$, where $H_{\mathcal{P}}$ is the aggregate hash of multi-labeled program $(\mathcal{P}, \Delta)$ defined as the composition of function f and programs $(\mathcal{P}_l, \Delta)$, $1 \leq l \leq n$.

From the preceding equation, we deduce that Equality 1 always holds for the aggregate signature Ψ returned by algorithm Eval.

Correctness of homomorphic signature Υ: We prove here that the output Υ of $\mathsf{GateEval}_{+,c,\times}$ always verifies Equations 2 and 3. We only show here the correctness of $\mathsf{GateEval}_{\times}$. A similar argument can be used to prove the correctness of $\mathsf{GateEval}_{+}$ and $\mathsf{GateEval}_{c}$.

Without loss of generality, let $\Upsilon^{(1)} = (m^{(1)}, \Lambda_i^{(1)}, \Gamma_i^{(1)}) \in \mathbb{F}_p \times \mathbb{G}_i^2$ and $\Upsilon^{(2)} = (m^{(2)}, \Lambda_j^{(2)}, \Gamma_j^{(2)}) \in \mathbb{F}_p \times \mathbb{G}_j^2$. We demonstrate in what follows that the output $\Upsilon = (m, \Lambda_{i+j}, \Gamma_{i+j})$ of $\mathsf{GateEval}_{\times}$ satisfies Equations 2 and 3.

Along these lines, we first remind the reader that: $m = m^{(1)}m^{(2)}$, $\Lambda_{i+j} = e(\Lambda_i, \Lambda_j)$, and $\Gamma_{i+j} = e(P_i{}^{m^{(1)}}, \Gamma_j^{(2)})e(\Gamma_i^{(1)}, \Lambda_j^{(2)})$.

Correctness of Λ_{i+j}. Note that $\mathsf{GateEval}_{\times}$ computes Λ_i as a bilinear map $e(\Lambda_i^{(1)}, \Lambda_j^{(2)})$, which matches the evaluation of a multiplication gate in the modified circuit $\bar{\mathcal{C}}_f$ (cf. Algorithm $\mathsf{OffVerify}$, Section 4.1). Hence, if we assume that $\Lambda_i^{(1)}$ and $\Lambda_j^{(2)}$ satisfy Equation 2, then Λ_{i+j} will also satisfy that equation.

Correctness of Γ_{i+j}. Assume that $\Gamma_i^{(1)} = P_i{}^{\gamma_1}$, $\Gamma_j^{(2)} = P_j{}^{\gamma_2}$, $\Lambda_i^{(1)} = P_i{}^{\lambda_1}$, and $\Lambda_j^{(2)} = P_j{}^{\lambda_2}$, for $\gamma_1, \gamma_2, \in \mathbb{F}_p$ and $\lambda_1, \lambda_2 \in \mathbb{F}_p^*$.

If we assume that $\Upsilon^{(l)}$, $l \in \{1, 2\}$, verifies Equation 3, then λ_l is an eigenvalue of matrix M_l associated with eigenvector $\boldsymbol{u} = (x, y)$, where

$$M_l = \begin{bmatrix} m^{(l)} & \gamma_l \\ 0 & \lambda_l \end{bmatrix}$$

Algorithm 2: The weak unforgeability experiment of homomorphic signatures

$\mathsf{param} \leftarrow \mathcal{O}_{\mathsf{KeyGen}}(1^\kappa, \mathcal{L})$;
$\mathcal{A}$ picks up to s dataset identifiers
$\mathcal{A} \rightarrow \Delta_i$;
For each dataset Δ_i, $\mathcal{A}$ generates t signature queries
$\mathcal{A} \rightarrow (\tau_{(i,j)}, m^{(i,j)})$; # if $l \neq j$, then $\tau_{(i,j)} \neq \tau_{(i,l)}$
$\mathcal{O}_{\mathsf{KeyGen}}$ returns the verification key
$\mathsf{VK} \leftarrow \mathcal{O}_{\mathsf{KeyGen}}(1^\kappa, \mathcal{L})$;
$\mathcal{O}_{\mathsf{Sign}}$ generates the signatures for $\mathcal{A}$'s queries
$\sigma^{(i,j)} \leftarrow \mathcal{O}_{\mathsf{Sign}}(\mathsf{VK}, \Delta_i, \tau_{(i,j)}, m^{(i,j)})$;
$\mathcal{A}$ outputs the tuple on which is going to be challenged
$\mathcal{A} \rightarrow (\mathcal{P}_\Delta, m, \sigma)$; # $\mathcal{P}_\Delta = (\mathcal{P}, \Delta)$
$b \leftarrow \mathsf{Verify}(\mathsf{VK}, \mathcal{P}_\Delta, m, \sigma)$;

and consequently, $\lambda_1\lambda_2$ is also an eigenvalue of matrix

$$M = M_1 M_2 = \begin{bmatrix} m^{(1)}m^{(2)} & m^{(1)}\gamma_2 + \gamma_1\lambda_2 \\ 0 & \lambda_1\lambda_2 \end{bmatrix}$$

and it is associated with vector $\boldsymbol{u} = (x, y)$. This implies that $M\boldsymbol{u} = \lambda_1\lambda_2\boldsymbol{u}$. Namely, we have $x(m^{(1)}m^{(2)}) + y(m^{(1)}\gamma_2 + \gamma_1\lambda_2) = x(\lambda_1\lambda_2)$, and Equality 3 ensues as a result.

4.3 Soundness

We first define our experiment of *weak unforgeability* in Algorithm 2.

The difference between this experiment and the experiment depicted in Algorithm 1 is that adversary $\mathcal{A}$ receives the verification key VK after submitting its signature queries $\langle \Delta_i, \tau_{(i,j)}, m^{(i,j)} \rangle$.

We say that adversary $\mathcal{A}$ breaks the weak unforgeability experiment if it provides a successful forgery of either **Type I**, **Type II**, or **Type III** (cf. Section 2.2).

Similarly to Definition 5, we say that a homomorphic scheme is (s, t)-weakly sound **iff**, the probability that adversary $\mathcal{A}$ succeeds in breaking the weak unforgeability experiment is negligible.

Theorem 3. *If the digital signature Σ_{Dig} and the aggregate signature Σ_{Agg} are secure, then the homomorphic signature introduced in Section 4.1 is (s, t)-weakly sound under the MDHI assumption.*

Proof. Assume there is an adversary $\mathcal{A}$ that breaks the (s, t)-weak soundness of our homomorphic signature with a non-negligible advantage $\epsilon_{\mathcal{A}}$. In the following, we demonstrate that given the security of digital signature Σ_{Dig} and aggregate signature Σ_{Agg}, there exists another adversary $\mathcal{B}$ that breaks the MDHI assumption, with a non-negligible advantage $\epsilon_{\mathcal{A}}/s$.

Indeed, to break the soundness of our scheme, adversary $\mathcal{A}$ might output either a forgery of **Type I**, a forgery of **Type II** or finally a forgery of **Type III**.

If digital signature Σ_{Dig} is secure, then it is infeasible for adversary $\mathcal{A}$ to successfully provide a valid **Type I** forgery. Similarly, the security of aggregate signature Σ_{Agg} assures that it is infeasible for adversary $\mathcal{A}$ to output a valid **Type III** forgery. These two facts entail that $\mathcal{A}$ breaks the (s,t)-weak soundness of our signature by providing a **Type II** forgery.

Below, we show how adversary $\mathcal{B}$ exploits the **Type II** forgery returned by adversary $\mathcal{A}$ to break the MDHI assumption:

First $\mathcal{B}$ queries the oracle $\mathcal{O}_{\mathsf{MDHI}}$ with security parameter 1^{κ} and a degree $d+1$ of multilinear map e. Upon query, oracle $\mathcal{O}_{\mathsf{MDHI}}$ returns the description of $d+1$ multilinear groups $\mathbb{G}_1, ..., \mathbb{G}_{d+1}$ of prime order p, a leveled multilinear map $e : \mathbb{G}_i \times \mathbb{G}_j \rightarrow \mathbb{G}_{i+j}$, $i, j \geq 1$ and $i + j \leq d + 1$, and a pair $(P_1, {P_1}^{\alpha}) \in \mathbb{G}_1^2$ for some randomly chosen $\alpha \in \mathbb{F}_p^*$.

We recall that the goal of $\mathcal{B}$ is to output $(P_{d+1})^{\alpha^{-1}} \in \mathbb{G}_{d+1}$. To this effect, $\mathcal{B}$ simulates the weak unforgeability experiment depicted in Algorithm 2 as shown below:

– $\mathcal{B}$ simulates oracle $\mathcal{O}_{\mathsf{KeyGen}}$ and publishes the parameters

$$\mathsf{param} = (p, \{\tau_i\}_{i=1}^{N}, H, e, \hat{e}, \{\mathbb{G}_i\}_{i=1}^{d+1}, \mathbb{G}, \mathbb{G}_T, P_1, P)$$

– $\mathcal{A}$ submits st signature queries $\langle \Delta_i, \tau_{(i,j)}, m^{(i,j)} \rangle$, $1 \leq i \leq s, 1 \leq j \leq t$, such that $\Delta_i \in \{0,1\}^*$, $\tau_{(i,j)} \in \{\tau_1, ..., \tau_N\}$, $\tau_{(i,j)} \neq \tau_{(i,l)}$ for $l \neq j$, and $m^{(i,j)} \in \mathbb{F}_p$.

– After receiving the signature queries $\langle \Delta_i, \tau_{(i,j)}, m^{(i,j)} \rangle$, $\mathcal{B}$ computes secret key SK and verification key VK as follows:

- It selects a pair of secret and public keys $(\mathsf{SK}_{\mathsf{Dig}}, \mathsf{PK}_{\mathsf{Dig}})$ for digital signature Σ_{Dig}. Then it chooses a keyed hash function F and a secret key K.
- It randomly picks a dataset identifier Δ from $\{\Delta_i\}_{i=1}^{s}$. Without loss of generality, we denote $\langle \Delta, \tau_j, m^{(j)} \rangle_{j=1}^{t}$ the signature queries corresponding to dataset identifier Δ. Next $\mathcal{B}$ generates $\gamma_j \in \mathbb{F}_p^*$ as $F(K, \Delta, \tau_j)$, $1 \leq j \leq t$, and defines the generators $\Lambda_1^{(\tau_j)}$ as:

$$\Lambda_1^{(\tau_j)} = {P_1}^{(m^{(j)})} ({P_1}^{\alpha})^{\gamma_j} \text{ [4]} \tag{4}$$

  Later, it picks $N - t$ random generators $\Lambda_1^{(\tau_j)} \in \mathbb{G}_1$, $t + 1 \leq j \leq N$.
- Finally, it defines SK and VK as:

$$\mathsf{SK} = (F, K, \mathsf{SK}_{\mathsf{Dig}})$$
$$\mathsf{VK} = (\mathsf{PK}_{\mathsf{Dig}}, \{(\tau_i, \Lambda_1^{(\tau_i)})\}_{i=1}^{N})$$

– After determining the pair $(\mathsf{SK}, \mathsf{VK})$, $\mathcal{B}$ returns verification key VK to $\mathcal{A}$.

[4] Note that the probability that $\Lambda_1^{(\tau_j)} = 1$ is equal to $\frac{1}{p}$, which is negligible. Still, in the unlikely event of $\Lambda_1^{(\tau_j)} = 1$, adversary $\mathcal{B}$ regenerates γ_j.

– Afterwards, on a query $\langle \Delta_i, \tau_{(i,j)}, m^{(i,j)} \rangle$, $\mathcal{B}$ simulates the responses of oracle $\mathcal{O}_{\mathsf{Sign}}$ as follows:
  If $\Delta_i \neq \Delta$, then $\mathcal{B}$ follows the signing algorithm Sign depicted in Section 4.1.
  If $\Delta_i = \Delta$, then $\langle \Delta_i, \tau_{(i,j)}, m^{(i,j)} \rangle = \langle \Delta, \tau_j, m^{(j)} \rangle$ and $\mathcal{B}$ acts as following:
  - It generates the pair of keys $(\mathsf{SK}_{\mathsf{Agg}}, \mathsf{PK}_{\mathsf{Agg}})$ as explained in algorithm Sign.
  - It computes digital signature Ω of $\mathsf{param}_{\Delta} = (\Delta, \mathsf{PK}_{\mathsf{Agg}}, \mathsf{PK}_{\mathsf{Dig}})$, and aggregate signature $\Psi^{(j)}$ of the generator $\Lambda_1^{(\tau_j)}$ associated with label τ_j.
  - It generates $x = F(K, \Delta, 1)$ in $\mathbb{F}_p^*$, computes $X_1 = {P_1}^{x_1}$ and $Y_1 = ({P_1}^{\alpha})^x$, and lets $\mathsf{PK}_{\mathsf{Hom}} = (X_1, Y_1)$.
  - It sets homomorphic signature Υ to $(m^{(j)}, \Lambda_1^{(\tau_j)}, \Gamma_1^{(j)})$, where $\Gamma_1^{(j)} = {P_1}^{\gamma_j}$ and $\gamma_j = F(K, \Delta, \tau_j)$.
    Note that by construction $\Lambda_1^{(\tau_j)} = (P_1)^{m^{(j)}} (\Gamma_1^{(j)})^{\alpha}$ (cf. Equation 4), therefore Υ is a valid signature as it verifies equation 3 for $(X_1, Y_1) = ({P_1}^{x}, {P_1}^{\alpha x})$.
  - Finally, $\mathcal{B}$ returns homomorphic signature $\sigma^{(j)} = (\mathsf{param}_{\Delta}, \Omega, \Psi^{(j)}, \Upsilon^{(j)})$.

At the end of the weak unforgeability experiment, $\mathcal{A}$ outputs a multi-labeled program $(\mathcal{P}, \tilde{\Delta})$, and a **Type II** forgery σ. Without loss of generality, we assume that $\mathcal{P} = (f, \tau_1, ..., \tau_t)$ and that f is a t-variate arithmetic circuit of degree $k \leq d$.

On receiving the description of $(\mathcal{P}, \tilde{\Delta})$, $\mathcal{B}$ checks whether $\tilde{\Delta} = \Delta$. If not, $\mathcal{B}$ aborts the experiment.

If $\tilde{\Delta} = \Delta$ and $\mathcal{A}$ succeeds in breaking the weak unforgeability experiment, then $\sigma = (\mathsf{param}_{\Delta}, \Omega, \Psi, \Upsilon)$ such that:

$$\Upsilon = (m, \Lambda_k, \Gamma_k) \in \mathbb{F}_p \times \mathbb{G}_k^2$$
$$\Lambda_k = f(\Lambda_1^{(\tau_1)}, ..., \Lambda_1^{(\tau_t)})$$

$$e(X_1, \Lambda_k) = e(X_1, P_k)^m e(Y_1, \Gamma_k) \tag{5}$$

Hence, to break the MDHI assumption, $\mathcal{B}$ first runs algorithm Eval on input of $(f, \boldsymbol{\sigma})$, where $\boldsymbol{\sigma} = (\sigma^{(1)}, ..., \sigma^{(t)})$ and for all $1 \leq j \leq t$, $\sigma^{(j)}$ is the homomorphic signature of message $m^{(j)}$ belonging to dataset Δ under label τ_j. In turn, algorithm Eval outputs a tuple $(\mathsf{param}_{\Delta}, \Omega, \Psi, \ddot{\Upsilon})$, such that $\ddot{\Upsilon} = (\ddot{m}, \Lambda_k, \ddot{\Gamma}_k) \in \mathbb{F}_p \times \mathbb{G}_k^2$.

Since (m, σ) is a forgery of **Type II**, then $m \neq \ddot{m}$ and $\mathcal{B}$ breaks the MDHI assumption by outputting:

$$(P_{d+1})^{\alpha^{-1}} = e\left(\frac{\Gamma_k}{\ddot{\Gamma}_k}, P_{d+1-k}\right)^{\frac{1}{\ddot{m}-m}}$$

Notice that by definition:

$$e(X_1, \Lambda_k) = e(X_1, P_k)^{\ddot{m}} e(Y_1, \ddot{\Gamma}_k) \tag{6}$$

From Equation 5 and Equation 6, we deduce the following:

$$e(X_1, P_k)^{\ddot{m}} e(Y_1, \ddot{\Gamma}_k) = e(X_1, P_k)^m e(Y_1, \Gamma_k) = e(X_1, \Lambda_k)$$
$$e(X_1, P_k)^{\ddot{m}-m} = e\left(Y_1, \frac{\Gamma_k}{\ddot{\Gamma}_k}\right)$$

$$e({P_1}^x, P_k)^{\dddot{m}-m} = e\left({P_1}^{\alpha x}, \frac{\Gamma_k}{\ddot{\Gamma}_k}\right)$$
$$e({P_1}^x, P_k) = e\left({P_1}^{\alpha x}, \frac{\Gamma_k}{\ddot{\Gamma}_k}\right)^{\frac{1}{\dddot{m}-m}}$$

Since $x \in \mathbb{F}_p^*$, $e(P_1, P_k) = e\left({P_1}^{\alpha}, \left(\frac{\Gamma_k}{\ddot{\Gamma}_k}\right)^{\frac{1}{\dddot{m}-m}}\right)$, which means that $\left(\frac{\Gamma_k}{\ddot{\Gamma}_k}\right)^{\frac{1}{\dddot{m}-m}} = {P_k}^{\alpha^{-1}}$.

We thus conclude that if adversary $\mathcal{A}$ breaks the weak unforgeability experiment with a non-negligible advantage $\epsilon_{\mathcal{A}}$, then adversary $\mathcal{B}$ breaks the MDHI assumption as long as it does not stop the unforgeability experiment.

Here we quantify the advantage $\epsilon_{\mathcal{B}}$ of adversary B.

Let $E_{\mathcal{B}}$ be the event that adversary $\mathcal{B}$ succeeds in breaking the MDHI assumption, and let E be the event that adversary $\mathcal{B}$ does not stop the unforgeability experiment.

We know that $\Pr(E) = \frac{1}{s}$ and $\Pr(E_{\mathcal{B}} \mid E) = \epsilon_{\mathcal{A}}$, and also that:

$$\begin{aligned}\epsilon_{\mathcal{B}} &= \Pr(E_{\mathcal{B}}) = \Pr(E_{\mathcal{B}} \cap E) + \Pr(E_{\mathcal{B}} \cap \overline{E}) \\ &= \Pr(E_{\mathcal{B}} \mid E)\Pr(E) + \Pr(E_{\mathcal{B}} \mid \overline{E})\Pr(\overline{E}) \\ &= \frac{\epsilon_{\mathcal{A}}}{s} + \Pr(E_{\mathcal{B}} \mid \overline{E})\Pr(\overline{E}) \geq \frac{\epsilon_{\mathcal{A}}}{s}\end{aligned}$$

5 An Adaptively Secure Homomorphic Signature

We hereby transform the solution described in Section 4.1 into an adaptively secure signature. This transformation is performed in three steps: (i) generate a one-degree polynomial t whose free coefficient is m, i.e. the original message to be signed; (ii) evaluate polynomial t at a secret point α; (iii) and finally, sign $t(\alpha)$ using the weakly secure signature. In Section 5.4, we show that this simple transformation yields an adaptively secure fully homomorphic signature under the MDHI assumption.

5.1 Description

$\mathsf{KeyGen}^*(1^\kappa, d, \mathcal{L}) \rightarrow (\mathsf{SK}^*, \mathsf{VK}^*, \mathsf{param})$ Given a security parameter 1^κ, an upper-bound d of the degree of circuits supported by the signature, and the set of labels $\mathcal{L} = \{\tau_1, ..., \tau_N\} \subset \{0,1\}^*$, algorithm KeyGen^* first runs KeyGen which in turn yields a tuple $(\mathsf{SK}, \mathsf{VK}, \mathsf{param})$. Algorithm KeyGen^* additionally selects a secret key $\alpha \in \mathbb{F}_p^*$, computes $A_1 = P_1^\alpha$ and outputs the following:

$$\begin{aligned}\mathsf{SK}^* &= (\mathsf{SK}, \alpha) = (F, K, \mathsf{SK}_{\mathsf{Dig}}, \alpha) \\ \mathsf{VK}^* &= (\mathsf{VK}, A_1) = (\mathsf{PK}_{\mathsf{Dig}}, \{(\tau_i, \Lambda_1^{(\tau_i)})\}_{i=1}^N, A_1) \\ \mathsf{param} &= (p, \{\tau_i\}_{i=1}^N, H, e, \hat{e}, \{\mathbb{G}_i\}_{i=1}^{d+1}, \mathbb{G}, \mathbb{G}_T, P_1, P)\end{aligned}$$

$\mathsf{Sign}^*(\mathsf{SK}^*, \Delta, \tau, m) \rightarrow \sigma^*$ On input of signing key $\mathsf{SK}^* = (\mathsf{SK}, \alpha)$, data set identifier Δ, a label $\tau \in \mathcal{L}$, and a message $m \in \mathbb{F}_p$, algorithm Sign^* generates a random

number $\theta_1 \in \mathbb{F}_p^*$, and computes the polynomial $t(z) = m + \theta_1 z$. Using secret key SK, this algorithm executes algorithm Sign over the tuple $(\Delta, \tau, t(\alpha))$, (i.e. algorithm Sign signs message $t(\alpha)$). This results in a signature $\sigma = (\mathsf{param}_\Delta, \Omega, \Psi, \Upsilon)$ with $\Upsilon = (t(\alpha), \Lambda_1, \Gamma_1)$. Thereafter, algorithm Sign^* defines the homomorphic signature Υ^* of tuple (m, τ, Δ) as:

$$\Upsilon^* = ([m, \boldsymbol{\theta}], \Lambda_1, \Gamma_1)$$

whereby $[m, \boldsymbol{\theta}] = [m, \theta_1, .., \theta_d]$ represents an at most d-degree polynomial $t(z) = m + \sum_{r=1}^{d} \theta_r z^r$ (i.e. the free coefficient is m). In the current setting, the degree of t is 1 and hence: $\Upsilon^* = ([m, \theta_1, 0, .., 0], \Lambda_1, \Gamma_1)$.
At the end, Sign^* returns the fully homomorphic signature:

$$\sigma^* = (\mathsf{param}_\Delta, \Omega, \Psi, \Upsilon^*)$$

$\mathsf{Eval}^*(\mathsf{VK}^*, f, \boldsymbol{\sigma^*}) \rightarrow \sigma^*$ When provided with an n-variate function f and a vector $\boldsymbol{\sigma}^*$ of n homomorphic signatures $\sigma^{*(l)} = (\mathsf{param}_\Delta, \Omega, \Psi^{(l)}, \Upsilon^{*(l)})$, such that each signature $\sigma^{*(l)}$ authenticates a messages $m^{(l)}$ for $1 \leq l \leq n$, algorithm Eval^* proceeds – similarly to Eval – in two steps:
Computation of aggregate signature Ψ. This aggregate signature is obtained by evaluating the same circuit $\tilde{\mathcal{C}}_f$ defined in algorithm Eval, on inputs of aggregate signatures $(\Psi^{(1)}, ..., \Psi^{(n)})$ (see Section 4.1).
Computation of homomorphic signature Υ^*. In this step, algorithm Eval^* evaluates the circuit $\mathcal{C}_f^*$ of function f on inputs $(\Upsilon^{*(1)}, ..., \Upsilon^{*(n)})$ using the following subroutines:

- $\mathsf{GateEval}_+^*(\Upsilon^{*(1)}, \Upsilon^{*(2)})$: Here we assume that for $l \in \{1, 2\}$, $\Upsilon^{*(l)} = ([m^{(l)}, \boldsymbol{\theta}^{(l)}], \Lambda_i^{(1)}, \Gamma_i^{(1)}) \in \mathbb{F}_p^{d+1} \times \mathbb{G}_i^2$. Whenever, an addition gate is encountered, $\mathsf{GateEval}_+^*$ outputs $\Upsilon^* = ([m, \boldsymbol{\theta}], \Lambda_i, \Gamma_i)$ such that:

$$[m, \boldsymbol{\theta}] = [m^{(1)}, \boldsymbol{\theta}^{(1)}] + [m^{(2)}, \boldsymbol{\theta}^{(2)}]$$

$$\Lambda_i = \Lambda_i^{(1)} \Lambda_i^{(2)} \; ; \; \Gamma_i = \Gamma_i^{(1)} \Gamma_i^{(2)}$$

- $\mathsf{GateEval}_c^*(\Upsilon^{*(1)}, c)$: This subroutine is called in order to evaluate a gate for multiplication by a constant. Accordingly, on inputs of a homomorphic signature $\Upsilon^{*(1)} = ([m^{(1)}, \boldsymbol{\theta}^{(1)}], \Lambda_i^{(1)}, \Gamma_i^{(1)}) \in \mathbb{F}_p^{d+1} \times \mathbb{G}_i^2$ and a constant c, $\mathsf{GateEval}_c$ outputs $\Upsilon^* = ([m, \boldsymbol{\theta}], \Lambda_i, \Gamma_i)$ where:

$$[m, \boldsymbol{\theta}] = c[m^{(1)}, \boldsymbol{\theta}^{(1)}]$$

$$\Lambda_i = (\Lambda_i^{(1)})^c \; ; \; \Gamma_i = (\Gamma_i^{(1)})^c$$

- $\mathsf{GateEval}_\times^*(\Upsilon^{*(1)}, \Upsilon^{*(2)})$: Without loss of generality, assume that $\Upsilon^{*(1)} = ([m^{(1)}, \boldsymbol{\theta}^{(1)}], \Lambda_i^{(1)}, \Gamma_i^{(1)}) \in \mathbb{F}_p^{d+1} \times \mathbb{G}_i^2$, and $\Upsilon^{*(2)} = ([m^{(2)}, \boldsymbol{\theta}^{(2)}], \Lambda_j^{(2)}, \Gamma_j^{(2)}) \in \mathbb{F}_p^{d+1} \times \mathbb{G}_j^2$. This means that (i) $\Upsilon^{*(1)}$ and $\Upsilon^{*(2)}$ are inputs of degree i and j respectively; and that (ii) $\boldsymbol{\theta}^{(1)} = [\theta_1^{(1)}, ...\theta_i^{(1)}, 0, ...0]$ and $\boldsymbol{\theta}^{(2)} = [\theta_1^{(2)}, ...\theta_j^{(2)}, 0, ...0]$.

Now whenever a message $m^{(1)}$ is going to be multiplied with a message $m^{(2)}$, $\mathsf{GateEval}^*_\times$ generates an $(i+j)$-degree polynomial t by multiplying the two polynomials $t^{(1)}(z) = m^{(1)} + \sum_{r=1}^{i} \theta_r^{(1)} z^r$ and $t^{(2)}(z) = m^{(2)} + \sum_{r=1}^{j} \theta_r^{(2)} z^r$. The coefficients of the resulting polynomial t are denoted by $[m^{(1)} m^{(2)}, \boldsymbol{\theta}] \in \mathbb{F}_p^{d+1}$, where $\boldsymbol{\theta} = [\theta_1, ..., \theta_{i+j}, 0, ..., 0]$.
Additionally, $\mathsf{GateEval}^*_\times$ recursively computes the parameters A_{r+1} and $B_{i,r}$, for all $1 \leq r \leq i-1$, as follows:

$$A_{r+1} = e(A_r, A_1)\ ;\ B_{i,r} = e(P_{i-r}, A_r)$$

Given these parameters, $\mathsf{GateEval}^*_\times$ computes:

$$\Lambda_{i+j} = e(\Lambda_i^{(1)}, \Lambda_j^{(2)})$$
$$\Gamma_{i+j} = e(P_i^{m^{(1)}} A_i^{\theta_i^{(1)}} \prod_{r=1}^{i-1} B_{i,r}^{\theta_r^{(1)}}, \Gamma_j^{(2)}) e(\Gamma_i^{(1)}, \Lambda_j^{(2)})$$

$\mathsf{GateEval}^*_\times$ then outputs $\Upsilon^* = ([m, \boldsymbol{\theta}], \Lambda_{i+j}, \Gamma_{i+j})$.
Algorithm Eval^* concludes its execution by outputting:

$$\sigma^* = (\mathsf{param}_\Delta, \Omega, \Psi, \Upsilon^*)$$

$\mathsf{OffVerify}^*(\mathsf{VK}^*, \mathcal{P}) \rightarrow \mathsf{VK}^*_\mathcal{P}$ We assume here that $\mathcal{P}$ evaluates an n-variate function f whose circuit is of degree k. Hence, given $\mathsf{VK}^* = (\mathsf{VK}, A_1)$, algorithm $\mathsf{OffVerify}^*$ first computes $\mathsf{VK}_\mathcal{P}$ using $\mathsf{OffVerify}$ (i.e. $\mathsf{VK}_\mathcal{P} \leftarrow \mathsf{OffVerify}(\mathsf{VK}, \mathcal{P})$), and generates the parameters:

$$A_{r+1} = e(A_r, A_1)\ ;\ B_{k,r} = e(P_{k-r}, A_r);\ \ 1 \leq r \leq k-1$$

Finally, algorithm $\mathsf{OffVerify}^*$ outputs:

$$\mathsf{VK}^*_\mathcal{P} = (\mathsf{VK}_\mathcal{P}, A_k, \{B_{k,1}, ..., B_{k,k-1}\})$$

where $\mathsf{VK}_\mathcal{P} = (\mathsf{PK}_{\mathsf{Dig}}, H_\mathcal{P}, f(\Lambda_1^{(\tau_1)}, ..., \Lambda_1^{(\tau_n)}))$.

$\mathsf{OnVerify}^*(\mathsf{VK}^*_\mathcal{P}, \Delta, m, \sigma^*) \rightarrow b$ On input of concise verification key $\mathsf{VK}^*_\mathcal{P}$, dataset identifier Δ, message m and signature $\sigma^* = (\mathsf{param}_\Delta, \Omega, \Psi, \Upsilon^*)$, algorithm $\mathsf{OnVerify}^*$ proceeds as following:

- It parses $\mathsf{VK}^*_\mathcal{P}$ as $(\mathsf{VK}_\mathcal{P}, A_k, \{B_{k,1}, ..., B_{k,k-1}\})$, dataset parameters param_Δ as $(\Delta, \mathsf{PK}_{\mathsf{Agg}}, \mathsf{PK}_{\mathsf{Hom}})$ and signature σ^* as $([m, \boldsymbol{\theta}], \Omega, \Psi, \Upsilon^*)$.
- Similarly to $\mathsf{OnVerify}$, it uses $\mathsf{VK}_\mathcal{P}$ to incrementally check the validity of digital signature Ω and aggregate signature Ψ. If any of these signatures is not valid, then $\mathsf{OnVerify}^*$ returns $b = 0$.
- Finally, it parses homomorphic signature Υ^* as a tuple $([m, \boldsymbol{\theta}], \Lambda_k, \Gamma_k) \in \mathbb{F}_p^{d+1} \times \mathbb{G}_k^2$, and given verification key $\mathsf{VK}_\mathcal{P}$, public key $\mathsf{PK}_{\mathsf{Hom}} = (X_1, Y_1) \in \mathbb{G}_1^2$ and $(A_k, \{B_{k,1}, ..., B_{k,k-1}\}) \in \mathbb{G}_k^k$ verifies whether the following equalities hold:

$$\Lambda_k = f(\Lambda_1^{(\tau_1)}, ..., \Lambda_1^{(\tau_n)}) \tag{7}$$

$$e(X_1, \Lambda_k) = e(X_1, P_k^m A_k^{\theta_k} \prod_{r=1}^{k-1} B_{k,r}^{\theta_r}) e(Y_1, \Gamma_k) \tag{8}$$

If both equalities hold, then $\mathsf{OnVerify}^*$ outputs $b = 1$; otherwise it outputs $b = 0$.

5.2 Efficiency

In the following, we briefly discuss the efficiency of our homomorphic signature.

Online-Offline Signing. We point out that the signature we described in Section 5 supports online-offline signatures. Similarly to previous work on online-offline signatures [19], the signer could sign a random dataset $\ddot{\Delta} = (\ddot{m}^{(1)}, ..., \ddot{m}^{(n)})$ offline, using the weakly secure signature. This yields a tuple $\mathsf{param}_{\ddot{\Delta}} = (\ddot{\Delta}, \ddot{\mathsf{PK}}_{\mathsf{Agg}}, \ddot{\mathsf{PK}}_{\mathsf{Hom}})$, a digital signature $\ddot{\Omega}$, n aggregate signatures $\ddot{\Psi}^{(i)}$ and n homomorphic signatures $\ddot{\Upsilon}^{(i)} = (\ddot{m}^{(i)}, \ddot{\Lambda}_1^{(i)}, \ddot{\Gamma}_1^{(i)})$. Later when a dataset $\Delta = (m^{(1)}, ..., m^{(n)})$ is generated, the signer proceeds as following:

- The signer uses secret key α to find for each message $m^{(i)}$ the coefficient $\theta_1^{(i)}$ verifying: $\ddot{m}^{(i)} = \alpha\theta_1^{(i)} + m^{(i)}$, and sets the homomorphic signature $\Upsilon^{(i)}$ to $([m^{(i)}, \theta_1^{(i)}], \ddot{\Lambda}_1^{(i)}, \ddot{\Gamma}_1^{(i)})$;
- she defines signature $\Psi^{(i)}$ of message $m^{(i)}$ as $\ddot{\Psi}^{(i)}$;
- she defines the public parameters of dataset Δ as $\mathsf{param}_{\Delta} = (\Delta, \ddot{\mathsf{PK}}_{\mathsf{Agg}}, \ddot{\mathsf{PK}}_{\mathsf{Hom}})$ and signs these public parameters using her public key PK.

Adaptive Security Transformation. Catalano et al. [11] and Gorbunov et al. [18] independently proposed generic transformations that make any weakly secure homomorphic signature into an adaptively secure one. While it is possible to leverage any of these transformations to define our adaptively secure signature, it was more practical and simple to opt for a dedicated transformation. For instance, compared to what is proposed in [11], our transformation yields shorter signatures: The evaluation of a circuit $\mathcal{C}_f$ of degree k results in a signature of size $\simeq k$ when using our transformation, rather than a signature of size $\simeq 3k$ in the case of the transformation in [11]. Additionally, we note that if we use any of these transformations, then we lose the online-offline signing feature.

5.3 Correctness

Theorem 4. *The adaptively secure homomorphic signature described above ensures authentication and evaluation correctness.*

Proof (Sketch). Due to the close similarity with the proofs of correctness of the weakly secure homomorphic signature, we only provide a proof sketch for evaluation correctness. A similar argument can be used to prove authentication correctness.

Let the inputs of algorithm Verify be: $\mathcal{P}_{\Delta} = (\Delta, \mathcal{P})$ where $\mathcal{P} = (f, \tau_1, .., \tau_n)$ is a program evaluating a circuit $\mathcal{C}_f$ of degree k, $m = f(m^{(1)}, ..., m^{(n)})$ and $\sigma^* = (\mathsf{param}_{\Delta}, \Omega, \Psi, \Upsilon^*)$ such that $\Upsilon^* = ([m, \boldsymbol{\theta}], \Lambda_k, \Gamma_k)$.

Also, let t denote the polynomial associated with vector $[m, \boldsymbol{\theta}]$.

Finally, let $[m^{(l)}, \boldsymbol{\theta}^{(l)}]$ be the vector from signature $\sigma^{*(l)}$ of message $m^{(l)}$, and $t^{(l)}(z) = m^{(l)} + \sum_{r=1}^{i} \theta_r^{(l)} z^r$ be the corresponding polynomial (i.e. $\boldsymbol{\theta}^{(l)} = [\theta_1^{(l)}, ..., \theta_i^{(l)}, 0, ..., 0]$).

To show evaluation correctness, we rely on two observations: (i) The first is that $f(t^{(1)}(\alpha), ..., t^{(n)}(\alpha)) = f(t^{(1)}, ..., t^{(n)})(\alpha)$, which means that $f(t^{(1)}(\alpha), ..., t^{(n)}(\alpha)) = m + \sum_{r=1}^{k} \theta_r \alpha^r = t(\alpha)$. (ii) The second is that $(t(\alpha), \Lambda_k, \Gamma_k)$ is a valid weakly secure signature. More precisely, it can be viewed as the output of algorithm Eval (from Section 4.1) on inputs of $(t^{(l)}(\alpha), \Lambda_i^{(l)}, \Gamma_i^{(l)})$.

Thus by correctness of our weakly homomorphic signature, Equation 3 is verified for $m = t(\alpha)$. From this we can see that Equation 8 holds for vector $[m, \boldsymbol{\theta}]$. Indeed, we have for all $k \geq 2$ and for all $1 \leq r \leq k-1$: $A_k = P_k^{\alpha^k}$ and $B_{k,r} = P_k^{\alpha^r}$. Hence, Equation 8 could be rewritten as $e(X_1, \Lambda_k) = e(X_1, P_k^{t(\alpha)}) e(Y_1, \Gamma_k)$, which corresponds to Equation 3 when m is replaced by $t(\alpha)$.

5.4 Soundness

Theorem 5. *If the digital signature Σ_{Dig} and the aggregate signature Σ_{Agg} are secure, then our homomorphic signature is (s,t)-sound under the MDHI assumption.*

Proof (Sketch). Assume there is an adversary $\mathcal{A}$ that breaks the (s,t)-soundness of our homomorphic signature with a non-negligible advantage $\epsilon_{\mathcal{A}}$. Similarly to the proof of Theorem 3, since Σ_{Dig} and Σ_{Agg} are secure, the only way $\mathcal{A}$ can succeed in breaking the (s,t)-soundness is through a **Type II** forgery. Given the existence of a **Type II** forgery by $\mathcal{A}$, we show that either there exists an adversary $\mathcal{B}$ that breaks our weakly secure homomorphic signature, or that adversary $\mathcal{A}$ breaks the MDHI assumption.

To break our weakly secure homomorphic signature, adversary $\mathcal{B}$ simulates the unforgeability experiment as follows:

- First, $\mathcal{B}$ enters the learning phase of Algorithm 2 and receives public parameters param whereby

$$\mathsf{param} = (p, \{\tau_i\}_{i=1}^{N}, H, e, \hat{e}, \{\mathbb{G}_i\}_{i=1}^{d+1}, \mathbb{G}, \mathbb{G}_T, P_1, P)$$

  Further, it submits sN signature queries $\langle \Delta_i, \tau^{(i,j)}, m^{(i,j)} \rangle$ for randomly generated messages $m^{(i,j)}$, $1 \leq i \leq s$ and $1 \leq j \leq N$, so that to receive $\mathsf{VK} = (\mathsf{PK}_{\mathsf{Dig}}, \{(\tau_i, \Lambda_1^{(\tau_i)})\}_{i=1}^{N})$ and signatures $\sigma^{(i,j)}$ such that:

$$\begin{aligned} \sigma^{(i,j)} &= (\mathsf{param}_{\Delta_i}, \Omega^{(i)}, \Psi^{(i,j)}, \Upsilon^{(i,j)}) \\ \mathsf{param}_{\Delta_i} &= (\Delta_i, \mathsf{PK}_{\mathsf{Agg}}^{(i)}, \mathsf{PK}_{\mathsf{Hom}}^{(i)}) \\ \Upsilon^{(i,j)} &= (m^{(i,j)}, \Lambda_1^{(i,j)}, \Gamma_1^{(i,j)}) \end{aligned}$$

  $\mathcal{B}$ keeps these signatures in a table T to further use them during its simulation.
- When $\mathcal{A}$ queries oracle $\mathcal{O}_{\mathsf{KeyGen}^*}$, $\mathcal{B}$ generates $(\alpha, A_1 = P_1^{\alpha})$, sets $\mathsf{VK}^* = (\mathsf{PK}_{\mathsf{Dig}}^*, \{(\tau_i, \Lambda_1^{(\tau_i)})\}_{i=1}^{N}, A_1)$ where $\mathsf{PK}_{\mathsf{Dig}}^*$ is its own public key, and returns accordingly $(\mathsf{param}, \mathsf{VK}^*)$.

– When $\mathcal{A}$ queries for the signature of $\langle \Delta_i^*, \tau_{(i,j)}^*, m^{*(i,j)} \rangle$, $\mathcal{B}$ first fetches the public parameters $(\Delta_i, \mathsf{PK}_{\mathsf{Agg}}^{(i)}, \mathsf{PK}_{\mathsf{Hom}}^{(i)})$ of dataset Δ_i in table T, and signs $\mathsf{param}_{\Delta_i^*} = (\Delta_i^*, \mathsf{PK}_{\mathsf{Agg}}^{(i)}, \mathsf{PK}_{\mathsf{Hom}}^{(i)})$ using secret key $\mathsf{SK}_{\mathsf{Dig}}^*$ matching public key $\mathsf{PK}_{\mathsf{Dig}}^*$. This results in a signature $\Omega^{*(i)}$.
Further, $\mathcal{B}$ finds in table T the message in dataset Δ_i that was signed under label $\tau_{(i,j)}^*$, Here, we assume that this message corresponds to $m^{(i,j)}$ (i.e. $\tau_{(i,j)}^* = \tau_{(i,j)}$). $\mathcal{B}$ then retrieves signature $\sigma^{(i,j)} = (\mathsf{param}_{\Delta_i}, \Omega^{(i)}, \Psi^{(i,j)}, \Upsilon^{(i,j)})$, and lets $\Upsilon^{*(i,j)} = ([m^{*(i,j)}, \theta_1^{(i,j)}], \Lambda_1^{(i,j)}, \Gamma_1^{(i,j)})$, whereby $\theta_1^{(i,j)}$ is computed so that $m^{(i,j)} = m^{*(i,j)} + \theta_1^{(i,j)}\alpha$. Finally, $\mathcal{B}$ returns $\sigma^{*(i,j)} = (\mathsf{param}_{\Delta_i^*}, \Omega^{*(i)}, \Psi^{(i,j)}, \Upsilon^{*(i,j)})$.

Eventually, $\mathcal{A}$ outputs pairs $(\mathcal{P}, \Delta_i^*)$ and (m^*, σ^*).

We suppose here that $\mathcal{P} = (f, \tau_{(i,1)}, ..., \tau_{(i,t)})$, with f being a t-variate function of degree k.

Hence, $\sigma^* = (\mathsf{param}_{\Delta_i^*}, \Omega^{*(i)}, \Psi^{(i)}, \Upsilon^{*(i)})$ where $\Upsilon^{*(i)} = ([m^*, \boldsymbol{\theta}], \Lambda_k, \Gamma_k)$, and $\boldsymbol{\theta} = [\theta_1, ..., \theta_k, 0, ..., 0]$.

In order to break our weakly secure signature, $\mathcal{B}$ runs Eval^* on inputs of function f and signatures $\sigma^{*(i,j)}$ with $1 \leq j \leq t$.

This results in a vector $[\ddot{m}^*, \ddot{\boldsymbol{\theta}}] \in \mathbb{F}_p^{d+1}$, where $\ddot{m}^* = f(m^{*(i,1)}, ..., m^{*(i,t)})$ and $\ddot{\boldsymbol{\theta}} = [\ddot{\theta}_1, ..., \ddot{\theta}_k, 0, ..., 0]$.

Since $\mathcal{A}$'s output is a **Type II** forgery, we know that $m^* \neq \ddot{m}^*$. Now depending on whether $m^* + \sum_{r=1}^k \theta_r \alpha^r$ equals $\ddot{m}^* + \sum_{r=1}^k \ddot{\theta}_r \alpha^r$, we show that either $\mathcal{B}$ breaks the weak unforgeability of our scheme, or that $\mathcal{A}$ is able to break the MDHI assumption.

– If $m^* + \sum_{r=1}^k \theta_r \alpha^r \neq \ddot{m}^* + \sum_{r=1}^k \ddot{\theta}_r \alpha^r$, then adversary $\mathcal{B}$ breaks our weakly secure signature by outputting message $m = m^* + \sum_{r=1}^k \theta_r \alpha^r$, multi-labeled program $(\mathcal{P}, \Delta_i)$ and $\sigma = (\mathsf{param}_{\Delta_i}, \Omega^{(i)}, \Psi^{(i)}, \Upsilon^{(i)})$, whereby $\Upsilon^{(i)} = (m, \Lambda_k, \Gamma_k)$. Indeed, notice that:

$$\begin{aligned} \ddot{m}^* + \sum_{r=1}^{k} \ddot{\theta}_r \alpha^r &= f(m^{*(i,1)} + \alpha\theta_1^{(i,1)}, ..., m^{*(i,t)} + \alpha\theta_1^{(i,t)}) \\ &= f(m^{(i,1)}, ..., m^{(i,t)}) \end{aligned}$$

Thus, $m \neq f(m^{(i,1)}, ..., m^{(i,t)})$.

– If $m^* + \sum_{r=1}^k \theta_r \alpha^r = \ddot{m}^* + \sum_{r=1}^k \ddot{\theta}_r \alpha^r$, then we can show that $\mathcal{A}$ breaks the MDHI assumption. Namely, it can compute ${P_k}^{\alpha^{-1}}$ from $(P_1, A_1 = P_1^{\alpha})$. Indeed:

$$\begin{aligned} P_k^{m^* + \sum_{r=1}^k \theta_r \alpha^r} &= P_k^{\ddot{m}^* + \sum_{r=1}^k \ddot{\theta}_r \alpha^r} \\ P_k^{m^* - \ddot{m}^*} &= P_k^{\sum_{r=1}^k \ddot{\theta}_r \alpha^r - \sum_{r=1}^k \theta_r \alpha^r} \end{aligned}$$

For the sake of clarity, we simplify the formula by replacing the subtraction of the two polynomials with one polynomial:

$$P_k^{m^* - \ddot{m}^*} = P_k^{\sum_{r=1}^k \hat{\theta}_r \alpha^r}$$

$$P_k = P_k^{\frac{\sum_{r=1}^{k} \hat{\theta}_r \alpha^r}{m^* - \dot{m}^*}} = P_k^{\alpha \frac{\sum_{r=1}^{k} \hat{\theta}_r \alpha^{r-1}}{m^* - \dot{m}^*}}$$

This implies that $\mathcal{A}$ breaks the MDHI assumption by first computing $P_k^{\alpha^{r-1}}$, $1 \leq r \leq k$, using P_1, A_1 and bilinear pairing e, and then outputting:

$$P_k^{\alpha^{-1}} = \left(\prod_{r=1}^{k} P_k^{\hat{\theta}_r \alpha^{r-1}} \right)^{(m^* - \dot{m}^*)^{-1}}$$

Finally to conclude, since our weakly secure signature is sound under the MDHI assumption, so is our adaptively secure solution.

6 Related Work

Verifiable Delegation of Computation. The advent of cloud computing has spurred interest in verifiable delegation of computation [12, 15, 20, 21]. The main concern of this line of work is to ensure that the verification of computation calls for less computational resources than the delegated function. However, verifiable computation requires the verifier to have access to the dataset on which computation has been performed, which may be unrealistic in some settings.

Succinct Non-interactive ARguments of Knowledge. SNARKs [2, 3, 16] are a powerful tool to build fully homomorphic signatures. Generally speaking, SNARKs allow anyone to generate a proof for any NP statement. In particular, given a value y and a function f, one can employ SNARKs to prove that there exists a witness x that verifies $y = f(x)$. While SNARKs give way to efficient verification procedures, their soundness is only ensured under non-falsifiable assumptions [2, 17].

Additively Homomorphic Signatures. First attempts to design homomorphic signatures focused on authenticating linear functions, cf. [1, 4, 8]. The motivating applications for this type of homomorphic signatures are namely: *secure network coding* [8] which enables the authentication of messages forwarded in the network, and *proofs of retrievability* [1, 22] which provide means to efficiently verify the availability of data stored at untrusted servers.

Homomorphic MACs. Gennaro and Wichs [14] proposed one of the first homomorphic symmetric authenticators dedicated to Boolean circuits. The propounded solution builds upon homomorphic encryption and assumes that the adversary does not have access to the results of the MAC verification (i.e. an adversary cannot know whether a pair of message and signature is valid or not). To overcome this caveat, Catalano and Fiore [9] introduced a solution that leverages the algebraic properties of the ring of polynomials to homomorphically sign messages. Briefly, the idea of [9] is to represent the signature as a polynomial of degree 1 in which the free term corresponds to the signed message. In this manner, the proposed MAC is much more efficient than the work of Gennaro and Wichs [14] and is suitable for arithmetic circuits. The issue however with this solution is that program composition yields MACs whose size grows with the degree of the circuit. As a followup, Catalano et al. [10] exploited multilinear maps to devise a homomorphic MAC that supports program composition while conserving succinctness.

Homomorphic Signatures. One of the first solutions for homomorphic signatures was devised by Boneh and Freeman [5]. The proposed solution uses *ideal lattices* to authenticate the evaluation of multivariate polynomials. However, this scheme is shown to be secure in the random oracle model only. To address this shortcoming, Catalano et al. [11] build upon their previous work [9, 10] and design a homomorphic signature that is suitable for multi-variate polynomial functions and secure in the standard model. However, as discussed in Section 5.2, our solution outperforms the signature proposed in [11] in terms of both size and computation. In a more recent work, Gorbunov et al. [18] introduced leveled fully-homomorphic signatures from standard lattices, which contrary to our signature and the signatures in [5, 11], authenticate arbitrary functions. The proposed signature relies on dedicated homomorphic trapdoor functions and is shown to be adaptively secure in the standard model. Nevertheless, the signature in [18] does not take **Type III** forgeries into account.

Note that if similarly to Catalano et al. [11], we only consider multivariate polynomials of degree d verifying $d/p < 1/2$, then the signature presented in 4.1 becomes weakly-secure in the standard model. Indeed, according to Proposition 2 in [10], if $d/p < 1/2$, then **Type II** and **Type III** forgeries are equivalent. This entails that there is no need for aggregate signatures whose security is proved in the random oracle model. Still, we emphasize that using aggregate signatures to thwart **Type III** forgeries is of independent interest as it can be employed in other homomorphic signatures to resist **Type III** forgeries (for e.g. [18]).

Finally, we remark that to the best of our knowledge, our solution is the only solution that features an online-offline signing process. This capability is advantageous especially in contexts where mobile devices (smart-phones, tablets...etc.) are prevalent.

7 Conclusion

In this paper, we introduced a new construction for homomorphic signatures suitable for multivariate polynomials of bounded degree. By tailoring the algebraic properties of eigenvectors and leveled multilinear maps, the proposed construction allows efficient verification in the amortized model and enables online-offline signing. Besides, our solution yields signatures whose size grows only linearly in the degree of evaluated polynomials, and we show it to be provably secure under the MDHI assumption.

Bibliography

[1] G. Ateniese, R. C. Burns, R. Curtmola, J. Herring, L. Kissner, Z. N. J. Peterson, and D. Song. Provable data possession at untrusted stores. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM Conference on Computer and Communications Security*, pages 598–609. ACM, 2007.

[2] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 326–349. ACM, 2012.

[3] N. Bitansky, A. Chiesa, Y. Ishai, O. Paneth, and R. Ostrovsky. Succinct Non-Interactive Arguments via Linear Interactive Proofs. In *Theory of Cryptography*, pages 315–333. Springer, 2013.

[4] D. Boneh and D. M. Freeman. Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In *Public Key Cryptography–PKC 2011*, pages 1–16. Springer, 2011.

[5] D. Boneh and D.M. Freeman. Homomorphic signatures for polynomial functions. In *Advances in Cryptology–EUROCRYPT 2011*, pages 149–168. Springer, 2011.

[6] D. Boneh and A. Silverberg. Applications of multilinear forms to cryptography. *Contemporary Mathematics*, 324(1):71–90, 2003.

[7] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *Advances in cryptology - EUROCRYPT 2003*, pages 416–432. Springer, 2003.

[8] D. Boneh, D. Freeman, J. Katz, and B. Waters. Signing a linear subspace: Signature schemes for network coding. In *Public Key Cryptography–PKC 2009*, pages 68–87. Springer, 2009.

[9] D. Catalano and D. Fiore. Practical Homomorphic MACs for Arithmetic Circuits. In *EUROCRYPT*, volume 7881, pages 336–352. Springer, 2013.

[10] D. Catalano, D. Fiore, R. Gennaro, and L. Nizzardo. Generalizing homomorphic MACs for arithmetic circuits. In *Public-Key Cryptography–PKC 2014*, pages 538–555. Springer, 2014.

[11] D. Catalano, D. Fiore, and B. Warinschi. Homomorphic Signatures with Efficient Verification for Polynomial Functions. In J.A. Garay and R. Gennaro, editors, *Advances in Cryptology – CRYPTO 2014*, volume 8616 of *Lecture Notes in Computer Science*, pages 371–389. Springer Berlin Heidelberg, 2014.

[12] 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*, CCS '12, pages 501–512. ACM, 2012.

[13] S. Garg, C. Gentry, and S. Halevi. Candidate multilinear maps from ideal lattices. In Thomas Johansson and PhongQ. Nguyen, editors, *Advances in Cryptology EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 1–17. Springer Berlin Heidelberg, 2013.

[14] R. Gennaro and D. Wichs. Fully Homomorphic Message Authenticators. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology - ASIACRYPT 2013*, volume 8270 of *Lecture Notes in Computer Science*, pages 301–320. Springer Berlin Heidelberg, 2013.

[15] R. Gennaro, C. Gentry, and B Parno. Non-Interactive Verifiable Computation: Outsourcing Computation To Untrusted Workers. In *In Proceedings of CRYPTO*. Citeseer, 2010.

[16] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic Span Programs and Succinct NIZKs without PCPs. In *EUROCRYPT*, volume 7881, pages 626–645. Springer, 2013.

[17] 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*, pages 99–108. ACM, 2011.

[18] S. Gorbunov, V. Vaikuntanathan, and D. Wichs. Leveled fully homomorphic signatures from standard lattices. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, STOC '15, pages 469–477, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3536-2.

[19] S. Hohenberger and B. Waters. Short and stateless signatures from the rsa assumption. In *Advances in Cryptology-CRYPTO 2009*, pages 654–670. Springer, 2009.

[20] B. Parno, M. Raykova, and V. Vaikuntanathan. How to Delegate and Verify in Public: Verifiable Computation from Attribute-Based Encryption. In Ronald Cramer, editor, *Theory of Cryptography*, volume 7194 of *Lecture Notes in Computer Science*, pages 422–439. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-28913-2.

[21] B. Parno, J. Howell, C. Gentry, and M. Raykova. Pinocchio: Nearly Practical Verifiable Computation. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 238–252. IEEE, 2013.

[22] H. Shacham and B. Waters. Compact proofs of retrievability. In *Proceedings of the 14th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '08, pages 90–107, Berlin, Heidelberg, 2008. Springer-Verlag.