Context Hiding Multi-Key Linearly Homomorphic Authenticators

Lucas Schabhüser, Denis Butin, and Johannes Buchmann

Technische Universität Darmstadt, Germany
{lschabhueser,dbutin,buchmann}@cdc.informatik.tu-darmstadt.de

Abstract. Demanding computations are increasingly outsourced to cloud platforms. For such outsourced computations, the efficient verifiability of results is a crucial requirement. When sensitive data is involved, the verification of a computation should preserve the privacy of the input values: it should be context hiding. Context hiding verifiability is enabled by existing homomorphic authenticator schemes. However, until now, no context hiding homomorphic authenticator scheme supports multiple independent clients, e.g. multiple keys. Multi-key support is necessary for datasets involving input authenticated by different clients, e.g. multiple hospitals in e-health scenarios. In this paper, we propose the first perfectly context hiding, publicly verifiable multi-key homomorphic authenticator scheme supporting linear functions. Our scheme is provably unforgeable in the standard model, and succinct. Verification time depends only linearly on the number of clients, in an amortized sense.

Keywords: Delegated Computation · Homomorphic Authenticators · Context Hiding

1 Introduction

Today, it is common practice to outsource time-consuming computations to the cloud. In such a situation, it is desirable to be able to verify the outsourced computation. The verification must be *efficient*, by which we mean that the verification procedure is significantly faster than the verified computation itself. Otherwise, the verifier could as well carry out the computation by himself, negating the advantage of outsourcing. In addition, there are scenarios in which the verification is required to provide *input privacy*, i.e. the verification does not reveal anything about the input to the computation. For instance, a cloud service may collect health data of individuals and compute statistics on them. These statistical evaluations are then given to third parties, such as insurance companies. While these third parties must be able to learn the statistical outcome, for privacy reasons, they must not learn the individual health data. Furthermore, many interesting statistics involve *multiple identities*; for instance, computing on health data across datasets provided by multiple hospitals. Keeping identities separate instead of merging all data supports fine-grained authenticity. Furthermore, using different keys instead of copies of a shared key avoids a single point of failure.

With multiple keys, the loss of a single key does not result in a total security loss, in contrast with the shared key approach.

Homomorphic Authenticators. In practice, the efficient verifiability of outsourced computation can be realised using *homomorphic authenticators*. The general idea of homomorphic authenticators is the following. Before delegating inputs to a function, the input values are authenticated. The homomorphic property allows the server to compute an authenticator to the output of a given function from the authenticators to the inputs to said function. In the public key setting, homomorphic authenticators are called *homomorphic signatures*. In the private key setting, they are called *homomorphic MACs*. Input privacy for homomorphic authenticators has been formalized in the form of the *context hiding* property.

State of the Art. Using such authenticators, there are solutions for efficient context hiding verifiability for the case of a *single* client, and for the equivalent case of multiple clients sharing a single secret key [4, 8, 22]. However, no context hiding solution for multiple clients with different keys exists. Fiore et al. [14] already presented multi-key homomorphic authenticators. However, their constructions are not context hiding. Hence the challenge arises to design efficient and context hiding verification procedures for outsourced computing that support multiple clients.

1.1 Contribution Overview

In this paper, we present the first publicly verifiable homomorphic authenticator scheme providing efficient and context hiding verification in the setting of multiple clients (allowing for multiple keys). We construct a multi-key linearly homomorphic signature scheme, and thus focus on the public key setting. We first define the context hiding property in the multi-key case. We then describe our main contribution, a new publicly verifiable multi-key linearly homomorphic authenticator scheme. Our scheme allows to generate an authenticator on the function value of a linear function from authenticators on the input values of various identities without knowledge of the authentication key. Furthermore, our scheme is perfectly context hiding, i.e. the authenticator to the output value does not leak any information about the input values. Using our multi-key homomorphic authenticator scheme, the verification procedure for outsourced computations of linear functions can be implemented as follows. The various clients each upload data, signed under their personal private key, to the cloud. The cloud server computes the result of the given function over these data. It also generates an authenticator to this result from the signatures on the inputs. The verifier uses this authenticator to check for correctness of the computation, by using the verification keys associated to the clients providing input to the computation. Regarding performance, verification time depends only on the number of identities involved (in an amortized sense).

Context Hiding Security for Multiple Clients. On a high level, we first define the context hiding property in the multi-key setting. Intuitively, this property provides some measure of input privacy, i.e. an authenticator to the output of a computation does not leak information about the input to the computation. In the multi-key setting, the question who exactly is meant to be prevented from learning about the input values becomes relevant. In particular, we differentiate between an external adversary — one that has not corrupted any of the identities involved in a computation — and an internal adversary, who has this additional knowledge. Thus we capture the two slightly different notions of keeping the input values private with respect to some outside party (*externally context hiding*), versus keeping the input values confidential with respect to an identity that also provided inputs to the computation (*internally context hiding*).

Our Construction. Then, we provide a concrete instantiation of a publicly verifiable multi-key linearly homomorphic authenticator scheme, i.e. a homomorphic signature scheme with these properties. Our authenticator size is $\mathcal{O}(k)$, where k is the number of identities involved in the computation, thus achieving succinctness. A homomorphically derived authenticator consists of both components associated to an identity id and global elements. In order to prevent the elements associated to id from leaking information about the inputs provided by id, the authenticators are randomized, and the global elements are used to deal with the randomization in order to preserve the homomorphic property. Our verification procedure naturally splits into two parts, only one of which involves the actual outcome of the computation. The other part only depends on the public verification key vk and the function to be evaluated, and can thus be precomputed. This allows for amortized efficient verification, i.e. after an expensive function-dependent one-time precomputation, all subsequent verifications occur in constant time.

Proving Unforgeability. A significant part of this paper is the security reduction used to prove our scheme's unforgeability. In Sec. 4, we present a series of games, followed by several lemmata considering the difference between the games. Most games only differ if a forgery of a specific type is produced by the adversary, i.e. a special case of a forgery, where one or several of the components are correct. We bound the probability of these events and can show that such forgeries imply solving the Discrete Logarithm, the Decisional Diffie–Hellman and the Flexible Diffie–Hellman Inversion problem (introduced by Catalano, Fiore and Nizzardo at CRYPTO 2015 [9]), respectively. Our security reduction is performed in the standard model. Our scheme does not use Fiore et al.'s lattice-based construction [14] and thus features a different structure. Consequently, our proof strategy is also novel. As is common in homomorphic authenticator schemes, we use identifiers or labels. They uniquely identify the position of an input in a dataset. There are elements in the public verification key associated to these labels. When simulating (some of) these security games, the simulator does not have access to the secret signing key and thus cannot run the algorithm Auth. By embedding a trapdoor into the elements associated to labels in the verification

key the simulator can perfectly simulate authenticators when presented with a Flexible Diffie–Hellman Inversion instance (and thus not knowing the secret key).

1.2 Outline

We recall relevant definitions for homomorphic authenticator schemes in Sec. 2 and define the context hiding property in the multi-key setting. In Sec. 3, we present our publicly verifiable multi-key homomorphic authenticator scheme for linear functions. We then address its properties, notably correctness and context hiding. In Sec. 4, we provide a security reduction for our scheme. Next, in Sec. 5, we compare our contribution to existing work on homomorphic authenticators and verifiable computation. Finally, in Sec. 6, we summarize our results and give an outlook to future work and open problems.

2 Formalising Multi-Key Homomorphic Authenticators

In this section, we provide the necessary background for homomorphic authenticators and their properties. We recall the definitions for correctness and unforgeability, ass well as efficiency properties in the form of succinctness and efficient verification. Then we present our generalization of the context hiding property to the multi-key setting. Finally we state the computational assumptions on which the security of our scheme is based.

To accurately describe both correct and legitimate operations for homomorphic authenticators, we use *multi-labeled programs* similarly to Backes, Fiore, and Reischuk [5]. The basic idea is to append a function by several identifiers, in our case *input identifiers* and *dataset identifiers*. Input identifiers label in which order the input values are to be used and dataset identifiers determine which authenticators can be homomorphically combined. The idea is that only authenticators created under the same dataset identifier can be combined. We now give formal definitions.

A *labeled program* $\mathcal{P}$ consists of a tuple $(f, \tau_1, \ldots, \tau_n)$, where $f : \mathcal{M}^n \to \mathcal{M}$ is a function with n inputs and $\tau_i \in \mathcal{T}$ is a label for the i^{th} input of f from some set $\mathcal{T}$. Given a set of labeled programs $\mathcal{P}_1, \ldots, \mathcal{P}_N$ and a function $g : \mathcal{M}^N \to \mathcal{M}$, they can be composed by evaluating g over the labeled programs, i.e. $\mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_N)$. The identity program with label τ is given by $\mathcal{I}_\tau = (f_{id}, \tau)$, where $f_{id} : \mathcal{M} \to \mathcal{M}$ is the identity function. The program $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ can be expressed as the composition of n identity programs $\mathcal{P} = f(\mathcal{I}_{\tau_1}, \ldots, \mathcal{I}_{\tau_n})$.

A *multi-labeled program* $\mathcal{P}_\Delta$ is a pair $(\mathcal{P}, \Delta)$ of the labeled program $\mathcal{P}$ and a dataset identifier Δ. Given a set of k multi-labeled programs with the same dataset identifier Δ, i.e. $(\mathcal{P}_1, \Delta), \ldots, (\mathcal{P}_N, \Delta)$, and a function $g : \mathcal{M}^N \to \mathcal{M}$, a composed multi-labeled program $\mathcal{P}^*_\Delta$ can be computed, consisting of the pair $(\mathcal{P}^*, \Delta)$, where $\mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_N)$. Analogously to the identity program for labeled programs, we refer to a multi-labeled identity program by $\mathcal{I}_{(\tau,\Delta)} = ((f_{id}, \tau), \Delta)$.

In particular, we use labeled programs to identify the different clients. Our multi-key homomorphic authenticators allow the verification of linear functions

evaluated over data signed by different keys. Following the convention of [14], we assume every client has an identity id in some identity space ID, and that public keys can be linked to an identity id. This can, for instance, be achieved by a public-key infrastructure (PKI). In order to identify which inputs (labeled by τ) were authenticated by id, our messages are assigned with a label $l \leftarrow (\mathsf{id}, \tau)$, where id is a client's identity and τ is an input identifier. Following convention, messages are grouped within datasets Δ and homomorphic evaluation is only supported over the same dataset.

Definition 1 (Multi-Key Homomorphic Authenticator ([14]). *A multi-key homomorphic authenticator scheme* $\mathsf{MKHAuth}$ *is a tuple of the following probabilistic polynomial time (PPT) algorithms:*

$\mathsf{Setup}(1^\lambda)$: *On input a security parameter λ, the algorithm returns a set of public parameter* pp*, consisting of (at least) the description of a tag space $\mathcal{T}$, an identity space* ID*, a message space $\mathcal{M}$, and a set of admissible functions $\mathcal{F}$. Given $\mathcal{T}$ and* ID *the label space of the scheme is defined as $\mathcal{L} = \mathsf{ID} \times \mathcal{T}$. The public parameters* pp *will implicitly be inputs to all following algorithms even if not explicitly specified.*

$\mathsf{KeyGen}(\mathsf{pp})$: *On input the public parameters* pp*, the algorithm returns a key triple* $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk})$*, where* sk *is the secret key authentication key,* ek *is a public evaluation key, and* vk *is a verification key that can be either private or public.*

$\mathsf{Auth}(\mathsf{sk}, \Delta, l, m)$: *On input a secret key* sk*, a dataset identifier Δ, a label $l = (\mathsf{id}, \tau)$, and a message m, the algorithm returns an authenticator σ.*

$\mathsf{Eval}(f, \{(\sigma_i, \mathsf{EKS}_i)\}_{i\in[n]})$: *On input a function $f : \mathcal{M}^n \to \mathcal{M}$ and a set $\{(\sigma_i, \mathsf{EKS}_i)\}_{i\in[n]}$ of evaluation keys, the algorithm returns an authenticator σ.*

$\mathsf{Ver}(\mathcal{P}_\Delta, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}}, m, \sigma)$: *On input a multi-labeled program $\mathcal{P}_\Delta$, a set of verification key $\{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}}$, corresponding to the identities* id *involved in the program $\mathcal{P}$, a message $m \in \mathcal{M}$, and an authenticator σ , the algorithm returns either* $\mathsf{1}$*(accept), or* $\mathsf{0}$*(reject).*

If the class $\mathcal{F}$ of admitted functions is the set of linear functions, we call $\mathsf{MKHAuth}$ a *multi-key linearly homomorphic authenticator*. If vk is private, we call $\mathsf{MKHAuth}$ a *multi-key homomorphic MAC*, while for a public vk we call it a *multi-key homomorphic signature*.

We now define properties relevant for the analysis of multi-key homomorphic authenticator schemes: authentication correctness, evaluation correctness, succinctness, unforgeability, efficient verification and context hiding.

Correctness naturally comes in two forms. We require both authenticators created directly with a secret signing key as well as those derived by the homomorphic property to verify correctly.

Definition 2 (Authentication Correctness). *A multi-key homomorphic authenticator scheme* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Auth}, \mathsf{Eval}, \mathsf{Ver})$ *satisfies* authentication correctness *if, for any security parameter λ, any public parameters* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$*, any key triple* $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$*, any label $l = (\mathsf{id}, \tau) \in \mathcal{L}$, any dataset identifier $\Delta \in \{0,1\}^*$, any message $m \in \mathcal{M}$, and any authenticator $\sigma \leftarrow$*

$\mathsf{Auth}(\mathsf{sk}, \Delta, l, m)$ *we have* $\mathsf{Ver}(\mathcal{I}_{(l,\Delta)}, \mathsf{vk}, m, \sigma) = 1$*, where* $\mathcal{I}_{l,\Delta}$ *is the multi-labeled identity program.*

Definition 3 (Evaluation Correctness). *A multi-key homomorphic authenticator scheme* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Auth}, \mathsf{Eval}, \mathsf{Ver})$ *satisfies* authentication correctness *if, for any security parameter* λ*, any public parameters* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$*, any set of key triples* $\{(\mathsf{sk}_{\mathsf{id}}, \mathsf{ek}_{\mathsf{id}}, \mathsf{vk}_{\mathsf{id}})\}_{\mathsf{id} \in \tilde{\mathsf{ID}}}$*, with* $(\mathsf{sk}_{\mathsf{id}}, \mathsf{ek}_{\mathsf{id}}, \mathsf{vk}_{\mathsf{id}}) \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{pp})$ *for all* $\mathsf{id} \in \tilde{\mathsf{ID}}$*, for some subset* $\tilde{\mathsf{ID}} \subset \mathsf{ID}$*, for any dataset identifier* $\Delta \in \{0,1\}^*$*, and any set of program/message/authenticator triples* $\{(\mathcal{P}_i, m_i, \sigma_i)\}_{i \in [N]}$*, such that* $\mathsf{Ver}(\mathcal{P}_{i,\Delta}, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}_i}, m_i, \sigma_i) = 1$ *the following holds: Let* $m^* = g(m_1, \ldots, m_N), \mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_N)$*, and* $\sigma^* = \mathsf{Eval}(g, \{(\sigma_i, \mathsf{EKS}_i)\}_{i \in [N]})$ *where* $\mathsf{EKS}_i = \{\mathsf{ek}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}_i}$*. Then* $\mathsf{Ver}(\mathcal{P}^*_\Delta, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}^*}, m^*, \sigma^*) = 1$ *holds.*

We now consider two properties impacting the practicality of homomorphic authenticator schemes. Succinctness on a high level guarantees that bandwidth requirements for deploying such a scheme are low. Efficient verification allows for low computational effort on behalf of the verifier.

Definition 4 (Succinctness [14]). *A multi-key homomorphic authenticator scheme* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Auth}, \mathsf{Eval}, \mathsf{Ver})$ *is said to be succinct if the size of every authenticator depends only logarithmically on the size of a dataset. However, we allow authenticators to depend on the number of keys involved in the computation. More formally, let* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$*,* $\mathcal{P} = (f, l_1, \ldots, l_n)$*, with* $l_i = (\mathsf{id}_i, \tau_i)$*,* $\{(\mathsf{sk}_{\mathsf{id}}, \mathsf{ek}_{\mathsf{id}}, \mathsf{vk}_{\mathsf{id}}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})\}_{\mathsf{id} \in \mathcal{P}}$*, and* $\sigma_i \leftarrow \mathsf{Auth}(\mathsf{sk}_{\mathsf{id}_i}, \Delta, l_i, m_i)$ *for all* $i \in [n]$*. A multi-key homomorphic authenticator is said to be succinct if there exists a fixed polynomial* p *such that* $|\sigma| = p(\lambda, k, \log n)$*, where* $\sigma = \mathsf{Eval}(f, \{\sigma_i, \mathsf{ek}_{\mathsf{id}_i}\}_{i \in [n]})$ *and* $k = |\{\mathsf{id} \in \mathcal{P}\}|$*.*

We explicitly allow the size of the authenticators to depend on the number of identities $k = |\{\mathsf{id} \in \mathcal{P}\}|$ involved in the computation.

Like Libert and Yung [21], we call a key *concise* if its size is independent of the input size n.

Definition 5 (Efficient Verification [9]). *A multi-key homomorphic authenticator scheme for multi-labeled programs allows for* efficient verification *if there exist two additional algorithms* $(\mathsf{VerPrep}, \mathsf{EffVer})$ *such that:*

$\mathsf{VerPrep}(\mathcal{P}, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}})$: *Given a labeled program* $\mathcal{P} = (f, l_1, \ldots, l_n)$*, and a set of verification keys* $\{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}}$ *this algorithm generates a concise verification key* $\mathsf{vk}_{\mathcal{P}}$*. This does* not *depend on a dataset identifier* Δ*.*

$\mathsf{EffVer}(\mathsf{vk}_{\mathcal{P}}, \Delta, m, \sigma)$**:** *Given a concise verification key* $\mathsf{vk}_{\mathcal{P}}$*, a dataset* Δ*, a message* m*, and an authenticator* σ*, it outputs* `1` *or* `0`*.*

The above algorithms are required to satisfy the following two properties:

Correctness: *Let* $\{(\mathsf{sk}_{\mathsf{id}}, \mathsf{ek}_{\mathsf{id}}, \mathsf{vk}_{\mathsf{id}})\}_{\mathsf{id} \in \mathsf{ID}}$ *be a set of honestly generated keys and* $(\mathcal{P}_\Delta, m, \sigma)$ *be a tuple such that,* $\mathsf{Ver}(\mathcal{P}_\Delta, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}}, m, \sigma) = 1$*. Then, for every* $\mathsf{vk}_{\mathcal{P}} \xleftarrow{\$} \mathsf{VerPrep}(\mathcal{P}, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}})$*,* $\Pr[\mathsf{EffVer}(\mathsf{vk}_{\mathcal{P}}, \Delta, m, \sigma) = 0] = \mathsf{negl}(\lambda)$*, where* $\mathsf{negl}(\lambda)$ *denotes any function negligible in the security parameter* λ*.*

Amortized Efficiency: *Let $\mathcal{P}$ be a program, let $m_1, \ldots, m_n$ be valid input values and let $t(n)$ be the time required to compute $\mathcal{P}(m_1, \ldots, m_n)$ with output m. Then, for any $\mathsf{vk}_{\mathcal{P}} \xleftarrow{\$} \mathsf{VerPrep}(\mathcal{P}, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}})$, and any $\Delta \in \{0,1\}^*$ the time required to compute $\mathsf{EffVer}(\mathsf{vk}_{\mathcal{P}}, \Delta, m, \sigma)$ is $t' = o(t(n))$, where $\sigma_i \leftarrow \mathsf{Auth}(\mathsf{sk}_{\mathsf{id}_i}, \Delta, l_i, m_i)$ for $i \in [n]$, and $\sigma \leftarrow \mathsf{Eval}(f, \{(\sigma_i, \mathsf{EKS}_i)\}_{i\in[n]})$.*

Here, *efficiency* is used in an amortized sense. There is a function-dependent pre-processing phase, so that the cost of verification amortizes over multiple datasets.

For the notion of unforgeability of a multi-key homomorphic authenticator scheme $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Auth}, \mathsf{Eval}, \mathsf{Ver})$, we define the following experiment between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. During the experiment, the adversary $\mathcal{A}$ can adaptively query the challenger $\mathcal{C}$ for authenticators on messages of his choice under labels of his choice. He can also make verification queries and corrupt clients. Intuitively, the homomorphic property allows anyone (with access to the evaluation keys) to derive new authenticators. This can be checked by the use of the corresponding program in the verification algorithm. An adversary should however not be able to derive authenticators beyond that. Preventing forgeries on programs involving inputs of corrupted clients is impossible in many cases (e.g. for any linear function), as knowledge of the secret key allows the creation of arbitrarily many authenticators under any multi-label (l, Δ). However, this security definition captures that knowledge of one client's secret key does not allow any forgeries on a computation not involving this corrupted client.

Definition 6 $(\mathsf{HomUF} - \mathsf{CMA}_{\mathcal{A},\mathsf{MKHAuth}}(\lambda))$**.**

Setup: *$\mathcal{C}$ runs $\mathsf{Setup}(1^\lambda)$ to obtain the public parameters pp that are sent to $\mathcal{A}$.*

Authentication Queries: *$\mathcal{A}$ can adaptively submit queries of the form (Δ, l, m) where Δ is a dataset identifier, $l = (\mathsf{id}, \tau) \in \mathcal{L}$ is a label, and $m \in \mathcal{M}$ is a message of its choice. $\mathcal{C}$ answers as follows:*

- *If (Δ, l, m) is the first query for the dataset Δ, $\mathcal{C}$ initializes an empty list $L_\Delta = \emptyset$ and proceeds as follows.*
- *If (Δ, l, m) is the first query with identity id, $\mathcal{C}$ generates keys $(\mathsf{sk}_{\mathsf{id}}, \mathsf{ek}_{\mathsf{id}}, \mathsf{vk}_{\mathsf{id}}) \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{pp})$ (that are implicitly assigned to identity id), gives $(\mathsf{ek}_{\mathsf{id}}, \mathsf{vk}_{\mathsf{id}})$ to $\mathcal{A}$ and proceeds as follows.*
- *If (Δ, l, m) is such that $(l, m) \notin L_\Delta$, $\mathcal{C}$ computes $\sigma_l \leftarrow \mathsf{Auth}(\mathsf{sk}_{\mathsf{id}}, \Delta, l, m)$ ($\mathcal{C}$ has already generated keys for the identity id), returns σ_l to $\mathcal{A}$ and updates the list $L_\Delta \leftarrow L_\Delta \cup (l, m)$.*
- *If (Δ, l, m) is such that $(l, \cdot) \in L_\Delta$ (which means that the adversary had already made a query (Δ, l, m') for the identity id), then $\mathcal{C}$ ignores the query.*

Verification Queries: *$\mathcal{A}$ is also given access to a verification oracle. Namely the adversary can submit a query $(\mathcal{P}_\Delta, m, \sigma)$ and $\mathcal{C}$ replies with the output of $\mathsf{Ver}(\mathcal{P}_\Delta, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}}, m, \sigma)$.*

Corruption Queries: *The adversary $\mathcal{A}$ has access to a corruption oracle. At the beginning of the experiment, the challenger $\mathcal{C}$ initializes an empty list*

$L_{\mathsf{corr}} = \emptyset$ of corrupted identities. During the game $\mathcal{A}$ can adaptively query identities $\mathsf{id} \in \mathsf{ID}$. If $\mathsf{id} \notin L_{\mathsf{corr}}$ then $\mathcal{C}$ replies with the triple $(\mathsf{sk}_{\mathsf{id}}, \mathsf{ek}_{\mathsf{id}}, \mathsf{vk}_{\mathsf{id}})$ (that is generated using KeyGen if not done before) and updates the list $L_{\mathsf{corr}} \leftarrow L_{\mathsf{corr}} \cup \mathsf{id}$. If $\mathsf{id} \in L_{\mathsf{corr}}$, then $\mathcal{C}$ replies with the triple $(\mathsf{sk}_{\mathsf{id}}, \mathsf{ek}_{\mathsf{id}}, \mathsf{vk}_{\mathsf{id}})$ assigned to id before.

Forgery: *In the end, $\mathcal{A}$ outputs a tuple $(\mathcal{P}^*_{\Delta^*}, m^*, \sigma^*)$. The experiment outputs 1 if the tuple returned by $\mathcal{A}$ is a forgery as defined below (see Def. 7), and 0 otherwise.*

This describes the case of publicly verifiable multi-key homomorphic authenticators. For multi-key homomorphic MACs, the verification keys vk_i are *not* given to the adversary $\mathcal{A}$ during the security experiment.

Definition 7 (Forgery [14]). *Consider a run of $\mathsf{HomUF} - \mathsf{CMA}_{\mathcal{A},\mathsf{MKHAuth}}(\lambda)$ where $(\mathcal{P}^*_{\Delta^*}, m^*, \sigma^*)$ is the tuple returned by the adversary in the end of the experiment, with $\mathcal{P}^* = (f^*, l^*_1, \ldots, l^*_n)$. This is a forgery if $\mathsf{Ver}({\mathcal{P}^*}_{\Delta^*}, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}^*}, m^*, \sigma^*) = 1$, $\mathsf{id} \notin L_{corr}$ (i.e. no identity involved in $\mathcal{P}^*$ is corrupted, and at least one of the following properties is satisfied:*

Type 1: *The list L_{Δ^*} was not initialized during the security experiment, i.e. no message was ever committed under the dataset identifier Δ^*.*

Type 2: *${\mathcal{P}^*}_{\Delta^*}$ is well defined with respect to list L_{Δ^*} and m^* is not the correct output of the computation, i.e. $m^* \neq f^*(m_1, \ldots, m_n)$*

Type 3: *${\mathcal{P}^*}_{\Delta^*}$ is not well defined with respect to L_{Δ^*} (see Def. 8).*

Definition 8 (Well Defined Program). *A labeled program $\mathcal{P} = (f, l_1, \ldots, l_n)$ is* well defined *with respect to a list $L \subset \mathcal{L} \times \mathcal{M}$ if one of the two following cases holds: First, there are messages $m_1, \ldots, m_n$ such that $(l_i, m_i) \in L \; \forall i \in [n]$. Second, there is an $i \in \{1, \ldots, n\}$ such that $(l_i, \cdot) \notin L$ and $f(\{m_j\}_{(l_j, m_j) \in L} \cup \{m'_k\}_{(l_k, \cdot) \notin L})$ is constant over all possible choices of $m'_k \in \mathcal{M}$.*

If f is a linear function, the labeled program $\mathcal{P} = (f, l_1, \ldots, l_n)$, with $f(m_1, \ldots, m_n) = \sum_{i=1}^{n} f_i m_i$ fulfills the second condition if and only if $f_k = 0$ for all $(l_k, \cdot) \notin L$.

Definition 9 (Unforgeability). *A multi-key homomorphic authenticator scheme $\mathsf{MKHAuth}$ is* unforgeable *if for any PPT adversary $\mathcal{A}$ we have*

$$\Pr[\mathsf{HomUF} - \mathsf{CMA}_{\mathcal{A},\mathsf{MKHAuth}}(\lambda) = 1] = \mathsf{negl}(\lambda).$$

Additionally, we will use the following statement:

Lemma 1. *Let $\mathsf{MKHAuth} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Auth}, \mathsf{Eval}, \mathsf{Ver})$ be a multi-key linearly homomorphic authenticator scheme over a message space $\mathcal{M} \subset R^t$ for some ring R and integer t. If $\mathsf{MKHAuth}$ is secure against Type 2 forgeries, then $\mathsf{MKHAuth}$ is also secure against Type 3 forgeries.*

Proof. This is an immediate corollary of a result by Freeman [15, Proposition 2.3].

We also consider a relaxation of the unforgeability definition in which the adversaries ask for corruptions in a non-adaptive way. More precisely, we say that an adversary $\mathcal{A}$ makes non-adaptive corruption queries if for every identity id asked to the corruption oracle, id was not queried earlier in the game to the authentication oracle or the verification oracle. For this class of adversaries, corruption queries are of no help as the adversary can generate keys on its own. We will use the following lemma:

Lemma 2 ([14, Proposition 1]). $\mathsf{MKHAuth}$ *is unforgeable against adversaries that do not make corruption queries if and only if* $\mathsf{MKHAuth}$ *is unforgeable against adversaries that make non-adaptive corruption queries.*

We are now ready to provide our notion of input privacy, in the form of the context hiding property and adapting this to the multi-key setting.

Our definition for the context hiding property is inspired by Gorbunov et al.'s definition [18] for the single-key case. However, in our case, the simulator is explicitly given the circuit for which the authenticator is supposed to verify. With respect to this difference, our definition is more general. We stress that the circuit is not hidden in either of these notions. Furthermore, we differentiate between an external adversary and an internal adversary, that corrupts some of the various identities involved in a computation, i.e. knows their secret keys and inputs to a computation. Such an adversary will learn more than the outcome of the computation, since it knows some of the secret keys. It is however desirable for any non-corrupted party to achieve context hiding privacy even against other parties involved in the computation, as far as that is possible. We now formally define context hiding for both kinds of adversaries.

Definition 10 (Context Hiding). *A multi-key homomorphic authenticator scheme for multi-labeled programs is* externally context hiding *if there exist two additional PPT procedures* $\tilde{\sigma} \leftarrow \mathsf{Hide}(\{\mathsf{vk_{id}}\}_{\mathsf{id}\in\mathsf{ID}}, m, \sigma)$ *and* $\mathsf{HideVer}(\{\mathsf{vk_{id}}\}_{\mathsf{id}\in\mathsf{ID}}, \mathcal{P}_\Delta, m, \tilde{\sigma})$ *such that:*

Correctness: *For any* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, $(\mathsf{sk_{id}}, \mathsf{ek_{id}}, \mathsf{vk_{id}}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ *and any tuple* $(\mathcal{P}_\Delta, m, \sigma)$*, such that* $\mathsf{Ver}(\mathcal{P}_\Delta, \{\mathsf{vk_{id}}\}_{\mathsf{id}\in\mathcal{P}}, m, \sigma) = 1$*, and* $\tilde{\sigma} \leftarrow \mathsf{Hide}(\{\mathsf{vk_{id}}\}_{\mathsf{id}\in\mathsf{ID}}, m, \sigma)$*, it holds that* $\mathsf{HideVer}(\{\mathsf{vk_{id}}\}_{\mathsf{id}\in\mathsf{ID}}, \mathcal{P}_\Delta, m, \tilde{\sigma}) = 1$.

Unforgeability: *The homomorphic authenticator scheme is unforgeable (see Def. 9) when replacing the algorithm* Ver *with* $\mathsf{HideVer}$ *in the security experiment.*

Context Hiding Security: *There is a simulator* Sim *such that, for any fixed (worst-case) choice of* $\{(\mathsf{sk_{id}}, \mathsf{ek_{id}}, \mathsf{vk_{id}}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})\}_{\mathsf{id}\in\mathcal{P}}$*, any multi-labeled program* $\mathcal{P}_\Delta = (f, l_1, \ldots, l_n, \Delta)$*, messages* $m_1, \ldots, m_n$*, and distinguisher* $\mathcal{D}$ *there exists a function* $\epsilon(\lambda)$ *such that* $|\Pr[\mathcal{D}(\mathsf{Hide}(\{\mathsf{vk_{id}}\}_{\mathsf{id}\in\mathcal{P}}, m, \sigma)) = 1] - \Pr[\mathcal{D}(\mathsf{Sim}(\{\mathsf{sk_{id}}\}_{\mathsf{id}\in\mathcal{P}}, \mathcal{P}_\Delta, m)) = 1]| = \epsilon(\lambda)$*, where* $\sigma_i \leftarrow \mathsf{Auth}(\mathsf{sk_{id}}_i, \Delta, l_i, m_i)$*,* $m \leftarrow f(m_1, \ldots, m_n)$*,* $\sigma \leftarrow \mathsf{Eval}(f, \{(\sigma_i, \mathsf{EKS}_i)\}_{i\in[n]})$*,and the probabilities are taken over the randomness of* $\mathsf{Auth}, \mathsf{Hide}$ *and* Sim.

If $\epsilon(\lambda) = \mathsf{negl}(\lambda)$*, we call the multi-key homomorphic authenticator scheme* statistically externally context hiding. *If* $\epsilon(\lambda) = 0$*, we call it* perfectly externally context hiding.

If for the context hiding security we even have $|\Pr[\mathcal{D}(\{\mathsf{sk_{id}}\}_{\mathsf{id}\in\tilde{\mathsf{ID}}}, \mathsf{Hide}(\mathsf{vk}, m, \sigma)) = 1] - \Pr[\mathcal{D}(\mathsf{Sim}(\{\mathsf{sk_{id}}\}_{\mathsf{id}\in\mathcal{P}}, \mathcal{I}, \mathcal{P}_\Delta, m)) = 1]| = \epsilon(\lambda)$*, where* $\mathcal{I} = (\{\mathsf{sk_{id}}\}_{\mathsf{id}\in\tilde{\mathsf{ID}}}, \{(m_{(\tau,\mathsf{id})}, \sigma_{(\tau,\mathsf{id})})\}_{\mathsf{id}\in\tilde{\mathsf{ID}}})$*,* $\tilde{\mathsf{ID}} \subset \mathsf{ID}$ *is a set of corrupted identities and the rest is like before, we call the multi-key homomorphic authenticator scheme (statistically or perfectly, depending on* $\epsilon(\lambda)$ *as above)* internally context hiding.

2.1 Computational Assumptions

We recall the computational assumptions on which our schemes are based.

Definition 11. *Let* $\mathcal{G}$ *be a generator of cyclic groups of order* p *and let* $\mathbb{G} \xleftarrow{\$} \mathcal{G}(1^\lambda)$*. We say the Discrete Logarithm assumption (DL) holds in* $\mathbb{G}$ *if there exists no PPT adversary* $\mathcal{A}$ *that, given* (g, g^a) *for a random generator* $g \in \mathbb{G}$ *and random* $a \in \mathbb{Z}_p$*, can output* a *with more than negligible probability, i.e. if* $\Pr[a \leftarrow \mathcal{A}(g, g^a) | g \xleftarrow{\$} \mathbb{G},\ a \xleftarrow{\$} \mathbb{Z}_p] = \mathsf{negl}(\lambda)$.

Definition 12 (Asymmetric bilinear groups). *An* asymmetric bilinear group *is a tuple* $\mathsf{bgp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$*, such that:*

- $\mathbb{G}_1, \mathbb{G}_2$*, and* $\mathbb{G}_T$ *are cyclic groups of prime order* p*,*
- $g_1 \in \mathbb{G}_1$ *and* $g_2 \in \mathbb{G}_2$ *are generators for their respective groups,*
- *the DL assumption holds in* $\mathbb{G}_1, \mathbb{G}_2$*, and* $\mathbb{G}_T$*,*
- $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ *is bilinear, i.e.* $e({g_1}^a, {g_2}^b) = e(g_1, g_2)^{ab}$ *holds for all* $a, b \in \mathbb{Z}$*,*
- e *is non-degenerate, i.e.* $e(g_1, g_2) \neq 1$*, and*
- e *is efficiently computable.*

We will write $g_t = e(g_1, g_2)$*.*

Definition 13. *Let* $\mathcal{G}$ *be a generator of asymmetric bilinear groups and let* $\mathsf{bgp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \xleftarrow{\$} \mathcal{G}(1^\lambda)$*. We say the Decisional Diffie–Hellman assumption (DDH) holds in* $\mathbb{G}_1$ *if, for every PPT adversary* $\mathcal{A}$*,*

$$|\Pr[\mathcal{A}(\mathsf{bgp}, g_1^x, g_1^y, g_1^{xy}) | x, y \xleftarrow{\$} \mathbb{Z}_p] - \Pr[\mathcal{A}(\mathsf{bgp}, g_1^x, g_1^y, g_1^z) | x, y, z \xleftarrow{\$} \mathbb{Z}_p]| \leq \mathsf{negl}(\lambda)$$

We also use the Flexible Diffie–Hellman Inversion hardness assumption, introduced by Catalano, Fiore and Nizzardo [9]. In the extended version of their CRYPTO2015 paper [8], they formally investigate the hardness of this assumption and analyse it in the generic group model.

Definition 14 ([9]). *Let* $\mathcal{G}$ *be a generator of asymmetric bilinear groups and let* $\mathsf{bgp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \xleftarrow{\$} \mathcal{G}(1^\lambda)$*. We say the Flexible Diffie–Hellman Inversion (FDHI) assumption holds in* bgp *if, for every PPT adversary* $\mathcal{A}$*,*

$$\Pr[W \in \mathbb{G}_1 \backslash \{1_{\mathbb{G}_1}\} \wedge W' = W^{\frac{1}{z}} : (W, W') \leftarrow \mathcal{A}(g_1, g_2, g_2^z, g_2^v, g_1^{\frac{z}{v}}, g_1^r, g_1^{\frac{r}{v}}) | z, r, v \xleftarrow{\$} \mathbb{Z}_p] \leq \mathsf{negl}(\lambda).$$

3 A Publicly Verifiable Multi-Key Linearly Homomorphic Authenticator Scheme

In this section, we present our multi-key homomorphic signature scheme, i.e. a publicly verifiable homomorphic authenticator. It supports linear functions. We analyse it with respect to its correctness, its succinctness and efficient verifiability. Finally, we proof that our scheme is indeed perfectly context hiding. Unforgeability is dealt with in the next section.

3.1 Our Construction

Notation If we have n possibly distinct messages $m_1, \ldots, m_n$, we denote by m_i the i$^{\text{th}}$ message. Since our messages are vectors, i.e. $m \in \mathbb{Z}_p^T$, we write $m[j]$ to indicate the j$^{\text{th}}$ entry of message vector m for $j \in [T]$. Therefore $m_i[j]$ denotes the j$^{\text{th}}$ entry of the i$^{\text{th}}$ message. Given a linear function f, its i^{th} coefficient is denoted by f_i, i.e. $f(m_1, \ldots, m_n) = \sum_{i=1}^{n} f_i m_i$. If we have n possibly distinct authenticator components, e.g. $\Lambda_1, \ldots, \Lambda_n$, we denote by Λ_i the i$^{\text{th}}$ component. A single authenticator comprises different components, corresponding to different identities. For authenticator Λ, we denote by Λ_{id} the component for identity id. We denote by $\Lambda_{\mathsf{id},i}$ the component of the i$^{\text{th}}$ authenticator corresponding to identity id. We use a regular signature scheme $\mathsf{Sig} = (\mathsf{KeyGen}_{sig}, \mathsf{Sign}_{sig}, \mathsf{Ver}_{sig})$ as a building block. $(\mathsf{sk}_{sig}, \mathsf{pk}_{sig})$ denotes a secret/public key pair for Sig.

$\mathsf{Setup}(1^\lambda)$: On input a security parameter λ, this algorithm chooses the parameters $k, n, T \in \mathbb{Z}$, a bilinear group $\mathsf{bgp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \xleftarrow{\$} \mathcal{G}(1^\lambda)$, the message space $\mathcal{M} = \mathbb{Z}_p^T$, the tag space $\mathcal{T} = [n]$, and the identity space $\mathsf{ID} = [k]$. Additionally it fixes a pseudorandom function $F : \mathcal{K} \times \{0,1\}^* \to \mathbb{Z}_p$. It chooses $H_1, \ldots H_T \in \mathbb{G}_1$ uniformly at random. It outputs the public parameters $\mathsf{pp} = (k, n, T, \mathsf{bgp}, H_1, \ldots, H_T, F, \lambda)$.

$\mathsf{KeyGen}(\mathsf{pp})$: On input the public parameters pp, the algorithm chooses $K \in \mathcal{K}$ uniformly at random. It runs $(\mathsf{sk}_{sig}, \mathsf{pk}_{sig}) \leftarrow \mathsf{KeyGen}_{sig}(1^\lambda)$. It chooses $x_1, \ldots, x_n, y \in \mathbb{Z}_p$ uniformly at random. It sets $h_i = g_t^{x_i}$ for all $i \in [n]$, as well as $Y = g_2^y$. It sets $\mathsf{sk} = (K, \mathsf{sk}_{sig}, x_1, \ldots x_n, y)$, $\mathsf{ek} = 0$, $\mathsf{vk} = (\mathsf{pk}_{sig}, h_1, \ldots, h_n, Y)$ and outputs $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk})$. Each identity performs KeyGen individually, and hence obtains its own key tuple $(\mathsf{sk}_{\mathsf{id}}, \mathsf{ek}_{\mathsf{id}}, \mathsf{vk}_{\mathsf{id}})$.

$\mathsf{Auth}(\mathsf{sk}, \Delta, l, m)$: On input a secret key sk, a dataset identifier Δ, a label $l = (\mathsf{id}, \tau)$, and a message m, the algorithm computes $z = F_K(\Delta)$, sets $Z = g_2^z$ and binds this parameter to the dataset by signing it, i.e. it computes $\sigma_\Delta \leftarrow \mathsf{Sign}_{sig}(\mathsf{sk}_{sig}, Z||\Delta)$. Then it chooses $r, s \in \mathbb{Z}_p$ uniformly at random and sets $R = g_1^{r-ys}, S = g_2^{-s}$. It parses $l = (\tau, \mathsf{id})$ and computes $A = \left(g_1^{x_\tau + r} \cdot \prod_{j=1}^{T} H_j^{ym[j]}\right)^{\frac{1}{z}}$ and $C = g_1^s \cdot \prod_{j=1}^{T} H_j^{m[j]}$. It sets $\Lambda = \{(\mathsf{id}, \sigma_\Delta, Z, A, C)\}$ and outputs $\sigma = (\Lambda, R, S)$.

$\mathsf{Eval}(f, \{(\sigma_i, \mathsf{EKS}_i)\}_{i\in[n]})$: On input an function $f : \mathcal{M}^n \to \mathcal{M}$ and a set $\{(\sigma_i, \mathsf{EKS}_i)\}_{i\in[n]}$ of authenticators and evaluation keys (in our construction,

no evaluation keys are need, so this set contains only authenticators), the algorithm parses $f = (f_1, \dots f_n)$ as a coefficient vector. It parses each σ_i as (Λ_i, R_i, S_i) and sets $R = \prod_{i=1}^{n} R_i^{f_i}$, $S = \prod_{i=1}^{n} S_i^{f_i}$. Set $L_{\mathsf{ID}} = \bigcup_{i=1}^{n} \{\mathsf{id}_i\}$. For each $\mathsf{id} \in L_{\mathsf{ID}}$ it chooses a pair $(\sigma_{\Delta,\mathsf{id}}, Z_{\mathsf{id}})$ uniformly at random such that a tuple $(\mathsf{id}, \sigma_{\Delta,\mathsf{id}}, Z_{\mathsf{id}}, A, C)$ is contained in one of the Λ_i. More formally, it chooses $(\sigma_{\Delta,\mathsf{id}}, Z_{\mathsf{id}}) \xleftarrow{\$} \{(\sigma, Z) \mid \exists\ A, C \mid (\mathsf{id}, \sigma_{\Delta}, Z, A, C) \in \bigcup_{i=1}^{n} \Lambda_i\}$. Then it computes $A_{\mathsf{id}} = \prod_{\substack{i=1 \\ \mathsf{id}_i = \mathsf{id}}}^{n} A_i^{f_i}$, $C_{\mathsf{id}} = \prod_{\substack{i=1 \\ \mathsf{id}_i = \mathsf{id}}}^{n} C_i^{f_i}$, and sets $\Lambda_{\mathsf{id}} = \{\mathsf{id}, \sigma_{\Delta,\mathsf{id}}, Z_{\mathsf{id}}, A_{\mathsf{id}}, C_{\mathsf{id}}\}$. Set $\Lambda = \bigcup_{\mathsf{id} \in L_{\mathsf{ID}}} \Lambda_{\mathsf{id}}$. It returns $\sigma = (\Lambda, R, S)$.

$\mathsf{Ver}(\mathcal{P}_\Delta, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}}, m, \sigma)$: On input a multi-labeled program $\mathcal{P}_\Delta$, a set of verification key $\{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}}$, corresponding to the identities id involved in the program $\mathcal{P}$, a message $m \in \mathcal{M}$, and an authenticator σ , the algorithm parses $\sigma = (\Lambda, R, S)$. For each id such that $(\mathsf{id}, \sigma_{\Delta,\mathsf{id}}, Z_{\mathsf{id}}, A_{\mathsf{id}}, C_{\mathsf{id}}) \in \Lambda$ it takes $\mathsf{pk}_{sig,\mathsf{id}}$ from $\mathsf{vk}_{\mathsf{id}}$ and checks whether $\mathsf{Ver}_{sig}(\mathsf{pk}_{sig,\mathsf{id}}, Z_{\mathsf{id}}||\Delta, \sigma_{\Delta,\mathsf{id}}) = 1$ holds, i.e. whether there is a valid signature on $(Z_{\mathsf{id}}||\Delta)$. If any check fails it returns 0. Otherwise it checks whether the following equations hold: $\prod_{\mathsf{id}\in\mathcal{P}} e\left(A_{\mathsf{id}}, Z_{\mathsf{id}}\right) = \prod_{i=1}^{n} h_{l_i}^{f_i} \cdot \prod_{\mathsf{id}\in\mathcal{P}} e\left(C_{\mathsf{id}}, Y_{\mathsf{id}}\right) \cdot e\left(R, g_2\right)$, as well as $e\left(g_1, S\right) \cdot e\left(\prod_{\mathsf{id}\in\mathcal{P}} C_{\mathsf{id}}, g_2\right) = e\left(\prod_{j=1}^{T} H_j^{m[j]}, g_2\right)$. If they do, it outputs 1, otherwise it outputs 0.

Our authenticators σ consist of several components, so we have $\sigma = (\Lambda, R, S)$, where Λ is a list of elements, each associated to some identity id, i.e. $\Lambda = \{(\mathsf{id}, \sigma_{\Delta,\mathsf{id}}, Z_{\mathsf{id}}, A_{\mathsf{id}}, C_{\mathsf{id}})\}_{\mathsf{id}\in\mathcal{P}}$. The R and S components are global. Note, that the $A_{\mathsf{id}}, C_{\mathsf{id}}$ are randomized in order to provide internal context hidingness and the global components are used to preserve the homomorphic property.

3.2 Correctness and Efficiency

We now analyse our scheme with respect to its correctness and efficiency. An obvious requirement for a homomorphic authenticator scheme is to be correct. Due to the homomorphic property, there are two different types of correctness to consider (see Def. 2 and Def. 3). The former ensures, that our scheme MKHAuth can be used as a conventional signature scheme, by verifying it with respect to the identity program. The latter property ensures a correct homomorphic evaluation will also be verified as correct.

Theorem 1. *The scheme* MKHAuth *presented in Subsection 3.1 satisfies authentication correctness (see Def. 2), if* Sig *is a correct signature scheme.*

Proof. Take any public parameters $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, any key triple $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$, any label $l = (\mathsf{id}, \tau) \in \mathcal{L}$, any dataset identifier $\Delta \in \{0,1\}^*$, and any authenticator $\sigma \leftarrow \mathsf{Auth}(\mathsf{sk}, \Delta, l, m)$. Then we have $\sigma = (\Lambda, R, S)$ and $\Lambda = (\mathsf{id}, \sigma_\Delta, Z, A, C)$. By construction we have $\sigma_\Delta \leftarrow \mathsf{Sign}_{sig}(\mathsf{sk}_{sig}, Z||\Delta)$ and if Sig is a correct signature scheme then $\mathsf{Ver}_{sig}(\mathsf{pk}_{sig,\mathsf{id}}, Z_{\mathsf{id}}||\Delta, \sigma_{\Delta,\mathsf{id}}) = 1$

holds. We have by construction $e(A,Z) = e\left(\left(g_1^{x_\tau+r}\cdot\prod_{j=1}^{T}H_j^{ym[j]}\right)^{\frac{1}{z}},g_2^z\right)$ $= e\left(g_1^{x_\tau+r-ys}\cdot g_1^{ys}\cdot\prod_{j=1}^{T}H_j^{ym[j]},g_2\right) = g_t^{x_\tau+r-ys}\cdot e\left(g_1^s\cdot\prod_{j=1}^{T}H_j^{m[j]},g_2^y\right) =$ $h_l\cdot e(C,Y)\cdot e(R,g_2)$, and $e(g_1,S)\cdot e(C,g_2) = g_t^{-s}\cdot g_t^s e\left(\prod_{j=1}^{T}H_j^m[j],g_2\right) =$ $e\left(\prod_{j=1}^{T}H_j^{m[j]},g_2\right)$, and thus $\mathsf{Ver}(\mathcal{I}_{l,\Delta},\mathsf{vk},m,\sigma)=1$ holds.

Theorem 2. *The scheme* MKHAuth *presented in Subsection 3.1 satisfies evaluation correctness (see Def 3), if* Sig *is a correct signature scheme.*

Proof. We have $\mathcal{P}^*_\Delta = g(\mathcal{P}^*_{\Delta,1},\dots\mathcal{P}^*_{\Delta,N})$. Since Sig is a correct signature scheme, $\mathsf{Ver}_{sig}(\mathsf{pk}_{sig,\mathsf{id}},Z_{\mathsf{id}}||\Delta,\sigma_{\Delta,\mathsf{id}})=1$ holds for all $\mathsf{id}\in\mathcal{P}^*$. If $\mathsf{Ver}(\mathcal{P}_{i,\Delta},\{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}_i},m_i,\sigma_i)=1$ holds, then in particular

$$\prod_{\mathsf{id}\in\mathcal{P}_i}e(A_{\mathsf{id},i},Z_{\mathsf{id}}) = \prod_{k=1}^{n}h_{l_{i,k}}^{f_{i,k}}\cdot e\left(\prod_{\mathsf{id}\in\mathcal{P}_i}C_{\mathsf{id},i},Y_{\mathsf{id}}\right)\cdot e(R_i,g_2)$$

holds as well as $e(g_1,S_i)\cdot e\left(\prod_{\mathsf{id}\in\mathcal{P}_i}C_{\mathsf{id},i},g_2\right) = e\left(\prod_{j=1}^{T}H_j^{m_i[j]},g_2\right)$ for all $i\in[N]$. Write g as a coefficient vector $(c_1,\dots c_N)$. Without loss of generality let $\{\mathsf{id}\in\mathcal{P}_i\}=\{\mathsf{id}\in\mathcal{P}_j\}$ for all $i,j\in[n]$. Let f_k for $k\in[n]$ denote the coefficients describing $\mathcal{P}=g(\mathcal{P}_1,\dots,\mathcal{P}_N)$. Then we have $f_k=\sum_{i=1}^{N}c_if_{i,k}$. We have

$$\prod_{i=1}^{N}\left(\prod_{\mathsf{id}\in\mathcal{P}_i}e(A_{\mathsf{id},i},Z_{\mathsf{id},i})\right)^{c_i} = \prod_{i=1}^{N}\left(\prod_{k=1}^{n}h_{l_{i,k}}^{f_{i,k}}\cdot\prod_{\mathsf{id}\in\mathcal{P}_i}e(C_{\mathsf{id},i},Y_{\mathsf{id}})\cdot e(R_i,g_2)\right)^{c_i}$$

and

$$\begin{aligned}\prod_{\mathsf{id}\in\mathcal{P}^*}e(A^*_{\mathsf{id}},Z^*_{\mathsf{id}}) &= \prod_{i=1}^{N}\left(\prod_{\mathsf{id}\in\mathcal{P}_i}e(A_{\mathsf{id},i},Z_{\mathsf{id},i})\right)^{c_i}\\ &= \prod_{i=1}^{N}\left(\prod_{k=1}^{n}h_{l_{i,k}}^{f_{i,k}}\cdot\prod_{\mathsf{id}\in\mathcal{P}_i}e(C_{\mathsf{id},i},Y_{\mathsf{id}})\cdot e(R_i,g_2)\right)^{c_i}\\ &= \prod_{k=1}^{n}h_{l_k}^{f_k}\cdot\prod_{\mathsf{id}\in\mathcal{P}^*}e(C^*_{\mathsf{id}},Y^*_{\mathsf{id}})\cdot e(R^*,g_2)\end{aligned}$$

We also have

$$\begin{aligned}&e(g_1,S^*)\cdot e(C^*,g_2) = e\left(g_1,\prod_{i=1}^{N}S_i^{c_i}\right)\cdot e\left(\prod_{i=1}^{N}C_i^{c_i},g_2\right)\\ &= \prod_{i=1}^{N}e\left(\prod_{j=1}^{T}H_j^{m_i[j]},g_2\right)^{c_i} = e\left(\prod_{j=1}^{T}H_j^{\sum_{i=1}^{N}c_im_i[j]},g_2\right) = e\left(\prod_{j=1}^{T}H_j^{m^*[j]},g_2\right)\end{aligned}$$

Thus all checks of Ver pass.

We now consider our scheme's efficiency properties, first with respect to bandwidth, in the form of succinctness, and then with respect to verification time.

A trivial solution to constructing a homomorphic signature scheme is to (conventionally) sign every input, and during Eval to just concatenate all the signatures along with the corresponding values. Verification then consists of checking every input value and then redoing the computation. This naive solution is obviously undesirable in terms of bandwidth, efficiency and does not provide any privacy guarantees.

Succinctness guarantees that a homomorphically derived signature is still small, thus keeping bandwidth requirements low. Efficient verification ensures that the time required to check an authenticator is low. This is achieved by splitting Ver into two sub-algorithms, one of which can be precomputed, and the other one EffVer can be faster than natively computing the function itself.

Theorem 3. *The scheme* $\mathsf{MKHAuth}$ *presented in Subsection 3.1 is succinct (see Def. 4).*

Proof. An authenticator consists of (at most) $k+1$ elements of $\mathbb{G}_1$, k elements of $\mathbb{G}_2$, k identities $\mathsf{id} \in \mathsf{ID}$, and k (conventional) signatures. None of this depends on the input size n. Therefore $\mathsf{MKHAuth}$ is succinct.

Theorem 4. *The scheme* $\mathsf{MKHAuth}$ *presented in Subsection 3.1 allows for efficient verification (see Def. 5).*

Proof. We describe the algorithms $(\mathsf{VerPrep}, \mathsf{EffVer})$:

$\mathsf{VerPrep}(\mathcal{P}, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}})$: On input a labeled program $\mathcal{P} = (f, l_1, \ldots, l_n)$, with f given by its coefficient vector $(f_1, \ldots f_n)$, the algorithm takes $(Y_{\mathsf{id}}, \mathsf{pk}_{sig,\mathsf{id}})$ from $\mathsf{vk}_{\mathsf{id}}$. For label $l_i = (\mathsf{id}_i, \tau_i)$ it takes h_{l_i} from $\mathsf{vk}_{\mathsf{id}_i}$. It computes $h_{\mathcal{P}} \leftarrow \prod_{i=1}^{n} h_{l_i}^{f_i}$ and outputs $\mathsf{vk}_{\mathcal{P}} \leftarrow (h_{\mathcal{P}}, \{(Y_{\mathsf{id}}, \mathsf{pk}_{sig,\mathsf{id}})\}_{\mathsf{id}\in\mathcal{P}})$. This is independent of the input size n.

$\mathsf{EffVer}(\mathsf{vk}_{\mathcal{P}}, \Delta, m, \sigma)$**:** On input a concise verification key $\mathsf{vk}_{\mathcal{P}}$, a dataset Δ, a message m, and an authenticator σ, the algorithm parses $\sigma = (\Lambda, R, S)$. For each $\mathsf{id} \in \mathcal{P}$ it checks whether $\mathsf{Ver}_{sig}(\mathsf{pk}_{sig,\mathsf{id}}, Z_{\mathsf{id}}||\Delta, \sigma_{\Delta,\mathsf{id}}) = 1$ holds. If not, it outputs $\mathtt{0}$. Otherwise, it checks whether the following equation holds: $\prod_{\mathsf{id}\in\mathcal{P}} e\left(A_{\mathsf{id}}, Z_{\mathsf{id}}\right) = h_{\mathcal{P}} \cdot \prod_{\mathsf{id}\in\mathcal{P}} e\left(C_{\mathsf{id}}, Y_{\mathsf{id}}\right) \cdot e\left(R, g_2\right)$ as well as $e\left(g_1, S\right) \cdot e\left(\prod_{\mathsf{id}\in\mathcal{P}} C_{\mathsf{id}}, g_2\right) = e\left(\prod_{j=1}^{T} H_j^{m[j]}, g_2\right)$. If they do, it outputs $\mathtt{1}$, otherwise it outputs $\mathtt{0}$.

This obviously satisfies correctness. We can see that the runtime of EffVer is $\mathcal{O}(k)$, and is independent of the input size n. Thus, for $n \gg k$, $\mathsf{MKHAuth}$ allows for efficient verification.

3.3 Context Hidingness

We now showcase our scheme's privacy property. On a high level, we want an authenticator to the output of a computation not to leak information about the

inputs to the computation, which we have formalized in Def. 10. Intuitively, the outcome of a function (e.g. the average) reveals significantly less information than the individual inputs to the computation. In our scenario, multiple clients upload data to a cloud server that performs the computation, and allows for public verification of the result due to the use of homomorphic authenticators. The context hiding property ensures that the verifier cannot use the authenticators provided to him to derive additional information about the inputs, beyond his knowledge of the output.

Theorem 5. *The scheme* MKHAuth *presented in Subsection 3.1 is perfectly internally context hiding (see Def. 10) and therefore also externally context hiding.*

Proof. First, in our case, the algorithm Hide is just the identity function. More precisely, we have $\mathsf{Hide}(\{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathsf{ID}}, m, \sigma) = \sigma$, for all possible verification keys $\mathsf{vk}_{\mathsf{id}}$, messages m and authenticators σ. Thus we have $\mathsf{HideVer} = \mathsf{Ver}$, so correctness and unforgeability hold by Theorem 1, Theorem 2, and Theorem 6.

We show how to construct a simulator Sim that outputs authenticators perfectly indistinguishable from the ones obtained by running Eval. Consider that for all linear functions f, we have $f(m_1, \ldots, m_n) = \sum_{i=1}^{n} f_i m_i = \sum_{i\in\mathcal{I}} f_i m_i + \sum_{j\in\mathcal{J}} f_j m_j$, for each $\mathcal{I}, \mathcal{J} \subset [n]$, with $\mathcal{I} \cup \mathcal{J} = [n]$ and $\mathcal{I} \cap \mathcal{J} = \emptyset$.

$\mathcal{S}$ can simulate the corrupted parties perfectly. By the identity shown before, we can in our case therefore reduce internal context hiding security to external context hiding security. We now show external context hiding security. Parse the simulator's input as $\mathsf{sk}_{\mathsf{id}} = (K_{\mathsf{id}}, K'_{\mathsf{id}}, \mathsf{sk}_{sig,\mathsf{id}}, x_{1,\mathsf{id}}, \ldots x_{n,\mathsf{id}}, y)$, $m = (m[1], \ldots, m[T])$, and $\mathcal{P}_\Delta = (f, l_1, \ldots, l_n, \Delta)$. With this information, the simulator computes:

$Z'_{\mathsf{id}} = g_2^{z_{\mathsf{id}}}$ where $z_{\mathsf{id}} \leftarrow F_{K_{\mathsf{id}}}(\Delta)$	$\sigma'_{\Delta,\mathsf{id}} \xleftarrow{\$} \mathsf{Sign}_{sig}(\mathsf{sk}_{sig,\mathsf{id}}, Z_{\mathsf{id}}\|\|\Delta)$
$r'_{\mathsf{id}} \xleftarrow{\$} \mathbb{Z}_p$	$r' = \sum_{l=(\tau,\mathsf{id})\in\mathcal{P}} c_l r'_{\mathsf{id}}$
$s'_{\mathsf{id}} \xleftarrow{\$} \mathbb{Z}_p$	$s' = \sum_{\mathsf{id}\in\mathcal{P}} s_l$
$A'_{\mathsf{id}} = \left(g_1^{\sum_{(\mathsf{id},\tau)\in\mathcal{P}} x_\tau c_\tau + r'} \cdot \prod_{j=1}^{T} H_j^{ym[j]}\right)^{\frac{1}{z_{\mathsf{id}}}}$	$\mathsf{id}^* \xleftarrow{\$} \mathsf{ID}$
$C'_{\mathsf{id}} = g_1^{-s'_{\mathsf{id}}}$ for all $\mathsf{id} \neq \mathsf{id}^*$	$C'_{\mathsf{id}^*} = g_1^{-s'_{\mathsf{id}^*}} \cdot \prod_{j=1}^{T} H_j^{m[j]}$
$R' = g_1^{r' + \sum_{\mathsf{id}\in\mathcal{P}} y_{\mathsf{id}} s'_{\mathsf{id}}}$	$S' = g_2^{-s'}$
$\Lambda' = \bigcup_{\mathsf{id}\in\mathcal{P}} \{(A'_{\mathsf{id}}, Z'_{\mathsf{id}}, \sigma'_{\Delta,\mathsf{id}})\}$	

The simulator outputs the authenticator $\sigma' = (\Lambda', R', S')$. We now show that this simulator allows for perfectly context hiding security. We fix arbitrary key pairs $(\mathsf{sk}_{\mathsf{id}}, \mathsf{pk}_{\mathsf{id}})$, a multi-labeled program $\mathcal{P}_\Delta$, and messages $m_1, \ldots, m_n \in \mathbb{Z}_p^T$.

Let $\sigma \leftarrow (f, \{(\sigma_i, \mathsf{EKS}_i)\}_{i\in[n]})$ and parse it as $\sigma = (\Lambda, R, S)$. We look at each component of the authenticator. We have $Z_{\mathsf{id}} = F_{K_{\mathsf{id}}}(\Delta)$ by definition and therefore also $Z_{\mathsf{id}} = Z'_{\mathsf{id}}$. Y_{id} and Y'_{id} are both taken from the public keys and therefore identical. In particular we also have $z_{\mathsf{id}} = z'_{\mathsf{id}}$. We have $\sigma_{\Delta_{\mathsf{id}}} = \mathsf{Sign}_{sig}(\mathsf{sk}', Z_{\mathsf{id}}||\Delta)$ by definition and since $Z = Z'$ therefore also $\sigma_\Delta = \sigma'_\Delta$

since Sign_{sig} is deterministic. These components are identical and therefore indistinguishable to any distinguisher $\mathcal{D}$.

A'_{id} is a uniformly random (u.r.) element of $\mathbb{G}_1$, as r'_{id} is also u.r. A_{id} is a u.r. element of $\mathbb{G}_1$, as $r_{\mathsf{id}} = \sum_{\mathsf{id}\in l} c_l r_l$ is u.r. as a linear combination of u.r. elements. C'_{id} is a u.r. element of $\mathbb{G}_1$, as s'_{id} is also u.r. C_{id} is a u.r. element of $\mathbb{G}_1$, as $s_{\mathsf{id}} = \sum_{\mathsf{id}\in l} c_l s_l$ is u.r. as a linear combination of u.r. elements. $R'$is a u.r. element of $\mathbb{G}_1$, as all r'_{id} are also u.r. R is a u.r. element of $\mathbb{G}_1$, as $r = \sum_{\mathsf{id}\in\mathcal{P}} r_{\mathsf{id}}$ is u.r. as a linear combination of u.r. elements. The $A'_{\mathsf{id}}, C'_{\mathsf{id}}$ as well as R' uniquely define S' and $A_{\mathsf{id}}, C_{\mathsf{id}}$ as well as R uniquely define S.

Thus, all simulated elements have the identical distribution as the ones from the real evaluation. They correspond to a different choice of randomness during Auth. This holds even if all secret keys $\mathsf{sk}_{\mathsf{id}}$ are known to $\mathcal{D}$. Hence σ and σ' are perfectly indistinguishable for any (computationally unbounded) distinguisher $\mathcal{D}$.

4 Unforgeability

In delegated computations, the question of the correctness of the result arises. Homomorphic authenticators aim at making these computations verifiable, thus allowing for the detection of incorrect results. It should therefore be infeasibly for any adversary to produce a authenticator that passes the Ver algorithm, that has not been produced by honestly performing the Eval algorithm. This has been formalized in Def. 6.

In this section, we present the security reduction for the unforgeability of our scheme. To this end, we first describe a sequence of games, allowing us to argue about different variants of forgeries. We then present a series of lemmata, where we bound the difference between those games.

Since our authenticators have multiple components, we consider specific types of forgeries in the various games, i.e. ones where one or multiple components are indeed correct, and in our final security reduction we consider the generic case. When simulating the final two games, the issue of providing signatures, without knowing the correct secret key arises. Here we use the elements $h_{\mathsf{id},i}$ taken from the public keys associated to the label $l = (\mathsf{id}, i)$ and embed an information theoretically hidden trapdoor into them, which we use to answer signing queries. Note, that by (conventionally) signing the concatenation $(\Delta||Z_{\mathsf{id}})$ we use a similar approach to Fiore et al. [14, Theorem 2]. Not directly using their more generic approach however yields a lower bound on the adversary's overall success probability in the security experiment.

Theorem 6. *The scheme* $\mathsf{MKHAuth}$ *presented in Subsection3.1 is unforgeable (see Def. 9), if* Sig *is an unforgeable (EU-CMA [17]) signature scheme, F is a pseudorandom function and $\mathcal{G}$ is a bilinear group generator, such that the DL assumption (see Def. 11), the DDH assumption (see Def. 14) hold.*

Proof. We can deal with corruptions via our generic result of Lemma 2. It is thus sufficient to prove the security against adversaries that make no corruptions. Recall that any corrupted party provides their key tuples $(\mathsf{sk}_{\mathsf{id}}, \mathsf{ek}_{\mathsf{id}}, \mathsf{vk}_{\mathsf{id}})$ to

the adversary, giving the adversary additional knowledge in order for him to adaptively query messages.

To prove Theorem 6, we define a series of games with the adversary $\mathcal{A}$ and we show that the adversary $\mathcal{A}$ wins, i.e. any game outputs $\mathtt{1}$, only with negligible probability. Following the notation of [9], we write $G_i(\mathcal{A})$ to denote that a run of game i with adversary $\mathcal{A}$ returns $\mathtt{1}$. We use flag values bad_i, initially set to false. If, at the end of each game, any of these previously defined flags is set to true, the game simply outputs $\mathtt{0}$. Let $\mathsf{Bad_i}$ denote the event that $\mathsf{bad_i}$ is set to true during game i. Using Lemma 1, any adversary who outputs a Type 3 forgery (see Def. 7) can be converted into one that outputs a Type 2 forgery. Hence we only have to deal with Type 1 and Type 2 forgeries.

Game 1 is the security experiment $\mathsf{HomUF-CMA}_{\mathcal{A},\mathsf{MKHAuth}}(\lambda)$ between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$, where $\mathcal{A}$ makes no corruption queries and only outputs Type 1 or Type 2 forgeries.

Game 2 is defined as Game 1, except for the following change: Whenever $\mathcal{A}$ returns a forgery $(\mathcal{P}^*_{\Delta^*}, m^*, \sigma^*)$ and the list L_{Δ^*} has not been initialized by the challenger during the queries, then Game 2 sets $\mathsf{bad}_2 = \mathsf{true}$. It is worth noticing that after this change the game never outputs 1 if $\mathcal{A}$ returns a Type 1 forgery. In Lemma 3, we show that Bad_2 cannot occur if Sig is unforgeable.

Game 3 is defined as Game 2, except that the keyed pseudorandom function F_K is replaced by a random function $\mathcal{R} : \{0,1\}^* \to \mathbb{Z}_p$. In Lemma 4, we show that these two games are indistinguishable if F is pseudorandom.

Game 4 is defined as Game 3, except for the following changes. It computes $\hat{m} = f^*(m_1, \ldots, m_n)$, as well as $\hat{\sigma} = \mathsf{Eval}(f^*, \{(\sigma_i, \mathsf{EKS}_i)\}_{i\in[n]})$, i.e. it runs an honest computation over the queried messages and generated authenticators in dataset Δ^*. The challenger runs an additional check. If $\prod_{j=1}^{T} H_j^{m^*[j]} = \prod_{j=1}^{T} H_j^{\hat{m}[j]}$ and $\hat{m} \neq m^*$ it sets $\mathsf{bad}_4 = \mathsf{true}$. We clearly have $|\Pr[G_3(\mathcal{A})] - \Pr[G_4(\mathcal{A})]| \leq \Pr[\mathsf{Bad}_4]$. In Lemma 5, we show that any adversary $\mathcal{A}$ for which Bad_4 occurs implies a solver for the DL problem.

Game 5 is defined as Game 4, except for the following change. The challenger runs an additional check. If $C^* = \hat{C}$ and $m^* \neq \hat{m}$ it sets $\mathsf{bad}_5 = \mathsf{true}$, where C^* is a component of the forged authenticator σ^* and $\hat{C}$ is a component of the honest execution of Eval over the queried data set, as defined in Game 4. We have $|\Pr[G_4(\mathcal{A})] - \Pr[G_5(\mathcal{A})]| \leq \Pr[\mathsf{Bad}_5]$. In Lemma 6, we show that any adversary $\mathcal{A}$ for which Bad_5 occurs implies a solver for the DDH problem.

Game 6 is defined as Game 5, except for the following change. At the beginning $\mathcal{C}$ chooses $\mu \in [Q]$ uniformly at random, with $Q = \mathsf{poly}(\lambda)$ is the number of queries made by $\mathcal{A}$ during the game. Let $\Delta_1, \ldots, \Delta_Q$ be all the datasets queried by $\mathcal{A}$. Then, if in the forgery $\Delta^* \neq \Delta_\mu$, set $\mathsf{bad}_6 = \mathsf{true}$. In Lemma 7, we show that $\Pr[G_5(\mathcal{A})] = Q \cdot \Pr[G_6(\mathcal{A})]$.

Game 7 is defined as Game 6, except for the following change. The challenger runs an additional check. If $\mathsf{Ver}(\mathcal{P}^*_{\Delta^*}, \{\mathsf{vk_{id}}\}_{\mathsf{id}\in\mathcal{P}^*}, m^*, \sigma^*) = 1$ as well as $\hat{m} \neq m^*$ and $\prod_{\mathsf{id}\in\mathcal{P}^*} e\left(\hat{A}_{\mathsf{id}}, Z^*_{\mathsf{id}}\right) = \prod_{\mathsf{id}\in\mathcal{P}^*} e\left(A^*_{\mathsf{id}}, Z^*_{\mathsf{id}}\right)$, where $\hat{A}_{\mathsf{id}}, A^*_{\mathsf{id}}$ are the components taken from $\hat{\sigma}$ and σ^* respectively, then $\mathcal{C}$ sets $\mathsf{bad}_7 = \mathsf{true}$. We

have $|\Pr[G_6(\mathcal{A})] - \Pr[G_7(\mathcal{A})]| \leq \Pr[\mathsf{Bad}_7]$. In Lemma 8, we show that any adversary $\mathcal{A}$ for which Bad_7 occurs implies a solver for the FDHI problem.

Finally, in Lemma 9, we show that any adversary $\mathcal{A}$ that wins Game 7 implies a solver for the FDHI problem. Together, Lemma 3—9 prove Theorem 6 and we have $\Pr[\mathcal{G}(\mathcal{A})] \leq \mathsf{Adv}^{\mathsf{UF-CMA}}_{\mathsf{Sig},\mathcal{F}}(\lambda) + \mathsf{Adv}^{PRF}_{F,\mathsf{D}}(\lambda) + (1-\frac{1}{p}) \cdot \mathsf{Adv}^{DL}_{\mathcal{S}}(\lambda) + \mathsf{Adv}^{DDH}_{\mathcal{S}}(\lambda) + 2Q\mathsf{Adv}^{FDHI}_{\mathcal{S}}(\lambda)$.

Lemma 3. *For every PPT adversary $\mathcal{A}$, there exists a PPT forger $\mathcal{F}$ such that* $|\Pr[G_1(\mathcal{A})] - \Pr[G_2(\mathcal{A})]| \leq \mathsf{Adv}^{\mathsf{UF-CMA}}_{\mathsf{Sig},\mathcal{F}}(\lambda)$.

Proof. Games 1 and 2 only differ if Bad_2 occurs, i.e. the list L_{Δ^*} was never initialized during the security experiment. In case of a successful forgery this means that there are valid signatures $\sigma_{\Delta^*,\mathsf{id}}$ for the concatenation $(\Delta^*||Z_{\mathsf{id}})$. even though no signature on $(\Delta^*||\cdot)$ was ever generated by the challenger. This immediately leads to an existential forgery for the regular signature scheme Sig, i.e. $\Pr[\mathsf{Bad}_2] = |\Pr[G_1(\mathcal{A})] - \Pr[G_2(\mathcal{A})]| \leq \mathsf{Adv}^{\mathsf{UF-CMA}}_{\mathsf{Sig},\mathsf{F}}(\lambda)$.

Lemma 4. *For every PPT adversary $\mathcal{A}$ running Game 3, there exists a PPT distinguisher $\mathcal{D}$ such that* $|\Pr[G_3(\mathcal{A})] - \Pr[G_2(\mathcal{A})]| \leq \mathsf{Adv}^{PRF}_{F,\mathsf{D}}(\lambda)$.

Proof. Assume we have a noticeable difference $|\Pr[G_3(\mathcal{A})] - \Pr[G_2(\mathcal{A})]| \geq \epsilon$. Since the only difference between these games is the replacement of the pseudorandom function F by the random function $\mathcal{R}$, this immediately leads to a distinguisher $\mathcal{D}$ that achieves an advantage of ϵ against the pseudorandomness of F.

Lemma 5. *For every PPT adversary $\mathcal{A}$ running Game 4, there exists a PPT simulator $\mathcal{S}$ such that* $\Pr[\mathsf{Bad}_4] = (1-\frac{1}{p}) \cdot \mathsf{Adv}^{DL}_{\mathcal{S}}(\lambda)$.

Proof. Assume we have a PPT adversary $\mathcal{A}$ that can produce a successful forgery during Game 4. We will show how a simulator $\mathcal{S}$ can use this to break the DL problem in $\mathbb{G}_1$. It takes as input $(\mathsf{bgp}, h_1 \in \mathbb{G}_1)$.

Setup Simulator $\mathcal{S}$ chooses $a_j, b_j \xleftarrow{\$} \mathbb{Z}_p$ uniformly at random for $i \in [T-1]$ as well as $a \xleftarrow{\$} \mathbb{Z}_p$ uniformly at random and sets $H_j = g_1^{a_j} h_1^{b_j}$ for $j \in [T]$.

Queries Simulator $\mathcal{S}$ can run KeyGen honestly and answer any authentication queries honestly.

Forgery Let $(\mathcal{P}^*_{\Delta^*}, m^*, \sigma^*)$ be the forgery returned by $\mathcal{A}$. $\mathcal{S}$ follows Game 4 to compute $\hat{m}, \hat{\sigma}$. For $\mathsf{bad}_4 = \mathsf{true}$, we have $\prod_{j=1}^{T} H_j^{m^*[j]} = \prod_{j=1}^{T} H_j^{\hat{m}[j]}$ and $m^* \neq \hat{m}$. This is only possible for $T > 1$. $\mathcal{S}$ sets $a = \sum_{j=1}^{T} a_j(\hat{m}[j] - m^*[j])$, $b = \sum_{j=1}^{T} b_j(m^*[j] - \hat{m}[j])$. If $b = 0$, it aborts. Otherwise, it outputs

$$x = \frac{a}{b} = \frac{\sum_{j=1}^{T} a_j(\hat{m}[j] - m^*[j])}{\sum_{j=1}^{T} b_j(m^*[j] - \hat{m}[j])}.$$

Since we have

$$\prod_{j=1}^{T} H_j^{m^*[j]} = \prod_{j=1}^{T} H_j^{\hat{m}[j]} \Leftrightarrow \prod_{j=1}^{T} H_j^{(m^*[j]-\hat{m}[j])} = 1$$
$$\Leftrightarrow \prod_{j=1}^{T} \left(g_1^{a_j} h_1^{b_j}\right)^{(m^*[j]-\hat{m}[j])} = 1$$
$$\Leftrightarrow h_1^{\sum_{j=1}^{T} b_j(m^*[j]-\hat{m}[j])} = g_1^{\sum_{j=1}^{T} a_j(\hat{m}[j]-m^*[j])} \Leftrightarrow h_1 = g_1^x$$

this is a solution to the DL problem. The b_j are information-theoretically hidden from the adversary $\mathcal{A}$, and are thus independent of $\mathcal{A}$'s output. Since $m^* \neq \hat{m}$, there exists a $j \in [T]$ such that $(m^*[j] - \hat{m}[j]) \neq 0$. For any tuple $(b_1, \ldots, b_{j-1}, b_{j+1}, \ldots b_T) \in \mathbb{Z}_p^{T-1}$, there exists a unique $b_j \in \mathbb{Z}_p$ such that $b = \sum_{j=1}^{T} b_j(m^*[j] - \hat{m}[j]) = 0$. Since all b_j are chosen uniformly at random, the probability that $\mathcal{S}$ aborts is therefore $\frac{p^{T-1}}{p^T} = \frac{1}{p}$.

Lemma 6. *For every PPT adversary $\mathcal{A}$ running Game 5, there exists a PPT simulator $\mathcal{S}$ such that* $\Pr[\mathsf{Bad}_5] = \mathsf{Adv}_{\mathcal{S}}^{DDH}(\lambda)$.

Proof. Assume we have a PPT adversary $\mathcal{A}$ that can produce a successful forgery during Game 5. We will show how a simulator $\mathcal{S}$ can use this to break the DDH problem in $\mathbb{G}_1$. It takes as input a tuple $(\mathsf{bgp}, g_1^x, g_1^y, g_1^z)$.

Setup Simulator $\mathcal{S}$ sets $\mathsf{bgp}' = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1^x, g_2, e)$ and chooses $c_j \in \mathbb{Z}_p$ uniformly at random for $j = 0, \ldots, T$ and sets $H_j = g_1^{c_j}$.

Queries Simulator $\mathcal{S}$ can run KeyGen honestly and answer any authentication queries honestly.

Forgery Let $(\mathcal{P}^*_{\Delta^*}, m^*, \sigma^*)$ be the forgery returned by $\mathcal{A}$. $\mathcal{S}$ follows Game 5 to compute $\hat{m}, \hat{\sigma}$. We have

$$e\left(g_1^x, S^*\right) \cdot e\left(C^*, g_2\right) = \prod_{j=1}^{T} H_j^{m^*[j]} = g_1^{\sum_{j=1}^{T} c_j m^*[j]},$$

as well as

$$e\left(g_1^x, \hat{S}\right) \cdot e\left(\hat{C}, g_2\right) = \prod_{j=1}^{T} H_j^{\hat{m}[j]} = g_1^{\sum_{j=1}^{T} c_j \hat{m}[j]}.$$

We set $M = \sum_{j=1}^{T} c_j(\hat{m}[j] - m^*[j])$. We now have $z = xy$ if and only if $e\left(g_1^z, g_2\right) = e\left(g_1^y, \left(\frac{S^*}{\hat{S}}\right)^{\frac{1}{M}}\right)$. Since $\mathsf{bad}_4 = \mathsf{false}$, we have $M \neq 0$.

Lemma 7. *For every PPT adversary $\mathcal{A}$ running Game 6, we have* $\Pr[G_5(\mathcal{A})] = Q \cdot \Pr[G_6(\mathcal{A})]$.

Proof. First, $\Pr[G_5(\mathcal{A})] = \Pr[G_6(\mathcal{A}) \wedge \mathsf{bad}_6 = \mathsf{false}] = \Pr[G_6(\mathcal{A}) \mid \mathsf{bad}_6 = \mathsf{false}] \cdot \Pr[\mathsf{bad}_6 = \mathsf{false}]$, since Game 6 will always output 0 when Bad_6 occurs. Second, observe that when Bad_6 does not occur, i.e. $\mathsf{bad}_6 = \mathsf{false}$, the challenger guessed the dataset Δ^* correctly and the outcome of Game 6 is identical to the outcome of Game 5. Since μ is chosen uniformly at random and is completely hidden to $\mathcal{A}$, we have $\Pr[\mathsf{bad}_6 = \mathsf{false}] = \frac{1}{Q}$ and therefore $\Pr[G_6(\mathcal{A})] = \frac{1}{Q}\Pr[G_5(\mathcal{A})]$.

Lemma 8. *For every PPT adversary $\mathcal{A}$ running Game 7, there exists a PPT simulator $\mathcal{S}$ such that* $\Pr[\mathsf{Bad}_7] = \mathsf{Adv}_{\mathcal{S}}^{FDHI}(\lambda)$.

Proof. Assume we have a PPT adversary $\mathcal{A}$ that can produce a successful forgery during Game 7 such that $\mathsf{bad}_7 = \mathsf{true}$. We will show how a simulator $\mathcal{S}$ can use this to break the FDHI assumption. Given $(g_1, g_2, g_2^z, g_2^v, g_1^{\frac{z}{v}}, g_1^r, g_1^{\frac{r}{v}})$ simulator $\mathcal{S}$ simulates Game 7.

Setup Simulator $\mathcal{S}$ chooses $c_j \in \mathbb{Z}_p$ uniformly at random for $j = 0, \ldots, T$ and sets $H_j = g_1^{c_j}$. It outputs the public parameters $\mathsf{pp} = (k, n, T, \mathsf{bgp}, H_1, \ldots, H_T, \lambda)$.

Key Generation Simulator $\mathcal{S}$ chooses an index $\mu \in [Q]$ uniformly at random. During the key generation it chooses $a_l, b_l \in \mathbb{Z}_p$ uniformly at random for all $l \in \mathcal{L}$. It sets $h_l = g_t^{a_l} \cdot e(g_1, g_2^z)^{b_l}$. It honestly runs $(\mathsf{sk}_{sig}, \mathsf{pk}_{sig}) \leftarrow \mathsf{KeyGen}_{sig}(1^\lambda)$, chooses y_{id} uniformly at random, and sets $Y_{\mathsf{id}} = (g_2^z)^{y_{\mathsf{id}}}$ for all $\mathsf{id} \in \mathsf{ID}$. It gives the public keys $\mathsf{ek}_{\mathsf{id}} = \emptyset$, $\mathsf{vk}_{\mathsf{id}} = (\mathsf{pk}_{sig,\mathsf{id}}, h_{\mathsf{id},1}, \ldots, h_{\mathsf{id},n}, Y_{\mathsf{id}})$ to $\mathcal{A}$ for all $\mathsf{id} \in \mathsf{ID}$.

Queries Let k be a counter for the number of datasets queried by $\mathcal{A}$. (Initially it sets $k = 1$). For every new queried dataset Δ simulator $\mathcal{S}$ creates a list L_Δ of pairs (l, m), which collects all the label/message pairs queried by the adversary on Δ and the respectively generated authenticators.
Moreover, whenever the k^{th} new dataset Δ_k is queried, $\mathcal{S}$ does the following: If $k = \mu$, it samples a random $\xi_{\mathsf{id},\mu} \in \mathbb{Z}_p$, for all $\mathsf{id} \in \mathsf{ID}$ sets $Z_{\mathsf{id},\mu} = (g_2^z)^{\xi_{\mathsf{id},\mu}}$ and stores $\xi_{\mathsf{id},\mu}$. If $k \neq \mu$, it samples a random $\xi_{\mathsf{id},k} \in \mathbb{Z}_p$ and sets $Z_{\mathsf{id},k} = (g_2^v)^{\xi_{\mathsf{id},k}}$ and stores $\xi_{\mathsf{id},k}$. Since all $Z_{\mathsf{id},k}$ are randomly distributed in $\mathbb{G}_2$, they have the same distribution as in Game 7. Given a query (Δ, l, m) with $\Delta = \Delta_k$, simulator $\mathcal{S}$ first computes $\sigma_{\Delta_k,\mathsf{id}} = \mathsf{Sign}(\mathsf{sk}_{sig,\mathsf{id}}, (\Delta_k || Z_{k,\mathsf{id}}))$.
If $k \neq \mu$, it samples $\rho_l, s_l \in \mathbb{Z}_p$ uniformly at random and computes $A_l = \left((g_1^{\frac{z}{v}})^{b_l + y_{\mathsf{id}} s_l + \sum_{j=1}^{T} y_{\mathsf{id}} c_j m[j]} \cdot (g_1^{\frac{r}{v}})^{\rho_l}\right)^{\frac{1}{\xi_{\mathsf{id},k}}}$, $R_l = g_1^{-a_l} \cdot (g_1^r)^{\rho_l}$, $S_l = g_2^{-s_l}$, as well as $C_l = g_1^{s_l + \sum_{j=1}^{T} c_j m[j]}$.
It sets $\Lambda_l = (\mathsf{id}, \sigma_{sig,\mathsf{id}}, Z_{k,\mathsf{id}}, A_l, C_l)$ and gives (Λ_l, R_l, S_l) to $\mathcal{A}$.
We have

$$e(A_l, Z_{k,\mathsf{id}}) = e\left(\left(\left(g_1^{\frac{z}{v}}\right)^{b_l + y_{\mathsf{id}} s_l + \sum_{j=1}^{T} y_{\mathsf{id}} c_j m[j]} \cdot \left(g_1^{\frac{r}{v}}\right)^{\rho_l}\right)^{\frac{1}{\xi_{\mathsf{id},k}}}, (g_2^v)^{\xi_{\mathsf{id},k}}\right)$$

$$= e\left((g_1^z)^{b_l + y_{\mathsf{id}} s_l + \sum_{j=1}^{T} y_{\mathsf{id}} c_j m[j]} \cdot (g_1^r)^{\rho_l}, g_2\right)$$

$$= e\left(g_1^{b_l z + a_l - a_l + r\rho_l}, g_2\right) \cdot e\left(g_1^{s_l + \sum_{j=1}^{T} c_j m[j]}, g_2^{z y_{\mathsf{id}}}\right) = h_l \cdot e\left(R, g_2\right) \cdot e\left(C, Y_{\mathsf{id}}\right)$$

as well as

$$e\left(g_1, S_l\right) \cdot e\left(C_l, g_2\right) = g_t^{-s+s+\sum_{j=1}^{T} c_j m[j]} = e\left(\prod_{j=1}^{T} H_j^{m[j]}, g_2\right)$$

hence this output is indistinguishable from the challenger's output during Game 7.

If $k = \mu$, simulator $\mathcal{S}$ computes $A_l = \left((g_1^{\frac{z}{v}})^{b_l + y_{\mathsf{id}} s_l + \sum_{j=1}^{T} y_{\mathsf{id}} c_j m[j]}\right)^{\frac{1}{\xi_{\mathsf{id},k}}}$, $R_l = g_1^{-a_l}$, $C_l = g_1^{s_l + \sum_{j=1}^{T} y_{\mathsf{id}} c_j m[j]}$, as well as $S_l = g_2^{-s_l}$. It sets $\Lambda_l = (\mathsf{id}, \sigma_{sig,\mathsf{id}}, Z_{k,\mathsf{id}}, A_l, C_l)$ and gives (Λ_l, R_l, S_l) to $\mathcal{A}$.
We have

$$e\left(A_l, Z_{k,\mathsf{id}}\right) = e\left(\left(g_1^{b_l + y_{\mathsf{id}} s_l + \sum_{j=1}^{T} y_{\mathsf{id}} c_j m[j]}\right)^{\frac{1}{\xi_{\mathsf{id},k}}}, (g_2^z)^{\xi_{\mathsf{id},k}}\right)$$

$$= e\left((g_1^z)^{b_l + y_{\mathsf{id}} s_l + \sum_{j=1}^{T} y_{\mathsf{id}} c_j m[j]}, g_2\right)$$

$$= e\left(g_1^{b_l z + a_l - a_l}, g_2\right) \cdot e\left(g_1^{s_l + \sum_{j=1}^{T} c_j m[j]}, g_2^{z y_{\mathsf{id}}}\right) = h_l \cdot e\left(R, g_2\right) \cdot e\left(C, Y_{\mathsf{id}}\right)$$

and

$$e\left(g_1 S_l\right) \cdot e\left(C_l, g_2\right) = g_t^{-s+s+\sum_{j=1}^{T} c_j m[j]} = e\left(\prod_{j=1}^{T} H_j^{m[j]}, g_2\right)$$

so this output is indistinguishable from the challenger's output during Game 7.

Forgery Let $(\mathcal{P}^*_{\Delta^*}, m^*, \sigma^*)$ be the forgery returned by $\mathcal{A}$. $\mathcal{S}$ follows Game 7 to compute $\hat{m}, \hat{\sigma}$. Since $(\mathcal{P}^*_{\Delta^*}, m^*, \sigma^*)$ is a successful forgery, we have $\prod_{\mathsf{id} \in \mathcal{P}^*} e\left(A^*_{\mathsf{id}}, Z_{\mathsf{id}}\right) = \prod_{i=1}^{n} h_{l_i}^{f_i^*} \cdot e\left(R^*, g_2\right) \cdot \prod_{\mathsf{id} \in \mathcal{P}^*} e\left(C^*_{\mathsf{id}}, Y_{\mathsf{id}}\right)$ as well as

$$\prod_{\mathsf{id} \in \mathcal{P}^*} e\left(\hat{A}_{\mathsf{id}}, Z_{\mathsf{id}}\right) = \prod_{i=1}^{n} h_{l_i}^{f_i^*} \cdot e\left(\hat{R}, g_2\right) \cdot \prod_{\mathsf{id} \in \mathcal{P}^*} e\left(\hat{C_{\mathsf{id}}}, Y_{\mathsf{id}}\right)$$

according to Theorem 2. We compute $A^* = \prod_{\mathsf{id} \in \mathcal{P}^*} \left(A^*_{\mathsf{id}}\right)^{\xi_{\mathsf{id},\mu}}$ as well as $\hat{A} = \prod_{\mathsf{id} \in \mathcal{P}^*} \left(\hat{A}_{\mathsf{id}}\right)^{\xi_{\mathsf{id},\mu}}$. We note that $e\left(A^*, g_2^z\right) = \prod_{\mathsf{id} \in \mathcal{P}^*} e\left(A^*_{\mathsf{id}}, Z_{\mathsf{id}}\right)$ and $e\left(\hat{A}, g_2^z\right) = \prod_{\mathsf{id} \in \mathcal{P}^*} e\left(\hat{A}_{\mathsf{id}}, Z_{\mathsf{id}}\right)$. Since we have $\mathsf{bad}_7 = \mathsf{true}$, we know that

$A^* = \hat{A}$. We compute $C^* = \prod_{\mathsf{id}\in\mathcal{P}^*} (C^*_{\mathsf{id}})^{y_{\mathsf{id}}}$ as well as $\hat{C} = \prod_{\mathsf{id}\in\mathcal{P}^*} \left(\hat{C}_{\mathsf{id}}\right)^{y_{\mathsf{id}}}$. We have $e\left(C^*, g_2\right) = \prod_{\mathsf{id}\in\mathcal{P}^*} e\left(C^*_{\mathsf{id}}, Y_{\mathsf{id}}\right)$ and $e\left(\hat{C}, g_2\right) = \prod_{\mathsf{id}\in\mathcal{P}^*} e(\hat{C}_{\mathsf{id}}, Y_{\mathsf{id}})$. By dividing the equations above and using $A^* = \hat{A}$, we obtain $e\left(\frac{\hat{C}}{C^*}, g_2^z\right) = e\left(\frac{R^*}{\hat{R}}, g_2\right)$. Setting $W = \frac{\hat{C}}{C^*}$ and $W' = \frac{R^*}{\hat{R}}$, we get a solution (W, W') to the FDHI assumption. Since $\mathsf{bad}_5 = \mathsf{false}$, we know that $C^* \neq \hat{C}$ and thus $(W, W') \neq (1, 1)$.

Lemma 9. *For every PPT adversary $\mathcal{A}$ running Game 7, there exists a PPT simulator $\mathcal{S}$ such that* $\Pr[G_7(\mathcal{A})] = \mathsf{Adv}^{FDHI}_{\mathcal{S}}(\lambda)$.

Proof. Assume we have a PPT adversary $\mathcal{A}$ that can produce a successful forgery during Game 7. We will show how a simulator $\mathcal{S}$ can use this to break the FDHI problem. Given $(g_1, g_2, g_2^z, g_2^v, g_1^{\frac{z}{v}}, g_1^r, g_1^{\frac{r}{v}})$ simulator $\mathcal{S}$ simulates Game 7.

Setup Simulator $\mathcal{S}$ chooses $c_j \in \mathbb{Z}_p$ uniformly at random for $j = 0, \ldots, T$ and sets $H_j = g_1^{c_j}$. It outputs the public parameters $\mathsf{pp} = (k, n, T, \mathsf{bgp}, H_1, \ldots, H_T, \lambda)$.

Key Generation Simulator $\mathcal{S}$ chooses an index $\mu \in [Q]$ uniformly at random. During the key generation, it chooses $a_l, b_l \in \mathbb{Z}_p$ uniformly at random for all $l \in \mathcal{L}$. It sets $h_l = g_t^{a_l} \cdot e\left(g_1, g_2^z\right)^{b_l}$. It honestly runs $(\mathsf{sk}_{sig}, \mathsf{pk}_{sig}) \leftarrow \mathsf{KeyGen}_{sig}(1^\lambda)$, chooses y_{id} uniformly at random, and sets $Y_{\mathsf{id}} = g_2^{y_{\mathsf{id}}}$ for all $\mathsf{id} \in \mathsf{ID}$. Note, that unlike Lemma 8, we do not use the element g_2^z taken from the problem instance to generate the Y_{id}. It gives the public keys $\mathsf{ek}_{\mathsf{id}} = \emptyset$, $\mathsf{vk}_{\mathsf{id}} = (\mathsf{pk}_{sig,\mathsf{id}}, h_{\mathsf{id},1}, \ldots, h_{\mathsf{id},n}, Y_{\mathsf{id}})$ to $\mathcal{A}$ for all $\mathsf{id} \in \mathsf{ID}$.

Queries Let k be a counter for the number of datasets queried by $\mathcal{A}$ (initially, it sets $k = 1$). For every new queried dataset Δ simulator $\mathcal{S}$ creates a list L_Δ of pairs (l, m), which collects all the label/message pairs queried by the adversary on Δ and the respectively generated authenticators.
Moreover, whenever the k^{th} new dataset Δ_k is queried, $\mathcal{S}$ does the following: If $k = \mu$, it samples a random $\xi_{\mathsf{id},\mu} \in \mathbb{Z}_p$, for all $\mathsf{id} \in \mathsf{ID}$ sets $Z_{\mathsf{id},\mu} = (g_2^z)^{\xi_{\mathsf{id},\mu}}$ and stores $\xi_{\mathsf{id},\mu}$. If $k \neq \mu$, it samples a random $\xi_{\mathsf{id},k} \in \mathbb{Z}_p$ and sets $Z_{\mathsf{id},k} = (g_2^v)^{\xi_{\mathsf{id},k}}$ and stores $\xi_{\mathsf{id},k}$. Since all $Z_{\mathsf{id},k}$ are randomly distributed in $\mathbb{G}_2$, they have the same distribution as in Game 7. Given a query (Δ, l, m) with $\Delta = \Delta_k$, simulator $\mathcal{S}$ first computes $\sigma_{\Delta_k,\mathsf{id}} = \mathsf{Sign}_{sig}(\mathsf{sk}_{sig,\mathsf{id}}, (\Delta_k || Z_{k,\mathsf{id}}))$.
If $k \neq \mu$, it samples $\rho_l, s_l \in \mathbb{Z}_p$ uniformly at random and computes $A_l = \left((g_1^{\frac{z}{v}})^{b_l} \cdot (g_1^{\frac{r}{v}})^{\rho_l}\right)^{\frac{1}{\xi_{\mathsf{id},k}}}$, $R_l = g_1^{-a_l - y_{\mathsf{id}} s_l} \cdot (g_1^r)^{\rho_l} \cdot g_1^{-\sum_{j=1}^{T} y_{\mathsf{id}} c_j m[j]}$, $S_l = g_2^{-s_l}$, as well as $C_l = g_1^{s_l + \sum_{j=1}^{T} c_j m[j]}$. It sets $\Lambda_l = (\mathsf{id}, \sigma_{sig,\mathsf{id}}, Z_{k,\mathsf{id}}, A_l, C_l)$ and gives (Λ_l, R_l, S_l) to $\mathcal{A}$.
We have

$$e\left(A_l, Z_{k,\mathsf{id}}\right) = e\left(\left((g_1^{\frac{z}{v}})^{b_l} \cdot (g_1^{\frac{r}{v}})^{\rho_l}\right)^{\frac{1}{\xi_{\mathsf{id},k}}}, (g_2^v)^{\xi_{\mathsf{id},k}}\right) = e\left(g_1^{z b_l + r\rho_l}, g_2\right)$$

$$= e\left(g_1^{zb_l+a_l-a_l+\sum_{j=1}^{T} y_{\mathsf{id}} c_j m[j]-\sum_{j=1}^{T} y_{\mathsf{id}} c_j m[j] s y_{\mathsf{id}} s_l - y_{\mathsf{id}} s_l + r\rho_l}, g_2\right)$$

$$= g_t^{a_l+zb_l} \cdot e\left(g_1^{-a_l} \cdot (g_1^r)^{\rho_l} \cdot g_1^{-y_{\mathsf{id}} s_l - \sum_{j=1}^{T} y_{\mathsf{id}} c_j m[j]} \cdot g_1^{s_l+\sum_{j=1}^{T} c_j m[j]}, g_2^Y\right)$$

$$= h_l \cdot e\left(R_l, g_2\right) \cdot e\left(C_l, Y_{\mathsf{id}}\right)$$

and

$$e\left(g_1, S_l\right) \cdot e\left(C_l, g_2\right) = g_t^{-s+s+\sum_{j=1}^{T} c_j m[j]} = e\left(\prod_{j=1}^{T} H_j^{m[j]}, g_2\right).$$

This output is indistinguishable from the challenger's output during Game 7.

If $k = \mu$, simulator $\mathcal{S}$ computes $A_l = \left(g_1^{b_l}\right)^{\frac{1}{\xi_{\mathsf{id},\mu}}}$, $C_l = g_1^{s_l+\sum_{j=1}^{T} y_{\mathsf{id}} c_j m[j]}$, $R_l = g_1^{-y_{\mathsf{id}} s_l - a_l - \sum_{j=1}^{T} c_j m[j]}$, as well as $S_l = g_2^{-s_l}$. It sets $\Lambda_l = (\mathsf{id}, \sigma_{sig,\mathsf{id}}, Z_{k,\mathsf{id}}, A_l, C_l)$ and gives (Λ_l, R_l, S_l) to $\mathcal{A}$.

We have

$$e\left(A_l, Z_{\mu,\mathsf{id}}\right) = e\left(\left(g_1^{b_l}\right)^{\frac{1}{\xi_{\mathsf{id},\mu}}}, (g_2^z)^{\xi_{\mathsf{id},\mu}}\right) = e\left(g_1^{zb_l}, g_2\right)$$

$$= e\left(g_1^{zb_l+a_l-a_l+\sum_{j=1}^{T} y_{\mathsf{id}} c_j m[j]-\sum_{j=1}^{T} y_{\mathsf{id}} c_j m[j]+y_{\mathsf{id}} s_l - y_{\mathsf{id}} s_l}, g_2\right)$$

$$= g_t^{a_l+zb_l} \cdot e\left(g_1^{-y_{\mathsf{id}} s_l - a_l - \sum_{j=1}^{T} y_{\mathsf{id}} c_j m[j]} \cdot g_1^{y_l s_l + \sum_{j=1}^{T} y_{\mathsf{id}} c_j m[j]}, g_2\right)$$

$$= h_l \cdot e\left(R_l, g_2\right) \cdot e\left(C_{\mathsf{id}}, Y_{\mathsf{id}}\right)$$

and

$$e\left(g_1, S_l\right) \cdot e\left(C_l, g_2\right) = g_t^{-s+s+\sum_{j=1}^{T} c_j m[j]} = e\left(\prod_{j=1}^{T} H_j^{m[j]}, g_2\right).$$

This output is indistinguishable from the challenger's output during Game 7.

Forgery Let $(\mathcal{P}^*_{\Delta^*}, m^*, \sigma^*)$ be the forgery returned by $\mathcal{A}$. $\mathcal{S}$ follows Game 7 to compute $\hat{m}, \hat{\sigma}$. Since $(\mathcal{P}^*_{\Delta^*}, m^*, \sigma^*)$ is a successful forgery, we have

$$\prod_{\mathsf{id} \in \mathcal{P}^*} e\left(A^*_{\mathsf{id}}, Z_{\mathsf{id}}\right) = \prod_{i=1}^{n} h_{l_i}^{f_i^*} \cdot e\left(R^*, g_2\right) \cdot \prod_{\mathsf{id} \in \mathcal{P}^*} e\left(C^*_{\mathsf{id}}, Y_{\mathsf{id}}\right)$$

as well as

$$\prod_{\mathsf{id} \in \mathcal{P}^*} e\left(\hat{A}_{\mathsf{id}}, Z_{\mathsf{id}}\right) = \prod_{i=1}^{n} h_{l_i}^{f_i^*} \cdot e\left(\hat{R}, g_2\right) \cdot \prod_{\mathsf{id} \in \mathcal{P}^*} e\left(\hat{C_{\mathsf{id}}}, Y_{\mathsf{id}}\right)$$

according to Theorem 2. We compute $A^* = \prod_{\mathsf{id}\in\mathcal{P}^*} (A^*_{\mathsf{id}})^{\xi_{\mathsf{id},\mu}}$ as well as $\hat{A} = \prod_{\mathsf{id}\in\mathcal{P}^*} \left(\hat{A}_{\mathsf{id}}\right)^{\xi_{\mathsf{id},\mu}}$. We note that $e\left(A^*, g_2^z\right) = \prod_{\mathsf{id}\in\mathcal{P}^*} e\left(A^*_{\mathsf{id}}, Z_{\mathsf{id}}\right)$ and $e\left(\hat{A}, g_2^z\right) = \prod_{\mathsf{id}\in\mathcal{P}^*} e\left(\hat{A}_{\mathsf{id}}, Z_{\mathsf{id}}\right)$. Since we have $\mathsf{bad}_7 = \mathsf{false}$ we know that $A^* \neq \hat{A}$. We compute $C^* = \prod_{\mathsf{id}\in\mathcal{P}^*} (C^*_{\mathsf{id}})^{y_{\mathsf{id}}}$ as well as $\hat{C} = \prod_{\mathsf{id}\in\mathcal{P}^*} \left(\hat{C}_{\mathsf{id}}\right)^{y_{\mathsf{id}}}$. We note that $e\left(C^*, g_2\right) = \prod_{\mathsf{id}\in\mathcal{P}^*} e\left(C^*_{\mathsf{id}}, Y_{\mathsf{id}}\right)$ and $e\left(\hat{C}, g_2\right) = \prod_{\mathsf{id}\in\mathcal{P}^*} e(\hat{C}_{\mathsf{id}}, Y_{\mathsf{id}})$. Dividing the equations above, we obtain $e\left(\frac{A^*}{\hat{A}}, g_2^z\right) = e\left(\frac{C^*\cdot R^*}{\hat{C}\cdot\hat{R}}, g_2\right)$ and setting $W = \frac{R^*\cdot C^*}{\hat{R}\cdot\hat{C}}$, as well as $W' = \frac{A^*}{\hat{A}}$ we have obtained a solution (W, W') to the FDHI assumption. Since we have $\mathsf{bad}_7 = \mathsf{false}$, we know that $A^* \neq \hat{A}$ and thus $(W, W') \neq (1, 1)$.

5 Related Work

We now review related work on homomorphic authenticators and verifiable computation, treating the special case of multi-key support separately for both scheme categories.

Homomorphic Authenticators Homomorphic authenticators have received substantial attention in previous work, focusing either on the public key setting, in the form of homomorphic signatures or on the private key setting, in the form of homomorphic MACs. The notion of homomorphic signatures was originally proposed by Johnson et al. [20]. The first published schemes were homomorphic only for linear functions (e.g. [2–4, 6, 7, 9, 10, 16, 22]), and found important applications in network coding and proofs of retrievability. Schemes supporting functions of higher degree also exist (e.g. [5,11]). The work by Catalano et al. [11] contains the first mechanism to verify signatures faster than the running time of the verified function. Gorbunov et al [18] have proposed the first homomorphic signature scheme that can evaluate arbitrary boolean circuits of bounded polynomial depth over signed data. However, none of the above schemes support multiple clients with different keys.

Multi-Key Homomorphic Authenticators Works considering multi-key homomorphic authenticators are more directly comparable to our scheme. Agrawal et al. [1] considered a notion of multi source signatures for network coding, and proposed a solution for linear functions. Network coding signatures are one application of homomorphic signatures, where signed data is combined to produce new signed data. Their solution allows for the usage of different keys in combining signatures, but differ slightly in their syntax and homomorphic property, as formalized in our definition of evaluation correctness (see Def. 3). Unlike this work, our scheme achieves efficient verification and is perfectly context hiding. Fiore et al. [14] have even constructed multi-key homomorphic authenticators for boolean circuits of bounded depth. While our scheme only supports linear functions, it allows the authentication of field elements, while in the case of [14] each single bit is signed

individually. Thus our authenticators are significantly smaller. Both their and our solution achieve fast amortized verification, independently of function complexity. Their solution, however, is not context hiding.

Verifiable Computation Verifiable computation also aims at detecting incorrect results in delegated computations. In this setting, a client wants to delegate the computation of a function f on input x to an untrusted server. If the server outputs y, the client's goal is to verify that indeed $y = f(x)$ at a faster runtime than an evaluation of f. For a detailed overview of this line of research, we refer to Demirel et al. [13]. Using homomorphic authenticators, clients can authenticate various (small) pieces of data independently and without storing previously outsourced data, thus allowing for incremental updates of data. In contrast, for other verifiable computation schemes, it is necessary to encode the entire input data before delegation and often such encoding can be used in a single computation only. Another advantage of homomorphic authenticators is their natural composition property. The outputs of some computations on authenticated data are already authenticated, and can be used as input for further computations.

Multi-Client Verifiable Computation Verifiable computation has also been considered in the multi-key setting [12, 19]. In these schemes the verifier is always one of the clients providing inputs to the functions, whereas our construction is publicly verifiable. Existing multi-client verifiable computation schemes also require a message from the verifier to the server, where it has to provide a certain encoding of the function f, which is not necessary for our homomorphic authenticators. Furthermore, the communication between the server and the verifier is at least linear in the total number of inputs of f, whereas in the case of succinct multi-key homomorphic authenticators the communication between the server and the verifier is proportional only to the number of clients, and depends only logarithmically on the total number of inputs. Finally, in multi-client verifiable computation, an encoding of one input can only be used in a single computation. Any input to be used in multiple computations has to be uploaded for each computation. In contrast, multi-key homomorphic authenticators allow the one-time authentication of every input and allow it to be used in an unbounded number of computations.

6 Conclusions

In this paper, we investigated the problem of constructing a context hiding publicly verifiable multi-key homomorphic authenticator scheme. We first presented two different definitions of the context hiding property in this setting, thereby distinguishing between adversaries with inside knowledge of the computation and purely external adversaries. We present the first scheme that fulfils both of these requirements. The context hiding property, both against internal and external adversaries holds in an information theoretic sense, allowing not even

computationally unbounded adversary to gain additional knowledge about the inputs.

Our authenticators are succinct, i.e. their size is independent of the number of inputs to a computation, thus keeping bandwidth low. Our verification procedure can be split into two parts, only one of which actually requires the signature to be verified. The other part can thus be precomputed, allowing for faster verification time. Regarding performance, verification time depends only on the number of identities involved (after a one time preprocessing), thus leading to efficient verification. We furthermore showed how to reduce the security of our scheme to the discrete logarithm, the decisional Diffie–Hellman and the Flexible Diffie–Hellman Inversion problems in the standard model.

In the future, we intend to investigate the viability of context hiding multi-key homomorphic authenticators for functions of higher degree. Another interesting question is whether authenticators can be constructed, whose size does not even depend on the number of identities involved.

References

1. Agrawal, S., Boneh, D., Boyen, X., Freeman, D.M.: Preventing Pollution Attacks in Multi-source Network Coding. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 161–176. Springer (2010)
2. Attrapadung, N., Libert, B.: Homomorphic Network Coding Signatures in the Standard Model. In: Catalano, D., Fazio, N., Gennaro, R., Nicolosi, A. (eds.) PKC 2011. LNCS, vol. 6571, pp. 17–34. Springer (2011)
3. Attrapadung, N., Libert, B., Peters, T.: Computing on Authenticated Data: New Privacy Definitions and Constructions. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 367–385. Springer (2012)
4. Attrapadung, N., Libert, B., Peters, T.: Efficient Completely Context-Hiding Quotable and Linearly Homomorphic Signatures. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 386–404. Springer (2013)
5. Backes, M., Fiore, D., Reischuk, R.M.: Verifiable delegation of computation on outsourced data. In: Sadeghi, A., Gligor, V.D., Yung, M. (eds.) ACM CCS 2013. pp. 863–874. ACM (2013)
6. Boneh, D., Freeman, D.M.: Linearly Homomorphic Signatures over Binary Fields and New Tools for Lattice-Based Signatures. In: Catalano, D., Fazio, N., Gennaro, R., Nicolosi, A. (eds.) PKC 2011. LNCS, vol. 6571, pp. 1–16. Springer (2011)
7. Boneh, D., Freeman, D.M., Katz, J., Waters, B.: Signing a Linear Subspace: Signature Schemes for Network Coding. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 68–87. Springer (2009)
8. Catalano, D., Fiore, D., Nizzardo, L.: Programmable Hash Functions go Private: Constructions and Applications to (Homomorphic) Signatures with Shorter Public Keys. IACR Cryptology ePrint Archive 2015, 826 (2015)
9. Catalano, D., Fiore, D., Nizzardo, L.: Programmable Hash Functions Go Private: Constructions and Applications to (Homomorphic) Signatures with Shorter Public Keys. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015, Part II. LNCS, vol. 9216, pp. 254–274. Springer (2015)
10. Catalano, D., Fiore, D., Warinschi, B.: Efficient Network Coding Signatures in the Standard Model. In: Fischlin, M., Buchmann, J.A., Manulis, M. (eds.) PKC 2012. LNCS, vol. 7293, pp. 680–696. Springer (2012)

11. Catalano, D., Fiore, D., Warinschi, B.: Homomorphic Signatures with Efficient Verification for Polynomial Functions. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 371–389. Springer (2014)
12. Choi, S.G., Katz, J., Kumaresan, R., Cid, C.: Multi-Client Non-interactive Verifiable Computation. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 499–518. Springer (2013)
13. Demirel, D., Schabhüser, L., Buchmann, J.A.: Privately and Publicly Verifiable Computing Techniques — A Survey. Springer Briefs in Computer Science, Springer (2017)
14. Fiore, D., Mitrokotsa, A., Nizzardo, L., Pagnin, E.: Multi-key Homomorphic Authenticators. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016, Part II. LNCS, vol. 10032, pp. 499–530 (2016)
15. Freeman, D.M.: Improved Security for Linearly Homomorphic Signatures: A Generic Framework. In: Fischlin, M., Buchmann, J.A., Manulis, M. (eds.) PKC 2012. LNCS, vol. 7293, pp. 697–714. Springer (2012)
16. Gennaro, R., Katz, J., Krawczyk, H., Rabin, T.: Secure Network Coding over the Integers. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 142–160. Springer (2010)
17. Goldwasser, S., Micali, S., Yao, A.C.: Strong Signature Schemes. In: Johnson, D.S., Fagin, R., Fredman, M.L., Harel, D., Karp, R.M., Lynch, N.A., Papadimitriou, C.H., Rivest, R.L., Ruzzo, W.L., Seiferas, J.I. (eds.) STOC 1983. pp. 431–439. ACM (1983)
18. Gorbunov, S., Vaikuntanathan, V., Wichs, D.: Leveled Fully Homomorphic Signatures from Standard Lattices. In: Servedio, R.A., Rubinfeld, R. (eds.) STOC 2015. pp. 469–477. ACM (2015)
19. Gordon, S.D., Katz, J., Liu, F., Shi, E., Zhou, H.: Multi-Client Verifiable Computation with Stronger Security Guarantees. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015 (2). LNCS, vol. 9015, pp. 144–168. Springer (2015)
20. Johnson, R., Molnar, D., Song, D.X., Wagner, D.A.: Homomorphic Signature Schemes. In: Preneel, B. (ed.) CT-RSA 2002. LNCS, vol. 2271, pp. 244–262. Springer (2002)
21. Libert, B., Yung, M.: Concise Mercurial Vector Commitments and Independent Zero-Knowledge Sets with Short Proofs. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 499–517. Springer (2010)
22. Schabhüser, L., Buchmann, J.A., Struck, P.: A Linearly Homomorphic Signature Scheme from Weaker Assumptions. In: O'Neill, M. (ed.) IMACC 2017. LNCS, vol. 10655, pp. 261–279. Springer (2017)