Multi-Key Homomorphic Authenticators

Dario Fiore[1], Aikaterini Mitrokotsa[2], Luca Nizzardo[1], and Elena Pagnin[2]

[1] IMDEA Software Institute, Madrid, Spain
{dario.fiore, luca.nizzardo}@imdea.org
[2] Chalmers University of Technology, Gothenburg, Sweden
{aikmitr, elenap}@chalmers.se

Abstract. Homomorphic authenticators (HAs) enable a client to authenticate a large collection of data elements $m_1, \ldots, m_t$ and outsource them, along with the corresponding authenticators, to an untrusted server. At any later point, the server can generate a *short* authenticator vouching for the correctness of the output y of a function f computed on the outsourced data, i.e., $y = f(m_1, \ldots, m_t)$. Recently researchers have focused on HAs as a solution, with minimal communication and interaction, to the problem of delegating computation on outsourced data. The notion of HAs studied so far, however, only supports executions (and proofs of correctness) of computations over data authenticated by a single user. Motivated by realistic scenarios (ubiquitous computing, sensor networks, etc.) in which large datasets include data provided by multiple users, we study the concept of *multi-key homomorphic authenticators*. In a nutshell, multi-key HAs are like HAs with the extra feature of allowing the holder of public evaluation keys to compute on data authenticated under different secret keys. In this paper, we introduce and formally define multi-key HAs. Secondly, we propose a construction of a multi-key homomorphic signature based on standard lattices and supporting the evaluation of circuits of bounded polynomial depth. Thirdly, we provide a construction of multi-key homomorphic MACs based only on pseudorandom functions and supporting the evaluation of low-degree arithmetic circuits. Albeit being less expressive and only secretly verifiable, the latter construction presents interesting efficiency properties.

1 Introduction

The technological innovations offered by modern IT systems are changing the way digital data is collected, stored, processed and consumed. As an example, think of an application where data is collected by some organizations (e.g., hospitals), stored and processed on remote servers (e.g., the Cloud) and finally consumed by other users (e.g., medical researchers) on other devices. On one hand, this computing paradigm is very attractive, particularly as data can be shared and exchanged by multiple users. On the other hand, it is evident that in such scenarios one may be concerned about security: while the users that collect and consume the data may trust each other (up to some extent), trusting the Cloud can be problematic for various reasons. More specifically, two main security concerns to be addressed are those about the *privacy* and *authenticity* of the data stored and processed in untrusted environments.

While it is widely known that privacy can be solved in such a setting using, e.g., homomorphic encryption [27], in this work we focus on the orthogonal problem of providing authenticity of data during computation. Towards this goal, our contribution is on advancing the study of *homomorphic authenticators* (HAs), a cryptographic primitive that has been the subject of recent work [32,9,26,30].

Homomorphic Authenticators. Using an homomorphic authenticator (HA) scheme a user Alice can authenticate a collection of data items $m_1, \dots, m_t$ using her secret key, and send the authenticated data to an untrusted server. The server can execute a program $\mathcal{P}$ on the authenticated data and use a public evaluation key to generate a value $\sigma_{\mathcal{P},y}$ vouching for the correctness of $y = \mathcal{P}(m_1, \dots, m_t)$. Finally, a user Bob who is given the tuple $(\mathcal{P}, y, \sigma_{\mathcal{P},y})$ and Alice's verification key can use the authenticator to verify the authenticity of y as output of the program $\mathcal{P}$ executed on data authenticated by Alice. In other words, Bob can check that the server did not tamper with the computation's result. Alice's verification key can be either secret or public. In the former case, we refer to the primitive as *homomorphic MACs* [26,11], while in the latter we refer to it as *homomorphic signatures* [9]. One of the attractive features of HAs is that the authenticator $\sigma_{\mathcal{P},y}$ is *succinct*, i.e., much shorter than $\mathcal{P}$'s input size. This means that the server can execute a program on a huge amount of data and convince Bob of its correctness by sending him only a short piece of information. As discussed in previous work (e.g., [26,5,30]), HAs provide a nice solution, with minimal communication and interaction, to the problem of delegating computations on outsourced data, and thus can be preferable to verifiable computation (more details on this comparison appear in Section 1.2).

Our Contribution: Multi-Key Homomorphic Authenticators. Up to now, the notion of HAs has inherently been single-key, i.e., homomorphic computations are allowed only on data authenticated using the same secret key. This characteristic is obviously a limitation and prevents HA schemes from suiting scenarios where the data is provided (and authenticated) by multiple users. Consider the previously mentioned example of healthcare institutions which need to compute on data collected by several hospitals or even some remote-monitored patients. Similarly, it is often required to compute statistics for time-series data collected from multiple users e.g., to monitor usage data in smart metering, clinical research or to monitor the safety of buildings. Another application scenario is in distributed networks of sensors. Imagine for instance a network of sensors where each sensor is in charge of collecting data about air pollution in a certain area of a city, it sends its data to a Cloud server, and then a central control unit asks the Cloud to compute on the data collected by the sensors (e.g., to obtain the average value of air pollution in a large area).

A trivial solution to address the problem of computing on data authenticated by multiple users is to use homomorphic authenticators in such a way that all data providers share the *same* secret authentication key. The desired functionality is obviously achieved since data would be authenticated using a single secret key. This approach however has several drawbacks. The first one is that users need to coordinate in order to agree on such a key. The second one is that in such a setting there would be no technical/legal way to differentiate between users (e.g., to make each user accountable for his/her duties) as any user can impersonate all the other ones. The third and more relevant reason is that sharing the same key exposes the overall system to way higher risks of attacks and makes disaster recovery more difficult: if a single user is compromised the whole system is compromised too, and everything has to be reinitialized from scratch.

In contrast, this paper provides an innovative solution through the notion of *multi-key homomorphic authenticators* (multi-key HAs). This primitive guarantees that the corruption of one user affects the data of that user only, but does not endanger the authenticity of computations among the other (un-corrupted) users of the system. Moreover, the proposed system is dynamic, in the sense that compromised users can be assigned new keys and be easily reintegrated.

1.1 An Overview of Our Results

Our contribution is mainly threefold. First of all, we elaborate a suitable definition of multi-key HAs. Second, we propose the first construction of a multi-key homomorphic signature (i.e., with public verifiability) which is based on standard lattices and supports the evaluation of circuits of bounded polynomial depth. Third, we present a multi-key homomorphic MAC that is based only on pseudorandom functions and supports the evaluation of low-degree arithmetic circuits. In spite of being less expressive and only secretly verifiable, this last construction is way more efficient than the signature scheme. In what follows, we elaborate more on our results.

MULTI-KEY HOMOMORPHIC AUTHENTICATORS: WHAT ARE THEY? At a high level, multi-key HAs are like HAs with the additional property that one can execute a program $\mathcal{P}$ on data authenticated using different secret keys. In multi-key HAs, Bob verifies using the verification keys of all users that provided inputs to $\mathcal{P}$. These features make multi-key HAs a perfect candidate for applications where multiple users gather and outsource data. Referring to our previous examples, using multi-key HAs each sensor can authenticate and outsource to the Cloud the data it collects; the Cloud can compute statistics on the authenticated data and provide the central control unit with the result along with a certificate vouching for its correctness.

An important aspect of our definition is a mechanism that allows the verifier to keep track of the users that authenticated the inputs of $\mathcal{P}$, i.e., to know which user contributed to each input wire of $\mathcal{P}$. To formalize this mechanism, we build on the model of *labeled data and programs* of Gennaro and Wichs [26] (we refer the reader to Section 3 for details). In terms of security, multi-key HAs allow the adversary to corrupt users (i.e., to learn their secret keys); yet this knowledge should not help the adversary in tampering with the results of programs which involve inputs of honest (i.e., uncorrupted) users only. Our model allows to handle compromised users in a similar way to what occurs with classical digital signatures: a compromised user could be banned by means of a certificate revocation, and could easily be re-integrated via a new key pair.[3] Thinking of

[3] Here we mean that this process does not add more complications than the ones already existing for classical digital signatures (e.g., relying on PKI mechanisms).

the sensor network application, if a sensor in the field gets compromised, the data provided by other sensors remains secure, and a new sensor can be easily introduced in the system with new credentials.

Finally, we require multi-key homomorphic authenticators to be *succinct* in the sense that the size of authenticators is bounded by some fixed polynomial in $(\lambda, n, \log t)$, where λ is the security parameter, n is the number of users contributing to the computation and t is the total number of inputs of $\mathcal{P}$. Although such dependence on n may look undesirable, we stress that it is still meaningful in many application scenarios where n is much smaller than t. For instance, in the application scenario of healthcare institutions a few hospitals can provide a large amount of data from patients.

A Multi-Key Homomorphic Signature for All Circuits. After setting the definition of multi-key homomorphic authenticators, we proceed to construct multi-key HA schemes. Our first contribution is a multi-key homomorphic signature that supports the evaluation of boolean circuits of depth bounded by a fixed polynomial in the security parameter. The scheme is proven secure based on the small integer solution (SIS) problem over standard lattices [36], and tolerates adversaries that corrupt users non-adaptively.[4] Our technique is inspired by the ones developed by Gorbunov, Vaikuntanathan and Wichs [30] to construct a (single-key) homomorphic signature. Our key contribution is on providing a new representation of the signatures that enables to homomorphically compute over them even if they were generated using different keys. Furthermore, our scheme enjoys an additional property, not fully satisfied by [30]: every user can authenticate separately every data item m_i of a collection $m_1, \dots m_t$, and the correctness of computations is guaranteed even when computing on not-yet-full datasets. Although it is possible to modify the scheme in [30] for signing data items separately, the security would only work against adversaries that query the whole dataset. In contrast, we prove our scheme to be secure under a stronger security definition where the adversary can *adaptively* query the various data items, and it can try to cheat by pretending to possess signatures on data items that it never queried (so-called Type 3 forgeries). We highlight that the scheme in [30] is *not* secure under the stronger definition (with Type 3 forgeries) used in this paper, and we had to introduce new techniques to deal with this scenario. This new property is particularly interesting as it enables users to authenticate and outsource data items in a streaming fashion, without ever having to store the whole dataset. This is useful in applications where the dataset size can be very large or not fixed a priori.

A Multi-Key Homomorphic MAC for Low-Degree Circuits. Our second construction is a multi-key homomorphic MAC that supports the evaluation of arithmetic circuits whose degree d is at most polynomial in the security parameter, and whose inputs come from a small number n of users. For results of such computations the corresponding authenticators have at most size $s = \binom{n+d}{d}$.[5]

[4] Precisely, our "core" scheme is secure against adversaries that make non-adaptive signing queries; this is upgraded to adaptive security via general transformations.

[5] Note that s can be bounded by $poly(n)$ for constant d, or by $poly(d)$ for constant n.

Notably, the authenticator's size is completely independent of the total number of inputs of the arithmetic circuit. Compared to our multi-key homomorphic signature, this construction is only secretly verifiable (i.e., Bob has to know the secret verification keys of all users involved in the computation) and supports a class of computations that is less expressive; also its succinctness is asymptotically worse. In spite of these drawbacks, our multi-key homomorphic MAC achieves interesting features. From the theoretical point of view, it is based on very simple cryptographic tools: a family of pseudorandom functions. Thus, the security relies only on one-way functions. On the practical side, it is particularly efficient: generating a MAC requires only one pseudo-random function evaluation and a couple of field operations; homomorphic computations boil down to additions and multiplications over a multi-variate polynomial ring $\mathbb{F}_p[X_1, \ldots, X_n]$.

1.2 Related Work

Homomorphic MACs and Signatures. Homomorphic authenticators have received a lot of attention in previous work focusing either on homomorphic signatures (publicly verifiable) or on homomorphic MACs (private verification with a secret key). The notion of homomorphic signatures was originally proposed by Johnson *et al.* [32]. The first schemes that appeared in the literature were homomorphic only for linear functions [8,14,15,23,13] and found important applications in network coding and proofs of retrievability. Boneh and Freeman [9] were the first to construct homomorphic signatures that can evaluate more than linear functions over signed data. Their scheme could evaluate bounded-degree polynomials and its security was based on the hardness of the SIS problem in ideal lattices in the random oracle model. A few years later, Catalano *et al.* [16] proposed an alternative homomorphic signature scheme for bounded-degree polynomials. Their solution is based on multi-linear maps and bypasses the need for random oracles. More interestingly, the work by Catalano *et al.* [16] contains the first mechanism to verify signatures faster than the running time of the verified function. Recently, Gorbunov *et al.* [30] have proposed the first (leveled) fully homomorphic signature scheme that can evaluate arbitrary circuits of bounded polynomial depth over signed data. Some important advances have been also achieved in the area of homomorphic MACs. Gennaro and Wichs [26] have proposed a fully homomorphic MAC based on fully homomorphic encryption. However, their scheme is not secure in the presence of verification queries. More efficient schemes have been proposed later [11,5,12] that are secure in the presence of verification queries and are more efficient at the price of supporting only restricted homomorphisms. Finally, we note that Agrawal *et al.* [1] considered a notion of *multi-key* signatures for network coding, and proposed a solution which works for linear functions only. Compared to this work, our contribution shows a full-fledged framework for multi-key homomorphic authenticators, and provides solutions that address a more expressive class of computations.

Verifiable Computation. Achieving correctness of outsourced computations is also the aim of *verifiable delegation of computation* (VC) [29,25,18,6,37,20]. In this setting, a client wants to delegate the computation of a function f on input x to an untrusted cloud-server. If the server replies with y, the client's goal is to verify the correctness of $y = f(x)$ spending less resources than those needed to execute f. As mentioned in previous work (e.g., [26,30]) a crucial difference between verifiable computation and homomorphic authenticators is that in VC the verifier has to know the input of the computation – which can be huge – whereas in HAs one can verify by only knowing the function f and the result y. Moreover, although some results of verifiable computation could be re-interpreted to solve scenarios similar to the ones addressed by HAs, results based on VC would still present several limitations. For instance, using homomorphic authenticators the server can prove correctness of $y = f(x)$ with a single message, without needing any special encoding of f from the delegator. Second, HAs come naturally with a composition property which means that the outputs of some computations on authenticated data (which is already authenticated) can be fed as input for follow-up computations. This feature is of particular interest to parallelize and or distribute computations (e.g., MapReduce). Emulating this composition within VC systems is possible by means of certain non-interactive proof systems [7] but leads to complex statements and less natural realizations. A last advantage is that by using HAs, clients can authenticate various (small) pieces of data independently and without storing previously outsourced data. In contrast, most VC systems require clients to encode the whole input in 'one shot', and often such encoding can be used in a single computation only.

Multi-Client Verifiable Computation. Another line of work, closely related to ours is that on *multi-client verifiable computation* [17,31]. This primitive, introduced by Choi *et al.* [17], aims to extend VC to the setting where inputs are provided by multiple users, and one of these users wants to verify the result's correctness. Choi *et al.* [17] proposed a definition and a multi-client VC scheme which generalizes that of Gennaro *et al.* [25]. The solution in [17], however, does not consider malicious or colluding clients. This setting was addressed by Gordon *et al.* in [31], where they provide a scheme with stronger security guarantees against a malicious server or an arbitrary set of malicious colluding clients.

It is interesting to notice that in the definition of multi-client VC all the clients but the one who verifies can encode inputs independently of the function to be later executed on them. One may thus think that the special case in which the verifier provides no input yields a solution similar to the one achieved by multi-key HAs. However, a closer look at the definitions and the existing constructions of multi-client VC reveals three main differences. (1) In multi-client VC, in order to prove the correctness of an execution of a function f, the server has to wait a message from the verifier which includes some encoding of f. This is not necessary in multi-key HAs where the server can directly prove the correctness of f on previously authenticated data with a single message and without any function's encoding. (2) The communication between the server and the verifier is at least linear in the total number of inputs of f: this can be prohibitive in the case of computations on very large inputs (think of TBytes of data). In contrast, with multi-key HAs the communication between the server and the verifier is proportional only to the number of users, and depends only logarithmically on the total number of inputs. (3) In multi-client VC an encoding of one input can be used in a single computation. Thus, if a user wants to first upload data on the server to later execute many functions on it, then the user has to provide as many encodings as the number of functions to be executed. In contrast, multi-key HAs allow one to encode (i.e., authenticate) every input only once and to use it for proving correctness of computations an *unbounded* number of times.

2 Preliminaries

We collect here the notation and basic definitions used throughout the paper.

Notation. The Greek letter λ is reserved for the security parameter of the schemes. A function $\epsilon(\lambda)$ is said to be *negligible* in λ (denoted as $\epsilon(\lambda) = \mathsf{negl}(\lambda)$) if $\epsilon(\lambda) = O(\lambda^{-c})$ for every constant $c > 0$. When a function can be expressed as a polynomial we often write it as $poly(\cdot)$. For any $n \in \mathbb{N}$, we refer to $[n]$ as $[n] := \{1, \ldots, n\}$. Moreover, given a set $\mathcal{S}$, the notation $s \xleftarrow{\$} \mathcal{S}$ stays for the process of sampling s uniformly at random from $\mathcal{S}$.

Definition 1 (Statistical Distance). *Let X, Y denote two random variables with support $\mathcal{X}, \mathcal{Y}$ respectively; the statistical distance between X and Y is defined as $\mathbf{SD}(X, Y) := \frac{1}{2}(\sum_{u \in \mathcal{X} \cup \mathcal{Y}} \mid \Pr[X = u] - \Pr[Y = u] \mid)$. If $\mathbf{SD}(X, Y) = \mathsf{negl}(\lambda)$, we say that X and Y are statistically close and we write $X \overset{\mathsf{stat}}{\approx} Y$.*

Definition 2 (Entropy [19]). *The min-entropy of a random variable X is defined as $\mathbf{H}_\infty(X) := -\mathsf{log}\big(\max_x \Pr[X = x]\big)$. The (average-) conditional min-entropy of a random variable X conditioned on a correlated variable Y is defined as $\mathbf{H}_\infty(X \mid Y) := -\mathsf{log}\Big(\underset{y \leftarrow Y}{\mathbf{E}}\Big[\max_x \Pr[X = x \mid Y = y]\Big]\Big)$. The optimal probability of an unbounded adversary guessing X when given a correlated value Y is $2^{-\mathbf{H}_\infty(X|Y)}$.*

Lemma 1 ([19]). *Let X, Y be arbitrarily random variables where the support of Y lies in $\mathcal{Y}$. Then $\mathbf{H}_\infty(X \mid Y) \geq \mathbf{H}_\infty(X) - \mathsf{log}(\mid \mathcal{Y} \mid)$.*

3 Multi-Key Homomorphic Authenticators

In this section, we present our new notion of *Multi-Key Homomorphic Authenticators* (multi-key HAs). Intuitively, multi-key HAs extend the existing notions of homomorphic signatures [9] and homomorphic MACs [26] in such a way that one can homomorphically compute a program $\mathcal{P}$ over data authenticated using different secret keys. For the sake of verification, in multi-key HAs the verifier needs to know the verification keys of all users that provided inputs to $\mathcal{P}$. Our definitions are meant to be general enough to be easily adapted to both the case in which verification keys are public and the one where verification keys are secret. In the former case, we call the primitive *multi-key homomorphic signatures* whereas in the latter case we call it *multi-key homomorphic MACs*.

As already observed in previous work about HAs, it is important that an authenticator $\sigma_{\mathcal{P},y}$ does not authenticate a value y out of context, but only as the output of a program $\mathcal{P}$ executed on previously authenticated data. To formalize this notion, we build on the model of *labeled data and programs* of Gennaro and Wichs [26]. The idea of this model is that every data item is authenticated under a unique label ℓ. For example, in scenarios where the data is outsourced, such labels can be thought of as a way to index/identify the remotely stored data. A labeled program $\mathcal{P}$, on the other hand, consists of a circuit f where every input wire i has a label ℓ_i. Going back to the outsourcing example, a labeled program is a way to specify on what portion of the outsourced data one should execute a circuit f. More formally, the notion of labeled programs of [26] is recalled below.

Labeled Programs [26]. A labeled program $\mathcal{P}$ is a tuple $(f, \ell_1, \ldots, \ell_n)$, such that $f : \mathcal{M}^n \to \mathcal{M}$ is a function of n variables (e.g., a circuit) and $\ell_i \in \{0,1\}^*$ is a label for the i-th input of f. Labeled programs can be composed as follows: given $\mathcal{P}_1, \ldots, \mathcal{P}_t$ and a function $g : \mathcal{M}^t \to \mathcal{M}$, the composed program $\mathcal{P}^*$ is the one obtained by evaluating g on the outputs of $\mathcal{P}_1, \ldots, \mathcal{P}_t$, and it is denoted as $\mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_t)$. The labeled inputs of $\mathcal{P}^*$ are all the distinct labeled inputs of $\mathcal{P}_1, \ldots \mathcal{P}_t$ (all the inputs with the same label are grouped together and considered as a unique input of $\mathcal{P}^*$). Let $f_{id} : \mathcal{M} \to \mathcal{M}$ be the identity function and $\ell \in \{0,1\}^*$ be any label. We refer to $\mathcal{I}_\ell = (f_{id}, \ell)$ as the identity program with label ℓ. Note that a program $\mathcal{P} = (f, \ell_1, \ldots, \ell_n)$ can be expressed as the composition of n identity programs $\mathcal{P} = f(\mathcal{I}_{\ell_1}, \ldots, \mathcal{I}_{\ell_n})$.

Using labeled programs to identify users. In our notion of multi-key homomorphic authenticators, one wishes to verify the outputs of computations executed on data authenticated under *different* keys. A meaningful definition of multi-key HAs thus requires that the authenticators are not out of context also with respect to the set of keys that contributed to the computation. To address this issue, we assume that every user has an identity id in some identity space ID, and that the user's keys are associated to id by means of any suitable mechanism (e.g., PKI). Next, in order to distinguish among data of different users and to identify to which input wires a certain user contributed, we assume that the label space contains the set ID. Namely, in our model a data item is assigned a label $\ell := (\mathsf{id}, \tau)$, where id is a user's identity, and τ is a tag; this essentially identifies uniquely a data item of user id with index τ. For compatibility with previous notions of homomorphic authenticators, we assume that data items can be grouped in datasets, and one can compute only on data within the same dataset. In our definitions, a dataset is identified by an arbitrary string Δ.[6]

Definition 3 (Multi-Key Homomorphic Authenticator). *A multi-key homomorphic authenticator scheme* MKHAut *consists of a tuple of PPT algorithms* (Setup, KeyGen, Auth, Eval, Ver) *satisfying the following properties:* authentication correctness, evaluation correctness, succinctness *and* security. *The five algorithms work as follows:*

$\mathsf{Setup}(1^\lambda)$. *The setup algorithm takes as input the security parameter λ and outputs some public parameters* pp. *These parameters include (at least) descriptions of a tag space $\mathcal{T}$, an identity space* ID*, the message space $\mathcal{M}$ and a set of admissible functions $\mathcal{F}$. Given $\mathcal{T}$ and* ID*, the label space of the scheme*

[6] Although considering the dataset notion complicates the definition, it also provides some benefits, as we illustrate later in the constructions. For instance, when verifying for the same program $\mathcal{P}$ over different datasets, one can perform some precomputation that makes further verifications cheap.

is defined as their cartesian product $\mathcal{L} := \mathsf{ID} \times \mathcal{T}$. For a labeled program $\mathcal{P} = (f, \ell_1, \ldots, \ell_t)$ with labels $\ell_i := (\mathsf{id}_i, \tau_i) \in \mathcal{L}$, we use $\mathsf{id} \in \mathcal{P}$ as compact notation for $\mathsf{id} \in \{\mathsf{id}_1, \ldots, \mathsf{id}_t\}$. The pp *are input to all the following algorithms, even when not specified.*

$\mathsf{KeyGen}(\mathsf{pp})$. *The key generation algorithm takes as input the public parameters and outputs a triple of keys* $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk})$, *where* sk *is a secret authentication key,* ek *is a public evaluation key and* vk *is a verification key which could be either public or private.*[7]

$\mathsf{Auth}(\mathsf{sk}, \Delta, \ell, \mathsf{m})$. *The authentication algorithm takes as input an authentication key* sk*, a dataset identifier Δ, a label $\ell = (\mathsf{id}, \tau)$ for the message* m*, and it outputs an authenticator σ.*

$\mathsf{Eval}(f, \{(\sigma_i, \mathsf{EKS}_i)\}_{i\in[t]})$. *The evaluation algorithm takes as input a t-input function $f : \mathcal{M}^t \longrightarrow \mathcal{M}$, and a set $\{(\sigma_i, \mathsf{EKS}_i)\}_{i\in[t]}$ where each σ_i is an authenticator and each EKS_i is a set of evaluation keys.*[8]

$\mathsf{Ver}(\mathcal{P}, \Delta, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}}, \mathsf{m}, \sigma)$. *The verification algorithm takes as input a labeled program $\mathcal{P} = (f, \ell_1, \ldots, \ell_t)$, a dataset identifier Δ, the set of verification keys $\{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}}$ corresponding to those identities* id *involved in the program $\mathcal{P}$, a message* m *and an authenticator σ. It outputs* 0 *(reject) or* 1 *(accept).*

Authentication Correctness. Intuitively, a Multi-Key Homomorphic Autenticator has authentication correctness if the output of $\mathsf{Auth}(\mathsf{sk}, \Delta, \ell, \mathsf{m})$ verifies correctly for m as the output of the identity program $\mathcal{I}_\ell$ over the dataset Δ. More formally, a scheme MKHAut satisfies authentication correctness if for all 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 $\ell = (\mathsf{id}, \tau) \in \mathcal{L}$ and any authenticator $\sigma \leftarrow \mathsf{Auth}(\mathsf{sk}, \Delta, \ell, \mathsf{m})$, we have $\mathsf{Ver}(\mathcal{I}_\ell, \Delta, \mathsf{vk}, \mathsf{m}, \sigma)$ outputs 1 with all but negligible probability.

Evaluation Correctness. Intuitively, this property says that running the evaluation algorithm on signatures $(\sigma_1, \ldots, \sigma_t)$ such that each σ_i verifies for m_i as the output of a labeled program $\mathcal{P}_i$ over the dataset Δ, it produces a signature σ which verifies for $f(\mathsf{m}_1, \ldots, \mathsf{m}_t)$ as the output of the composed program $f(\mathcal{P}_1, \ldots, \mathcal{P}_t)$ over the dataset Δ. More formally, let us fix the public parameters $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, a set of key triples $\{(\mathsf{sk}_{\mathsf{id}}, \mathsf{ek}_{\mathsf{id}}, \mathsf{vk}_{\mathsf{id}})\}_{\mathsf{id}\in\tilde{\mathsf{ID}}}$ for some $\tilde{\mathsf{ID}} \subseteq \mathsf{ID}$, a dataset Δ, a function $g : \mathcal{M}^t \rightarrow \mathcal{M}$, and any set of program/message/authenticator triples $\{(\mathcal{P}_i, m_i, \sigma_i)\}_{i\in[t]}$ such that $\mathsf{Ver}(\mathcal{P}_i, \Delta, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}_i}, \mathsf{m}_i, \sigma_i) = 1$ for all $i \in [t]$. Let $\mathsf{m}^* = g(\mathsf{m}_1, \ldots, \mathsf{m}_t)$, $\mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_t)$, and $\sigma^* = \mathsf{Eval}(g, \{(\sigma_i, \mathsf{EKS}_i)\}_{i\in[t]})$ where $\mathsf{EKS}_i = \{\mathsf{ek}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}_i}$. Then, $\mathsf{Ver}(\mathcal{P}^*, \Delta, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}^*}, \mathsf{m}^*, \sigma^*) = 1$ holds with all but negligible probability.

Succinctness. A multi-key HA is said to be *succinct* if the size of every authenticator depends only logarithmically on the size of a dataset. However, we

[7] As mentioned earlier, the generated triple $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk})$ will be associated to an identity $\mathsf{id} \in \mathsf{ID}$. When this connection becomes explicit, we will refer to $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk})$ as $(\mathsf{sk}_{\mathsf{id}}, \mathsf{ek}_{\mathsf{id}}, \mathsf{vk}_{\mathsf{id}})$.

[8] The motivation behind the evaluation-keys set EKS_i is that, if σ_i authenticates the output of a labeled program $\mathcal{P}_i$, then $\mathsf{EKS}_i = \{\mathsf{ek}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}_i}$ should be the set of evaluation keys corresponding to identities involved in the computation of $\mathcal{P}_i$.

allow authenticators to depend on the number of keys involved in the computation. More formally, let $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, $\mathcal{P} = (f, \ell_1, \ldots, \ell_t)$ with $\ell_i = (\mathsf{id}_i, \tau_i)$, $\{(\mathsf{sk}_{\mathsf{id}}, \mathsf{ek}_{\mathsf{id}}, \mathsf{vk}_{\mathsf{id}}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})\}_{\mathsf{id} \in \mathcal{P}}$, and $\sigma_i \leftarrow \mathsf{Auth}(\mathsf{sk}_{\mathsf{id}_i}, \Delta, \ell_i, \mathsf{m}_i)$ for all $i \in [t]$. A multi-key HA is said to be *succinct* if there exists a fixed polynomial p such that $|\sigma| = p(\lambda, n, \log t)$ where $\sigma = \mathsf{Eval}(g, \{(\sigma_i, \mathsf{ek}_{\mathsf{id}_i})\}_{i \in [t]})$ and $n = |\{\mathsf{id} \in \mathcal{P}\}|$.

Remark 1. Succinctness is one of the crucial properties that make multi-key HAs an interesting primitive. Without succinctness, a trivial multi-key HA construction is the one where Eval outputs the concatenation of all the signatures (and messages) given in input, and the verifier simply checks each message-signature pair and recomputes the function by himself. Our definition of succinctness, where signatures can grow linearly with the number of keys but logarithmically in the total number of inputs, is also non-trivial, especially when considering settings where there are many more inputs than keys (in which case, the above trivial construction would not work). Another property that can make homomorphic signatures an interesting primitive is privacy—context-hiding—as considered in prior work. Intuitively, context-hiding guarantees that signatures do not reveal information on the original inputs. While we leave the study of context-hiding for multi-key HAs for future work, we note that a trivial construction that is context-hiding but not succinct can be easily obtained with the additional help of non-interactive zero-knowledge proofs of knowledge: the idea is to extend the trivial construction above by requiring the evaluator to generate a NIZK proof of knowledge of valid input messages and signatures that yield the public output. In this sense, we believe that succinctness is the most non-trivial property to achieve in homomorphic signatures, and this is what we focus on in this work.

SECURITY. Intuitively, our security model for multi-key HAs guarantees that an adversary, without knowledge of the secret keys, can only produce authenticators that were either received from a legitimate user, or verify correctly on the results of computations executed on the data authenticated by legitimate users. Moreover, we also give to the adversary the possibility of corrupting users. In this case, it must not be able to cheat on the outputs of programs that get inputs from uncorrupted users only. In other words, our security definition guarantees that the corruption of one user affects the data of that user only, but does not endanger the integrity of computations among the other (un-corrupted) users of the system. We point out that preventing cheating on programs that involve inputs of corrupted users is inherently impossible in multi-key HAs, at least if general functions are considered. For instance, consider an adversary who picks the function $(x_1 + x_2 \mod p)$ where x_1 is supposed to be provided by user Alice. If the adversary corrupts Alice, it can use her secret key to inject any input authenticated on her behalf and thus bias the output of the function at its will.

The formalization of the intuitions illustrated above is more involved. For a scheme MKHAut we define security via the following experiment between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$ ($\mathsf{HomUF\text{-}CMA}_{\mathcal{A},\mathsf{MKHAut}}(\lambda)$):

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 $(\Delta, \ell, \mathsf{m})$, where Δ is a dataset identifier, $\ell = (\mathsf{id}, \tau)$ is a label in $\mathsf{ID} \times \mathcal{T}$ and $\mathsf{m} \in \mathcal{M}$ are messages of his choice. $\mathcal{C}$ answers as follows:

- If $(\Delta, \ell, \mathsf{m})$ is the first query for the dataset Δ, $\mathcal{C}$ initializes an empty list $L_\Delta = \emptyset$ and proceeds as follows.
- If $(\Delta, \ell, \mathsf{m})$ 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 $\mathsf{ek}_{\mathsf{id}}$ to $\mathcal{A}$ and proceeds as follows.
- If $(\Delta, \ell, \mathsf{m})$ is such that $(\ell, \mathsf{m}) \notin L_\Delta$, $\mathcal{C}$ computes $\sigma_\ell \xleftarrow{\$} \mathsf{Auth}(\mathsf{sk}_{\mathsf{id}}, \Delta, \ell, \mathsf{m})$ (note that $\mathcal{C}$ has already generated keys for the identity id), returns σ_ℓ to $\mathcal{A}$, and updates the list $L_\Delta \leftarrow L_\Delta \cup (\ell, \mathsf{m})$.
- If $(\Delta, \ell, \mathsf{m})$ is such that $(\ell, \cdot) \in L_\Delta$ (which means that the adversary had already made a query $(\Delta, \ell, \mathsf{m}')$ for some message m'), 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 $(\mathcal{P}, \Delta, \mathsf{m}, \sigma)$, and $\mathcal{C}$ replies with the output of $\mathsf{Ver}(\mathcal{P}, \Delta, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}}, \mathsf{m}, \sigma)$.

Corruption The adversary has access to a corruption oracle. At the beginning of the game, the challenger initialises 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 $(\mathcal{P}^*, \Delta^*, \mathsf{m}^*, \sigma^*)$. The experiment outputs 1 if the tuple returned by $\mathcal{A}$ is a forgery (defined below), and 0 otherwise.

Definition 4 (Forgery). *Consider an execution of* $\mathsf{HomUF\text{-}CMA}_{\mathcal{A},\mathsf{MKHAut}}(\lambda)$ *where* $(\mathcal{P}^*, \Delta^*, \mathsf{m}^*, \sigma^*)$ *is the tuple returned by the adversary in the end of the experiment, with* $\mathcal{P}^* = (f^*, \ell_1^*, \dots, \ell_t^*)$*. This is a forgery if* $\mathsf{Ver}(\mathcal{P}^*, \Delta^*, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}^*}, \mathsf{m}^*, \sigma^*) = 1$*, for all* $\mathsf{id} \in \mathcal{P}^*$ *we have that* $\mathsf{id} \notin L_{\mathsf{corr}}$ *(i.e., no identity involved in* $\mathcal{P}^*$ *is corrupted), and either one of the following properties is satisfied:*

Type 1: L_{Δ^*} *has not been initialized during the game (i.e., the dataset* Δ^* *was never queried).*

Type 2: *For all* $i \in [t]$*,* $\exists (\ell_i^*, \mathsf{m}_i) \in L_{\Delta^*}$*, but* $\mathsf{m}^* \neq f^*(\mathsf{m}_1, \dots, \mathsf{m}_t)$ *(i.e.,* m^* *is not the correct output of* $\mathcal{P}^*$ *when executed over previously authenticated messages).*

Type 3: *There exists a label* ℓ^* *such that* $(\ell^*, \cdot) \notin L_{\Delta^*}$ *(i.e.,* $\mathcal{A}$ *never made a query with label* ℓ^**).*

We say that a HA scheme MKHAut is *secure* if for every PPT adversary $\mathcal{A}$, its advantage $\mathbf{Adv}^{\mathsf{HomUF\text{-}CMA}}_{\mathsf{MKHAut},\mathcal{A}}(\lambda) = \Pr[\mathsf{HomUF\text{-}CMA}_{\mathcal{A},\mathsf{MKHAut}}(\lambda) = 1]$ is negligible.

Remark 2 (Comparison with previous security definitions). Our security notion can be seen as the multi-key version of the one proposed by Gennaro and Wichs in [26] (in their model our Type 3 forgeries are called ‘Type 1’ as they do not consider multiple datasets). We point out that even in the special case of a single key, our security definition is stronger than the ones used in previous work [9,23,16,30] (with the only exception of [26]). The main difference lies in our definition of Type 3 forgeries. The intuitive idea of this kind of forgeries is that an adversary who did not receive an authenticated input labeled by a certain ℓ^* cannot produce a valid authenticator for the output of a computation which has ℓ^* among its inputs. In [9,30] these forgeries were not considered at all, as the adversary is assumed to query the dataset always in full. Other works [23,11,16] consider a weaker Type 3 notion, which deals with the concept of "well defined programs", where the input wire labeled by the missing label ℓ^* is required to "contribute" to the computation (i.e., it must change its outcome). The issue with such a Type 3 definition is that it may not be efficient to test if an input contributes to a function, especially if the admissible functions are general circuits. In contrast our definition above is simpler and efficiently testable since it simply considers a Type 3 forgery one where the labeled program $\mathcal{P}^*$ involves an un-queried input.

Multi-Key Homomorphic Signatures. As previously mentioned, our definitions are general enough to be easily adapted to either case in which verification is secret or public. The only difference is whether the adversary is allowed to see the verification keys in the security experiment. When the verification is public, we call the primitive *multi-key homomorphic signatures.* More formally:

Definition 5 (Multi-Key Homomorphic Signatures). *A multi-key homomorphic signature is a multi-key homomorphic authenticator in which verification keys are also given to the adversary in the security experiment.*

Note that making verification keys public also allows to slightly simplify the security experiment by removing the verification oracle (the adversary can run the verification by itself). In the sequel, when referring to multi-key homomorphic signatures we adapt our notation to the typical one of digital signatures, namely we denote $\mathsf{Auth}(\mathsf{sk}, \Delta, \ell, \mathsf{m})$ as $\mathsf{Sign}(\mathsf{sk}, \Delta, \ell, \mathsf{m})$, and call its outputs *signatures.*

Non-Adaptive Corruption Queries. In our work, we consider a relaxation of the security 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, it is easy to see that corruption queries are essentially of no help as the adversary can generate keys on its own. More precisely, the following proposition holds (see the full version [22] for the proof).

Proposition 1. MKHAut *is secure against adversaries that do* not *make corruption queries if and only if* MKHAut *is secure against adversaries that make* non-adaptive *corruption queries.*

Weakly-Adaptive Secure multi-key HAs. In our work, we also consider a weaker notion of security for multi-key HAs in which the adversary has to declare all the queried messages at the beginning of the experiment. More precisely, we consider a notion in which the adversary declares only the messages and the respective tags that will be queried, for every dataset and identity, without, however, needing to specify the names of the datasets or of the identities. In a sense, the adversary $\mathcal{A}$ is adaptive on identities and dataset names, but not on tags and messages. The definition is inspired by the one, for the single-key setting, of Catalano *et al.* [16].

To define the notion of weakly-adaptive security for multi-key HAs, we introduce here a new experiment $\mathsf{Weak\text{-}HomUF\text{-}CMA}_{\mathcal{A},\mathsf{MKHAut}}$, which is a variation of experiment $\mathsf{HomUF\text{-}CMA}_{\mathcal{A},\mathsf{MKHAut}}$ (Definition 3) as described below.

Definition 6 (Weakly-Secure Multi-Key Homomorphic Authenticators). *In the security experiment $\mathsf{Weak\text{-}HomUF\text{-}CMA}_{\mathcal{A},\mathsf{MKHAut}}$, before the setup phase, the adversary $\mathcal{A}$ sends to the challenger $\mathcal{C}$ a collection of sets of tags $\mathcal{T}_{i,k} \subseteq \mathcal{T}$ for $i \in [Q_{\mathsf{id}}]$ and $k \in [Q_\Delta]$, where Q_{id} and Q_Δ are, respectively, the total numbers of distinct identities and datasets that will be queried during the game. Associated to every set $\mathcal{T}_{i,k}$, $\mathcal{A}$ also sends a set of messages $\{m_\tau\}_{\tau \in \mathcal{T}_{i,k}}$. Basically the adversary declares, prior to key generation, all the messages and tags that it will query later on; however $\mathcal{A}$ is not required to specify identity and dataset names. Next, the adversary receives the public parameters from $\mathcal{C}$ and can start the query-phase. Verification queries are handled as in $\mathsf{HomUF\text{-}CMA}_{\mathcal{A},\mathsf{MKHAut}}$. For authentication queries, $\mathcal{A}$ can adaptively submit pairs (id, Δ) to $\mathcal{C}$. The challenger then replies with a set of authenticators $\{\sigma_\tau\}_{\tau \in \mathcal{T}_{i,k}}$, where indices i, k are such that id is the i-th queried identity, and Δ is the k-th queried dataset.*

An analogous security definition of weakly-secure multi-key homomorphic signatures is trivially obtained by removing a verification oracle.

In the full version of this paper, we present two generic transformations that turn weakly secure multi-key homomorphic authenticator schemes into adaptive secure ones. Our first transformation holds in the standard model and works for schemes in which the tag space $\mathcal{T}$ has polynomial size, while the second one avoids this limitation on the size of $\mathcal{T}$ but holds in the random oracle model.

4 Our Multi-Key Fully Homomorphic Signature

In this section, we present our construction of a multi-key homomorphic signature scheme that supports the evaluation of arbitrary circuits of bounded polynomial depth. The scheme is based on the *SIS* problem on standard lattices, a background of which is provided in the next section. Precisely, in Section 4.2 we present a scheme that is weakly-secure and supports a single dataset. Later, in Section 4.3 we discuss how to extend the scheme to handle multiple datasets, whereas the support of adaptive security can be obtained via the applications of our transformations as shown in [22].

4.1 Lattices and Small Integer Solution Problem

We recall here notation and some basic results about lattices that are useful to describe our homomorphic signature construction.

For any positive integer q we denote by $\mathbb{Z}_q$ the ring of integers modulo q. Elements in $\mathbb{Z}_q$ are represented as integers in the range $(-\frac{q}{2}, \frac{q}{2}]$. The absolute value of any $x \in \mathbb{Z}_q$ (denoted with $|x|$) is defined by taking the modulus q representative of x in $(-\frac{q}{2}, \frac{q}{2}]$, i.e., take $y = x \mod q$ and then set $|x| = |y| \in [0, \frac{q}{2}]$. Vectors and matrices are denoted in bold. For any vector $\mathbf{u} := (u_i, \dots, u_n) \in \mathbb{Z}_q^n$, its infinity norm is $\|\mathbf{u}\|_\infty := \max_{i \in [n]} |u_i|$, and similarly for a matrix $\mathbf{A} := [a_{i,j}] \in \mathbb{Z}_q^{n \times m}$ we write $\|\mathbf{A}\|_\infty := \max_{i \in [n], j \in [m]} |a_{i,j}|$.

The Small Integer Solution Problem (SIS). For integer parameters n, m, q and β, the $\mathsf{SIS}(n, m, q, \beta)$ problem provides to an adversary $\mathcal{A}$ a uniformly random matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, and requires $\mathcal{A}$ to find a vector $\mathbf{u} \in \mathbb{Z}_q^n$ such that $\mathbf{u} \neq \mathbf{0}$, $\|\mathbf{u}\|_\infty \leq \beta$, and $\mathbf{A} \cdot \mathbf{u} = \mathbf{0}$. More formally,

Definition 7 (SIS [36]). *Let $\lambda \in \mathbb{N}$ be the security parameter. For values $n = n(\lambda), m = m(\lambda), q = q(\lambda), \beta = \beta(\lambda)$, defined as functions of λ, the $\mathsf{SIS}(n, m, q, \beta)$ hardness assumption holds if for any PPT adversary $\mathcal{A}$ we have*

$$\Pr\left[\mathbf{A} \cdot \mathbf{u} = \mathbf{0} \ \wedge \ \mathbf{u} \neq \mathbf{0} \ \wedge \ \|\mathbf{u}\|_\infty \leq \beta : \mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}, \mathbf{u} \leftarrow \mathcal{A}(1^\lambda, \mathbf{A})\right] \leq \mathsf{negl}(\lambda).$$

For standard lattices, the *SIS* problem is known to be as hard as solving certain worst-case instances of lattice problems [2,33,36,35], and is also implied by the hardness of learning with error (we refer any interested reader to the cited papers for the technical details about the parameters).

In our paper, we assume that for any $\beta = 2^{\mathsf{poly}(\lambda)}$ there are some $n = \mathsf{poly}(\lambda)$, $q = 2^{\mathsf{poly}(\lambda)}$, with $q > \beta$, such that for all $m = \mathsf{poly}(\lambda)$ the $\mathsf{SIS}(n, m, q, \beta)$ hardness assumption holds. This parameters choice assures that hardness of worst-case lattice problems holds with sub-exponential approximation factors.

Trapdoors for Lattices. The *SIS* problem is hard to solve for a random matrix $\mathbf{A}$. However, there is a way to sample a random $\mathbf{A}$ together with a *trapdoor* such that *SIS* becomes easy to solve for that $\mathbf{A}$, given the trapdoor. Additionally, it has been shown that there exist "special" (non random) matrices $\mathbf{G}$ for which *SIS* is easy to solve as well. The following lemma summarizes the above known results (similar to a lemma in [10]):

Lemma 2 ([3,28,4,34]). *There exist efficient algorithms $\mathsf{TrapGen}, \mathsf{SamPre}, \mathsf{Sam}$ such that the following holds: given integers $n \geq 1$, $q \geq 2$, there exist some $m^* = m^*(n, q) = O(n \log q)$, $\beta_{\mathsf{sam}} = \beta_{\mathsf{sam}}(n, q) = O(n\sqrt{\log q})$ such that for all $m \geq m^*$ and all k (polynomial in n) we have:*

1. *$\mathsf{Sam}(1^m, 1^k, q) \to \mathbf{U}$ samples a matrix $\mathbf{U} \in \mathbb{Z}_q^{m\times k}$ such that $\|\mathbf{U}\|_\infty \leq \beta_{\mathsf{sam}}$ (with probability 1).*
2. *For $(\mathbf{A}, \mathsf{td}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q)$, $\mathbf{A}' \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$, $\mathbf{U} \leftarrow \mathsf{Sam}(1^m, 1^k, q)$, $\mathbf{V} := \mathbf{AU}$, $\mathbf{V}' \xleftarrow{\$} \mathbb{Z}_q^{n\times k}$, $\mathbf{U}' \leftarrow \mathsf{SamPre}(\mathbf{A}, \mathbf{V}', \mathsf{td})$, we have the following statistical indistinguishability (negligible in n)*
$$\mathbf{A} \overset{\mathsf{stat}}{\approx} \mathbf{A}' \quad \textit{and} \quad (\mathbf{A}, \mathsf{td}, \mathbf{U}, \mathbf{V}) \overset{\mathsf{stat}}{\approx} (\mathbf{A}, \mathsf{td}, \mathbf{U}', \mathbf{V}')$$
and $\mathbf{U}' \leftarrow \mathsf{SamPre}(\mathbf{A}, \mathbf{V}', \mathsf{td})$ always satisfies $\mathbf{AU}' = \mathbf{V}'$ and $\|\mathbf{U}'\|_\infty \leq \beta_{\mathsf{sam}}$.
3. *Given n, m, q as above, there is an efficiently and deterministically computable matrix $\mathbf{G} \in \mathbb{Z}_q^{n\times m}$ and a deterministic polynomial-time algorithm $\mathbf{G}^{-1}$ that on input $\mathbf{V} \in \mathbb{Z}_q^{n\times k}$ (for any integer k) outputs $\mathbf{R} = \mathbf{G}^{-1}(\mathbf{V})$ such that $\mathbf{R} \in \{0, 1\}^{m\times k}$ and $\mathbf{GR} = \mathbf{V}$.*

4.2 Our Multi-Key Homomorphic Signature for Single Dataset

In this section, we present our multi-key homomorphic signature that supports the evaluation of boolean circuits of bounded polynomial depth. Our construction is inspired by the (single-key) one of Gorbunov *et al.* [30], with the fundamental difference that in our case we enable computations over data signed using different secret keys. Moreover, our scheme is secure against Type 3 forgeries. We achieve this via a new technique which consists into adding to every signature a component that specifically protects against this type of forgeries. We prove the scheme to be weakly-secure under the SIS hardness assumption.

Parameters. Before describing the scheme, we discuss how to set the various parameters involved. Let λ be the security parameter, and let $d = d(\lambda) = \mathsf{poly}(\lambda)$ be the bound on the depth of the circuits supported by our scheme. We define the set of parameters used in our scheme $\mathsf{Par} = \{n, m, q, \beta_{\mathsf{SIS}}, \beta_{\mathsf{max}}, \beta_{\mathsf{init}}\}$ in terms of λ, d and of the parameters required by the trapdoor algorithm in Lemma 2: $m^*, \beta_{\mathsf{sam}}$, where $m^* = m^*(n, q) := O(n \log q)$ and $\beta_{\mathsf{sam}} := O(n\sqrt{\log q})$. More precisely, we set: $\beta_{\mathsf{max}} := 2^{\omega(\log \lambda)d}$; $\beta_{\mathsf{SIS}} := 2^{\omega(\log \lambda)}\beta_{\mathsf{max}}$; $n = \mathsf{poly}(\lambda)$; $q = \mathcal{O}(2^{\mathsf{poly}(\lambda)}) > \beta_{\mathsf{SIS}}$ is a prime (as small as possible) so that the $\mathsf{SIS}(n, m', q, \beta_{\mathsf{SIS}})$ assumption holds for all $m' = \mathsf{poly}(\lambda)$; $m = \max\{m^*, n \log q + \omega(\log(\lambda))\} = \mathsf{poly}(\lambda)$ and, finally, $\beta_{\mathsf{init}} := \beta_{\mathsf{sam}} = \mathsf{poly}(\lambda)$.

Construction. The PPT algorithms ($\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Eval}, \mathsf{Ver}$) which define our construction of Multi-key Homomorphic Signatures work as follows:

$\mathsf{Setup}(1^\lambda)$**.** The setup algorithm takes as input the security parameter λ and generates the public parameters pp which include: the bound on the circuit depth d (which defines the class $\mathcal{F}$ of functions supported by the scheme, i.e., boolean circuits of depth d), the set $\mathsf{Par} = \{n, m, q, \beta_{\mathsf{SIS}}, \beta_{\mathsf{max}}, \beta_{\mathsf{init}}\}$, the set $\mathcal{U} = \{\mathbf{U} \in \mathbb{Z}_q^{m \times m} : \|\mathbf{U}\|_\infty \leq \beta_{\mathsf{max}}\}$, the set $\mathcal{V} = \{\mathbf{V} \in \mathbb{Z}_q^{n \times m}\}$, descriptions of the message space $\mathcal{M} = \{0, 1\}$, the tag space $\mathcal{T} = [\mathsf{T}]$, and the identity space $\mathsf{ID} = [\mathsf{C}]$, for integers $\mathsf{T}, \mathsf{C} \in \mathbb{N}$. In this construction, the tag space is of polynomial size, i.e., $\mathsf{T} = \mathsf{poly}(\lambda)$ while the identity space is essentially unbounded, i.e., we set $\mathsf{C} = 2^\lambda$. Also recall that $\mathcal{T}$ and ID immediately define the label space $\mathcal{L} = \mathsf{ID} \times \mathcal{T}$. The final output is $\mathsf{pp} = \{d, \mathsf{Par}, \mathcal{U}, \mathcal{V}, \mathcal{M}, \mathcal{T}, \mathsf{ID}\}$. We assume that these public parameters pp are input of all subsequent algorithms, and often omit them from the input explicitly.

$\mathsf{KeyGen}(\mathsf{pp})$**.** The key generation algorithm takes as input the public parameters pp and generates a key-triple $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk})$ defined as follows. First, it samples T random matrices $\mathbf{V}_1, \ldots, \mathbf{V}_\mathsf{T} \xleftarrow{\$} \mathcal{V}$. Second, it runs $(\mathbf{A}, \mathsf{td}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q)$ to generate a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ along with its trapdoor td. Then, it outputs $\mathsf{sk} = (\mathsf{td}, \mathbf{A}, \mathbf{V}_1, \ldots, \mathbf{V}_\mathsf{T})$, $\mathsf{ek} = \mathbf{A}$, $\mathsf{vk} = (\mathbf{A}, \mathbf{V}_1, \ldots, \mathbf{V}_\mathsf{T})$. Note that it is possible to associate the key-triple to an identity $\mathsf{id} \in \mathsf{ID}$, when we need to stress this explicitly we write $(\mathsf{sk}_{\mathsf{id}}, \mathsf{ek}_{\mathsf{id}}, \mathsf{vk}_{\mathsf{id}})$. We also observe that the key generation process can be seen as the combination of two independent sub-algorithms [9] KeyGen_1 and KeyGen_2, where $\{\mathbf{V}_1, \ldots \mathbf{V}_\mathsf{T}\} \leftarrow \mathsf{KeyGen}_1(\mathsf{pp})$ and $(\mathbf{A}, \mathsf{td}) \leftarrow \mathsf{KeyGen}_2(\mathsf{pp})$.

$\mathsf{Sign}(\mathsf{sk}, \ell, \mathsf{m})$**.** The signing algorithm takes as input a secret key sk, a label $\ell = (\mathsf{id}, \tau)$ for the message m and it outputs a signature $\sigma := (\mathsf{m}, \mathsf{z}, \mathsf{I}, \mathbf{U}_{\mathsf{id}}, \mathbf{Z}_{\mathsf{id}})$

[9] This splitting will be used to extend our multi-key homomorphic signature scheme from supporting a single dataset to support multiple datasets. This extension holds in the standard model and is described in Section 4.3.

where $\mathsf{I} = \{\mathsf{id}\}$, $\mathbf{U}_{\mathsf{id}}$ is generated as $\mathbf{U}_{\mathsf{id}} \leftarrow \mathsf{SamPre}(\mathbf{A}, \mathbf{V}_\ell - \mathsf{m}\mathbf{G}, \mathsf{td})$ (using the algorithm SamPre from Lemma 2), $\mathsf{z} = \mathsf{m}$ and $\mathbf{Z}_{\mathsf{id}} = \mathbf{U}_{\mathsf{id}}$. The two latter terms are responsible for protection against Type 3 forgeries. Although they are redundant for fresh signatures, their value will become different from $(\mathsf{m}, \mathbf{U}_{\mathsf{id}})$ during homomorphic operations, as we clarify later on. More generally, in our construction signatures are of the form $\sigma := (\mathsf{m}, \mathsf{z}, \mathsf{I}, \{\mathbf{U}_{\mathsf{id}}\}_{\mathsf{id}\in\mathsf{I}}, \{\mathbf{Z}_{\mathsf{id}}\}_{\mathsf{id}\in\mathsf{I}})$ with $\mathsf{I} \subseteq \mathsf{ID}$ and $\mathbf{U}_{\mathsf{id}}, \mathbf{Z}_{\mathsf{id}} \in \mathcal{U}, \forall\, \mathsf{id} \in \mathsf{I}$.

$\mathsf{Eval}\big(f, \{(\sigma_i, \mathsf{EKS}_i)\}_{i\in[t]}\big)$**.** The evaluation algorithm takes as input a t-input function $f : \mathcal{M}^t \longrightarrow \mathcal{M}$, and a set of pairs $\{(\sigma_i, \mathsf{EKS}_i)\}_{i\in[t]}$ where each σ_i is a signature and each EKS_i is a set of evaluation keys. In our description below we treat f as an arithmetic circuit over $\mathbb{Z}_q$ consisting of addition and multiplication gates.[10] Therefore, we only describe how to evaluate homomorphically a fan-in-2 addition (resp. multiplication) gate as well as a unary multiplication-by-constant gate.

Let g be a fan-in-2 gate with left input $\sigma_{\mathsf{L}} := (\mathsf{m}_{\mathsf{L}}, \mathsf{z}_{\mathsf{L}}, \mathsf{I}_{\mathsf{L}}, \mathbf{U}_{\mathsf{L}}, \mathbf{Z}_{\mathsf{L}})$ and right input $\sigma_{\mathsf{R}} := (\mathsf{m}_{\mathsf{R}}, \mathsf{z}_{\mathsf{R}}, \mathsf{I}_{\mathsf{R}}, \mathbf{U}_{\mathsf{R}}, \mathbf{Z}_{\mathsf{R}})$. To generate the signature $\sigma := (\mathsf{m}, \mathsf{z}, \mathsf{I}, \mathbf{U}, \mathbf{Z})$ on the gate's output one proceeds as follows. First set $\mathsf{I} = \mathsf{I}_{\mathsf{L}} \cup \mathsf{I}_{\mathsf{R}}$. Second, "expand" $\mathbf{U}_{\mathsf{L}} := \{\mathbf{U}_{\mathsf{L}}^{\mathsf{id}}\}_{\mathsf{id}\in\mathsf{I}_{\mathsf{L}}}$ as:

$$\hat{\mathbf{U}}_{\mathsf{L}}^{\mathsf{id}} = \begin{cases} \mathbf{0} & \text{if } \mathsf{id} \notin \mathsf{I}_{\mathsf{L}} \\ \mathbf{U}_{\mathsf{L}}^{\mathsf{id}} & \text{if } \mathsf{id} \in \mathsf{I}_{\mathsf{L}} \end{cases}, \quad \forall\, \mathsf{id} \in \mathsf{I}\ .$$

where $\mathbf{0}$ denotes an $(m\times m)$-matrix with all zero entries. Basically, we extend the set to be indexed over all identities in $\mathsf{I} = \mathsf{I}_{\mathsf{L}} \cup \mathsf{I}_{\mathsf{R}}$ by inserting zero matrices for identities in $\mathsf{I} \setminus \mathsf{I}_{\mathsf{L}}$. The analogous expansion process is applied to $\mathbf{U}_{\mathsf{R}} := \{\mathbf{U}_{\mathsf{R}}^{\mathsf{id}}\}_{\mathsf{id}\in\mathsf{I}_{\mathsf{L}}}$, $\mathbf{Z}_{\mathsf{L}} := \{\mathbf{Z}_{\mathsf{L}}{}^{\mathsf{id}}\}_{\mathsf{id}\in\mathsf{I}_{\mathsf{R}}}$ and $\mathbf{Z}_{\mathsf{R}} := \{\mathbf{Z}_{\mathsf{R}}^{\mathsf{id}}\}_{\mathsf{id}\in\mathsf{I}_{\mathsf{R}}}$, denoting the expanded sets $\{\hat{\mathbf{U}}_{\mathsf{R}}^{\mathsf{id}}\}_{\mathsf{id}\in\mathsf{I}}$, $\{\hat{\mathbf{Z}}_{\mathsf{L}}^{\mathsf{id}}\}_{\mathsf{id}\in\mathsf{I}}$ and $\{\hat{\mathbf{Z}}_{\mathsf{R}}^{\mathsf{id}}\}_{\mathsf{id}\in\mathsf{I}}$ respectively.

Next, depending on whether g is an addition or multiplication gate one proceeds as follows.

ADDITION GATE. If g is additive, compute $\mathsf{m} = \mathsf{m}_{\mathsf{L}} + \mathsf{m}_{\mathsf{R}}$, $\mathsf{z} = \mathsf{z}_{\mathsf{L}} + \mathsf{z}_{\mathsf{R}}$, $\mathbf{U} = \{\mathbf{U}_{\mathsf{id}}\}_{\mathsf{id}\in\mathsf{I}} := \{\hat{\mathbf{U}}_{\mathsf{L}}^{\mathsf{id}} + \hat{\mathbf{U}}_{\mathsf{R}}^{\mathsf{id}}\}_{\mathsf{id}\in\mathsf{I}}$ and $\mathbf{Z} = \{\mathbf{Z}_{\mathsf{id}}\}_{\mathsf{id}\in\mathsf{I}} := \{\hat{\mathbf{Z}}_{\mathsf{L}}^{\mathsf{id}} + \hat{\mathbf{Z}}_{\mathsf{R}}^{\mathsf{id}}\}_{\mathsf{id}\in\mathsf{I}}$.

If we refer to β_{L} and β_{R} as $\|\mathbf{U}_{\mathsf{L}}\|_\infty := \max\{\|\mathbf{U}_{\mathsf{L}}^{\mathsf{id}}\|_\infty : \mathsf{id} \in \mathsf{I}_{\mathsf{L}}\}$ and $\|\mathbf{U}_{\mathsf{R}}\|_\infty := \max\{\|\mathbf{U}_{\mathsf{R}}^{\mathsf{id}}\|_\infty : \mathsf{id} \in \mathsf{I}_{\mathsf{R}}\}$ respectively, then for any fan-in-2 addition gate it holds $\beta := \|\mathbf{U}\|_\infty = \beta_{\mathsf{L}} + \beta_{\mathsf{R}}$. The same noise growth applies to $\mathbf{Z}$.

MULTIPLICATION GATE. If g is multiplicative, compute $\mathsf{m} = \mathsf{m}_{\mathsf{L}} \cdot \mathsf{m}_{\mathsf{R}}$, $\mathsf{z} = \mathsf{z}_{\mathsf{L}} + \mathsf{z}_{\mathsf{R}}$, define $\mathbf{V}_{\mathsf{L}} = \sum_{\mathsf{id}\in\,\mathsf{I}_{\mathsf{L}}} \mathbf{A}_{\mathsf{id}}\mathbf{U}_{\mathsf{id}} + \mathsf{m}_{\mathsf{L}}\mathbf{G}$, set

$$\mathbf{U} = \{\mathbf{U}_{\mathsf{id}}\}_{\mathsf{id}\in\mathsf{I}} := \{\mathsf{m}_{\mathsf{R}}\hat{\mathbf{U}}_{\mathsf{L}}^{\mathsf{id}} + \hat{\mathbf{U}}_{\mathsf{R}}^{\mathsf{id}} \cdot \mathbf{G}^{-1}(\mathbf{V}_{\mathsf{L}})\}_{\mathsf{id}\in\mathsf{I}}$$

and $\mathbf{Z} = \{\mathbf{Z}_{\mathsf{id}}\}_{\mathsf{id}\in\mathsf{I}} := \{\hat{\mathbf{Z}}_{\mathsf{L}}^{\mathsf{id}} + \hat{\mathbf{Z}}_{\mathsf{R}}^{\mathsf{id}}\}_{\mathsf{id}\in\mathsf{I}}$.

Letting β_{L} and β_{R} as defined before, then for any fan-in-2 multiplication gate it holds $\beta := \|\mathbf{U}\|_\infty = |\mathsf{m}_{\mathsf{R}}|\beta_{\mathsf{L}} + m\beta_{\mathsf{R}}$, while the noise growth of $\mathbf{Z}$ is the same as in the addition gate.

[10] We point out that considering f as an arithmetic circuit over $\mathbb{Z}_q$ is enough to describe any boolean circuits consisting of NAND gates as $\mathsf{NAND}(\mathsf{m}_1, \mathsf{m}_2) = 1 - \mathsf{m}_1 \cdot \mathsf{m}_2$ holds for $\mathsf{m}_1, \mathsf{m}_2 \in \{0, 1\}$.

Multiplication by constant gate. Let g be a unary gate representing a multiplication by a constant $a \in \mathbb{Z}_q$, and let its single input signature be $\sigma_{\mathsf{R}} := (\mathsf{m}_{\mathsf{R}}, \mathsf{z}_{\mathsf{R}}, \mathsf{I}_{\mathsf{R}}, \mathbf{U}_{\mathsf{R}}, \mathbf{Z}_{\mathsf{R}})$. The output $\sigma := (\mathsf{m}, \mathsf{z}, \mathsf{I}, \mathbf{U}, \mathbf{Z})$ is obtained by setting $\mathsf{m} = a \cdot \mathsf{m}_{\mathsf{R}} \in \mathbb{Z}_q$, $\mathsf{z} = \mathsf{z}_{\mathsf{R}}$, $\mathsf{I} = \mathsf{I}_{\mathsf{R}}$, $\mathbf{Z} = \mathbf{Z}_{\mathsf{R}}$, and $\mathbf{U} = \{\mathbf{U}^{\mathsf{id}}\}_{\mathsf{id} \in \mathsf{I}}$ where, for all $\mathsf{id} \in \mathsf{I}$, $\mathbf{U}^{\mathsf{id}} = a \cdot \mathbf{U}^{\mathsf{id}}_{\mathsf{R}}$ or, alternatively, $\mathbf{U}^{\mathsf{id}} = \mathbf{U}^{\mathsf{id}}_{\mathsf{R}} \cdot \mathbf{G}^{-1}(a \cdot \mathbf{G})$. In the first case, the noise parameter becomes $\beta := \|\mathbf{U}\|_\infty = |a|\beta_{\mathsf{L}}$ (thus a needs to be *small*), whereas in the second case it holds $\beta := \|\mathbf{U}\|_\infty \leq m\beta_{\mathsf{L}}$, which is independent of a's size.

$\mathsf{Ver}(\mathcal{P}, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}}, \mathsf{m}, \sigma)$. The verification algorithm takes as input a labeled program $\mathcal{P} = (f, \ell_1, \ldots, \ell_n)$, the set of the verification keys $\{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}}$ of users involved in the program $\mathcal{P}$, a message m and a signature $\sigma = (\mathsf{m}, \mathsf{z}, \mathsf{I}, \mathbf{U}, \mathbf{Z})$. It then performs three main checks and outputs 0 if at least one check fails, otherwise it returns 1.

Firstly, it checks if the list of identities declared in σ corresponds to the ones in the labels of $\mathcal{P}$:

$$\mathsf{I} = \{\mathsf{id} : \mathsf{id} \in \mathcal{P}\} \tag{1}$$

Secondly, from the circuit f (again seen as an arithmetic circuit) and the values $\{\mathbf{V}_{\ell_1}, \ldots, \mathbf{V}_{\ell_t}\}$ contained in the verification keys, it computes two values $\mathbf{V}^*$ and $\mathbf{V}^+$ proceeding gate by gate as follows. Given as left and right input matrices $\mathbf{V}^*_{\mathsf{L}}, \mathbf{V}^*_{\mathsf{R}}$ (resp. $\mathbf{V}^+_{\mathsf{L}}, \mathbf{V}^+_{\mathsf{R}}$), at every *addition gate* one computes $\mathbf{V}^* = \mathbf{V}^*_{\mathsf{L}} + \mathbf{V}^*_{\mathsf{R}}$ (resp. $\mathbf{V}^+ = \mathbf{V}^+_{\mathsf{L}} + \mathbf{V}^+_{\mathsf{R}}$); at every *multiplication gate* one computes $\mathbf{V}^* = \mathbf{V}^*_{\mathsf{R}}\mathbf{G}^{-1}\mathbf{V}^*_{\mathsf{L}}$ (resp. $\mathbf{V}^+ = \mathbf{V}^+_{\mathsf{L}} + \mathbf{V}^+_{\mathsf{R}}$). Every gate representing a multiplication by a constant $a \in \mathbb{Z}_q$, on input $\mathbf{V}^*_{\mathsf{R}}$ (resp. $\mathbf{V}^+_{\mathsf{R}}$) outputs $\mathbf{V}^* = a \cdot \mathbf{V}^*_{\mathsf{R}}$ (resp. $\mathbf{V}^+ = \mathbf{V}^+_{\mathsf{R}}$). Note that the computation of $\mathbf{V}^+$ is essentially the computation of a linear function $\mathbf{V}^+ = \sum_{i=1}^t \gamma_i \cdot \mathbf{V}_{\ell_i}$, for some coefficients γ_i that depend on the structure of the circuit f.

Thirdly, the verification algorithm parses $\mathbf{U} = \{\mathbf{U}_{\mathsf{id}}\}_{\mathsf{id} \in \mathsf{I}}$ and $\mathbf{Z} = \{\mathbf{Z}_{\mathsf{id}}\}_{\mathsf{id} \in \mathsf{I}}$ and checks:

$$\|\mathbf{U}\|_\infty \leq \beta_{\mathsf{max}} \quad \text{and} \quad \|\mathbf{Z}\|_\infty \leq \beta_{\mathsf{max}} \tag{2}$$

$$\sum_{\mathsf{id} \in \mathsf{I}} \mathbf{A}_{\mathsf{id}} \mathbf{U}_{\mathsf{id}} + \mathsf{m} \cdot \mathbf{G} = \mathbf{V}^* \tag{3}$$

$$\sum_{\mathsf{id} \in \mathsf{I}} \mathbf{A}_{\mathsf{id}} \mathbf{Z}_{\mathsf{id}} + \mathsf{z} \cdot \mathbf{G} = \mathbf{V}^+ \tag{4}$$

Finally, it is worth noting that the computation of the matrices $\mathbf{V}^*$ and $\mathbf{V}^+$ can be precomputed (or performed offline), prior to seeing the actual signature σ. In the multiple dataset extension of Section 4.3 this precomputation becomes particularly beneficial as the same $\mathbf{V}^*, \mathbf{V}^+$ can be re-used every time one wants to verify for the same labeled program $\mathcal{P}$ (i.e., one can verify faster, in an amortized sense, than that of running f).

In the following paragraphs, we analyse succinctness and security of the proposed construction; for what regards correctness we just give an intuition and refer the interested reader to the full version of this paper available in [22].

Noise Growth and Succinctness. First we analyse the noise growth of the components $\mathbf{U}, \mathbf{Z}$ in the signatures of our MKHSig construction. In particular we need to show that when starting from "fresh" signatures, in which the noise is bounded by β_{init}, and we apply an admissible circuit, then one ends up with signatures in which the noise is within the allowable amount β_{max}.

An analysis similar to the one of Gorbunov *et al.* [30] is applicable also to our construction whenever the admissible functions are boolean circuits of depth d composed only of NAND gates.

Let us first consider the case of the $\mathbf{U}$ component of the signatures. At every NAND gate, if $\|\mathbf{U}_{\mathsf{L}}\|_\infty, \|\mathbf{U}_{\mathsf{R}}\|_\infty \leq \beta$, the noise of the resulting $\mathbf{U}$ is at most $(m+1)\beta$. Therefore, if the circuit has depth d, the noise of the matrix $\mathbf{U}$ at the end of the evaluation is bounded by $\|\mathbf{U}\|_\infty \leq \beta_{\mathsf{init}} \cdot (m+1)^d \leq 2^{O(\log\lambda)d} \leq \beta_{\mathsf{max}}$. For what regards the computation performed over the matrices $\mathbf{Z}$, we observe that we perform only additions (or identity functions) over them. This means that at every gate of any f, the noise in the $\mathbf{Z}$ component at most doubles. Given that we consider depth-d circuits we have that $\|\mathbf{Z}\|_\infty \leq \beta_{\mathsf{init}} \cdot 2^d \leq 2^{O(\log\lambda)+d} \leq \beta_{\mathsf{max}}$. Finally, by inspection one can see that the size of every signature σ on a computation's output involving n users is at most $(1 + 2^d + n\lambda + 2n\beta_{\mathsf{max}})$ that is $O(n \cdot p(\lambda))$ for some fixed polynomial $p(\cdot)$.

Authentication Correctness. This is rather simple and follows from the noise growth property mentioned above and by observing that equation $\mathbf{A}_{\mathsf{id}}\mathbf{U}_{\mathsf{id}} + \mathsf{m}\mathbf{G} = \mathbf{V}_\ell = \mathbf{V}^*$ holds by construction.

Evaluation Correctness. Evaluation correctness follows from two main facts: the noise growth mentioned earlier, and the preservation of the invariant $\sum_{\mathsf{id}\in\mathsf{I}} \mathbf{A}_{\mathsf{id}}\mathbf{U}_{\mathsf{id}} + \mathsf{m}\mathbf{G} = \mathbf{V}^*$. At every gate, it is easy to see that the expansion of $\mathbf{U}$ still preserves the invariant for both left and right inputs. For additive gates, assuming validity of the inputs, i.e., $\mathbf{V}_{\mathsf{L}} = \sum_{\mathsf{id}\in\mathsf{I}_{\mathsf{L}}} \mathbf{A}_{\mathsf{id}}\mathbf{U}_{\mathsf{L}}^{\mathsf{id}} + \mathsf{m}_{\mathsf{L}}\mathbf{G}$ (and similarly $\mathbf{V}_{\mathsf{R}}$) and by construction of $\mathbf{U}_{\mathsf{id}} = \mathbf{U}_{\mathsf{L}}^{\mathsf{id}} + \mathbf{U}_{\mathsf{R}}^{\mathsf{id}}$, one obtains $\sum_{\mathsf{id}\in\mathsf{I}_{\mathsf{L}}\cup\mathsf{I}_{\mathsf{R}}} \mathbf{A}_{\mathsf{id}}\mathbf{U}_{\mathsf{id}} + (\mathsf{m}_{\mathsf{L}} + \mathsf{m}_{\mathsf{R}})\mathbf{G} = \mathbf{V}_{\mathsf{L}} + \mathbf{V}_{\mathsf{R}} = \mathbf{V}^*$. For what regards multiplicative gates, by construction of every $\mathbf{U}_{\mathsf{id}}$ we obtain $\sum_{\mathsf{id}\in\mathsf{I}} \mathbf{A}_{\mathsf{id}}\mathbf{U}_{\mathsf{id}} + \mathsf{m}\mathbf{G} := \sum_{\mathsf{id}\in\mathsf{I}} \mathbf{A}_{\mathsf{id}}(\mathsf{m}_{\mathsf{R}}\hat{\mathbf{U}}_{\mathsf{L}}^{\mathsf{id}} + \hat{\mathbf{U}}_{\mathsf{R}}^{\mathsf{id}}\mathbf{G}^{-1}(\mathbf{V}_{\mathsf{L}})) + (\mathsf{m}_{\mathsf{L}}\mathsf{m}_{\mathsf{R}})\mathbf{G}$. Grouping by m_{R} and applying the definition of $\mathbf{V}_{\mathsf{L}}$, the equation can be rewritten as $\mathsf{m}_{\mathsf{R}}\mathbf{V}_{\mathsf{L}} + \left(\sum_{\mathsf{id}\in\mathsf{I}} \mathbf{A}_{\mathsf{id}} \ \hat{\mathbf{U}}_{\mathsf{R}}^{\mathsf{id}}\right) \mathbf{G}^{-1}(\mathbf{V}_{\mathsf{L}})$. If now we write $\mathsf{m}_{\mathsf{R}}\mathbf{V}_{\mathsf{L}}$ as $\mathsf{m}_{\mathsf{R}}\mathbf{G}\mathbf{G}^{-1}(\mathbf{V}_{\mathsf{L}})$ and we group by $\mathbf{G}^{-1}(\mathbf{V}_{\mathsf{L}})$, we get $\left[\left(\sum_{\mathsf{id}\in\mathsf{I}_{\mathsf{R}}} \mathbf{A}_{\mathsf{id}}\mathbf{U}_{\mathsf{R}}^{\mathsf{id}}\right) + \mathsf{m}_{\mathsf{R}}\mathbf{G}\right] \mathbf{G}^{-1}(\mathbf{V}_{\mathsf{L}}) = \mathbf{V}^*$, where the last equation follows from the definitions of $\mathbf{V}_{\mathsf{R}}$ and $\mathbf{V}^*$. Correctness of computations over the matrices $\mathbf{Z}$ is quite analogous.

Security. The following theorem states the security of the scheme MKHSig.

Theorem 1. *If the* $\mathsf{SIS}(n, m \cdot Q_{\mathsf{id}}, q, \beta_{\mathsf{SIS}})$ *hardness assumption holds,* $\mathsf{MKHSig} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Eval}, \mathsf{Ver})$ *is a multi-key homomorphic signature weakly-adaptive secure against adversaries that make signing queries involving at most* Q_{id} *different identities and that make non-adaptive corruption queries.*

Proof. Note that we can deal with corruptions via our generic result of Proposition 1. Therefore it is sufficient to prove the security against adversaries that make no corruptions. Moreover, since this scheme works for a single dataset note that Type 1 forgeries cannot occur.

For the proof let us recall how the weakly-adaptive security experiment (Definition 6) works for our multi-key homomorphic signature scheme MKHSig. This is a game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ that has four main phases:

(1) $\mathcal{A}$ declares an integer Q representing the number of different identities that it will ask in the signing queries. Moreover, for every $i \in [Q]$ $\mathcal{A}$ sends to $\mathcal{C}$ a set $\mathcal{T}_i \subseteq \mathcal{T} := \{\tau_1, \ldots, \tau_{\mathsf{T}}\}$ and a set of pairs $\{(\mathsf{m}_\tau, \tau)\}_{\tau \in \mathcal{T}_i}$.
(2) $\mathcal{C}$ runs $\mathsf{Setup}(1^\lambda)$ to obtain the public parameters and sends them to $\mathcal{A}$.
(3) $\mathcal{A}$ adaptively queries identities $\mathsf{id}_1, \ldots, \mathsf{id}_Q$. When $\mathcal{C}$ receives the query id_i it generates a key-triple $(\mathsf{sk}_{\mathsf{id}_i}, \mathsf{ek}_{\mathsf{id}_i}, \mathsf{vk}_{\mathsf{id}_i})$ by running $\mathsf{KeyGen}(\mathsf{pp})$, and for all labels $\ell = (\mathsf{id}_i, \tau)$ such that $\tau \in \mathcal{T}_i$ it runs $\sigma^i_\tau \leftarrow \mathsf{Sign}(\mathsf{sk}_{\mathsf{id}_i}, \ell, \mathsf{m}_\tau)$. Then $\mathcal{C}$ sends to $\mathcal{A}$: the public keys $\mathsf{vk}_{\mathsf{id}_i} := (\mathbf{A}_{\mathsf{id}_i}, \{\mathbf{V}_\ell\}_{\tau \in \mathcal{T}})$ and $\mathsf{ek}_{\mathsf{id}_i} := (\mathbf{A}_{\mathsf{id}_i})$, and the signatures $\{\sigma^i_\tau\}_{\tau \in \mathcal{T}_i}$.
(4) The adversary produces a forgery consisting of a labeled program $\mathcal{P}^* = (f^*, \ell^*_1, \ldots, \ell^*_t)$ where $f^* \in \mathcal{F}, f^* : \mathcal{M}^t \to \mathcal{M}$, a message m^* and a signature σ^*.

$\mathcal{A}$ wins the non-adaptive security game if $\mathsf{Ver}(\mathcal{P}^*, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}^*}, \mathsf{m}^*, \sigma^*) = 1$ and one of the following conditions holds:

Type 2 Forgery: there exist messages $\mathsf{m}_{\ell^*_1}, \ldots, \mathsf{m}_{\ell^*_t}$ s.t. $\mathsf{m}^* \neq f^*(\mathsf{m}_{\ell^*_1}, \ldots, \mathsf{m}_{\ell^*_t})$ (i.e., m^* is not the correct output of $\mathcal{P}^*$ when executed over previously signed messages).

Type 3 Forgery: there exists at least one label $\ell^* = (\mathsf{id}^*, \tau^*)$ that was not queried by $\mathcal{A}$.

Consider a variation of the above game obtained modifying phase (3) as follows: $(3')$ $\mathcal{C}$ picks an instance $\mathbf{A} \in \mathbb{Z}_q^{n \times m'}$ of the $\mathsf{SIS}(n, m', q, \beta_{\mathsf{SIS}})$ problem for $m' = m \cdot Q = \mathsf{poly}(\lambda)$, and parse $\mathbf{A} := (\mathbf{A}_{\mathsf{id}_1} | \ldots | \mathbf{A}_{\mathsf{id}_Q}) \in \mathbb{Z}_q^{n \times m'}$ as the concatenation of Q different blocks of $n \times m$ matrices.
Next, when $\mathcal{C}$ receives the i-th query id_i from $\mathcal{A}$, it does the following:

– it samples a matrix $\mathbf{U}_{\mathsf{id}_i,\tau} \xleftarrow{\$} \mathcal{U}$ such that $\|\mathbf{U}_{\mathsf{id}_i,\tau}\| \leq \beta_{\mathsf{init}}$;
– for all $\ell := (\mathsf{id}_i, \tau)$ with $\tau \in \mathcal{T}_i$, $\mathcal{C}$ computes $\mathbf{V}_\ell = \mathbf{A}_{\mathsf{id}_i}\mathbf{U}_{\mathsf{id}_i,\tau} + \mathsf{m}_\tau \cdot \mathbf{G}$;
– for all $\ell := (\mathsf{id}_i, \tau)$ with $\tau \notin \mathcal{T}_i$, $\mathcal{C}$ computes $\mathbf{V}_\ell = \mathbf{A}_{\mathsf{id}_i}\mathbf{U}_{\mathsf{id}_i,\tau} + b_{i,\tau} \cdot \mathbf{G}$, where $b_{i,\tau} \xleftarrow{\$} \{0,1\}$.
– $\mathcal{C}$ sends to $\mathcal{A}$ the public keys $\mathsf{vk}_{\mathsf{id}_i} := (\mathbf{A}_{\mathsf{id}_i}, \{\mathbf{V}_\ell\}_{\tau \in \mathcal{T}})$ and $\mathsf{ek}_{\mathsf{id}_i} := (\mathbf{A}_{\mathsf{id}_i})$, along with signatures $\{\sigma^i_\tau\}_{\tau \in \mathcal{T}_i}$ where $\sigma^i_\tau := (\mathsf{m}_\tau, \mathsf{m}_\tau, \mathsf{I} := \{\mathsf{id}_i\}, \mathbf{U}_{\mathsf{id}_i,\tau}, \mathbf{U}_{\mathsf{id}_i,\tau})$.

Clearly, if $\mathbf{A}$ is a uniformly random matrix so is each block $\{\mathbf{A}_{\mathsf{id}_i}\}_{i \in [Q]}$.

Due to point (2) of Lemma 2, since $(\mathbf{A}_{\mathsf{id}_i}\mathbf{U}_{\mathsf{id}_i,\tau})$ is statistically indistinguishable from a random matrix, all the matrices $\mathbf{V}_\ell$ generated in $(3')$ are statistically close to the ones generated in (3). Thus, the two games are statistically indistinguishable. At this point we show that for every PPT adversary $\mathcal{A}$ which produces a forgery in the modified game we can construct a PPT algorithm $\mathcal{B}$ that solves the $\mathsf{SIS}(n, m \cdot Q, q, \beta_{\mathsf{SIS}})$ problem. $\mathcal{B}$ receives an SIS instance $\mathbf{A} := (\mathbf{A}_{\mathsf{id}_1} | \ldots | \mathbf{A}_{\mathsf{id}_Q}) \in \mathbb{Z}_q^{n \times mQ}$ and simulates the modified game to $\mathcal{A}$ by acting exactly as the challenger $\mathcal{C}$ described above. Then, once $\mathcal{A}$ outputs its forgery, according to the forgery's type, $\mathcal{B}$ proceeds as described below.

Type 2 Forgeries. Let $(\mathcal{P}^* := (f^*, \ell_1^*, \dots, \ell_t^*), \mathsf{m}^*, \sigma^* := (\mathsf{m}^*, \mathsf{z}^*, \mathsf{I}^*, \mathbf{U}^*, \mathbf{Z}^*))$ be a Type 2 forgery produced by $\mathcal{A}$ in the modified game. Moreover let $\sigma = (\mathsf{m}, \mathsf{z}, \mathsf{I}, \mathbf{U}, \mathbf{Z})$ be the signature obtained by honestly applying Eval to the signatures corresponding to labels $\ell_1^*, \dots, \ell_t^*$ that were given to $\mathcal{A}$. Parse $\mathbf{U} := \{\mathbf{U}_{\mathsf{id}}\}_{\mathsf{id} \in \mathsf{I}}$ and notice that by the correctness of the scheme we have that $\mathsf{m} = f^*(\mathsf{m}_{\ell_1^*}, \dots, \mathsf{m}_{\ell_t^*})$, $\mathsf{I} = \{\mathsf{id} : \mathsf{id} \in \mathcal{P}^*\}$, and $\sum_{\mathsf{id} \in \mathsf{I}} \mathbf{A}_{\mathsf{id}} \mathbf{U}_{\mathsf{id}} + \mathsf{m} \cdot \mathbf{G} = \mathbf{V}^*$. Moreover, by definition of Type 2 forgery recall that $\mathsf{m}^* \neq f^*(\mathsf{m}_{\ell_1^*}, \dots, \mathsf{m}_{\ell_t^*})$ and that the tuple satisfies verification. In particular, satisfaction of check (1) implies that $\mathsf{I} = \mathsf{I}^*$, while check (3) means $\sum_{\mathsf{id} \in \mathsf{I}^*} \mathbf{A}_{\mathsf{id}} \mathbf{U}_{\mathsf{id}}^* + \mathsf{m}^* \cdot \mathbf{G} = \mathbf{V}^*$. Combining the two equations above we obtain $\sum_{\mathsf{id} \in \mathsf{I}} \mathbf{A}_{\mathsf{id}} \tilde{\mathbf{U}}_{\mathsf{id}} = \tilde{\mathsf{m}} \cdot \mathbf{G}$, where $\tilde{\mathsf{m}} = \mathsf{m} - \mathsf{m}^* \neq \mathbf{0}$ and, for all $\mathsf{id} \in \mathsf{I}$, $\tilde{\mathbf{U}}_{\mathsf{id}} = \mathbf{U}_{\mathsf{id}}^* - \mathbf{U}_{\mathsf{id}} \in \mathcal{U}$ such that $\|\tilde{\mathbf{U}}_{\mathsf{id}}\|_\infty \leq \beta_{\mathsf{max}}$. Notice that there must exist at least one $\bar{\mathsf{id}} \in \mathsf{I}$ for which $\tilde{\mathbf{U}}_{\bar{\mathsf{id}}} \neq \mathbf{0}$.

Moreover, for all $\mathsf{id} \in \{\mathsf{id}_1, \dots, \mathsf{id}_Q\} \setminus \mathsf{I}$, define $\tilde{\mathbf{U}}_{\mathsf{id}} = \mathbf{0}$ and set $\tilde{\mathbf{U}} = \begin{pmatrix} \tilde{\mathbf{U}}_{\mathsf{id}_1} \\ \vdots \\ \tilde{\mathbf{U}}_{\mathsf{id}_Q} \end{pmatrix} \in \mathbb{Z}_q^{mQ \times m}$. Then, we have $\mathbf{A}\tilde{\mathbf{U}} = \tilde{\mathsf{m}} \cdot \mathbf{G}$.

Next $\mathcal{B}$ samples $\mathbf{r} \xleftarrow{\$} \{0,1\}^{mQ}$, sets $\mathbf{s} = \mathbf{Ar} \in \mathbb{Z}_q^n$, and computes $\mathbf{r}' = \mathbf{G}^{-1}(\tilde{\mathsf{m}}^{-1} \cdot \mathbf{s})$, so that $\mathbf{r}' \in \{0,1\}^m$ and $\tilde{\mathsf{m}} \cdot \mathbf{Gr}' = \mathbf{s}$. Finally, $\mathcal{B}$ outputs $\mathbf{u} = \tilde{\mathbf{U}}\mathbf{r}' - \mathbf{r} \in \mathbb{Z}_q^{mQ}$. We conclude the proof by claiming that the vector $\mathbf{u}$ returned by $\mathcal{B}$ is a solution of the SIS problem for the matrix $\mathbf{A}$. To see this observe that

$$\mathbf{A}(\tilde{\mathbf{U}}\mathbf{r}' - \mathbf{r}) = (\mathbf{A}\tilde{\mathbf{U}})\mathbf{r}' - \mathbf{Ar} = \tilde{\mathsf{m}} \cdot \mathbf{G} \cdot \mathbf{G}^{-1}(\tilde{\mathsf{m}}^{-1} \cdot \mathbf{s}) - \mathbf{s} = \mathbf{0} \; .$$

and $\|\mathbf{u}\|_\infty \leq (2m+1)\beta_{\mathsf{max}} \leq \beta_{\mathsf{SIS}}$.

It remains to show that $\mathbf{u} \neq \mathbf{0}$. We show that this is the case (i.e., $\tilde{\mathbf{U}}\mathbf{r}' \neq \mathbf{r}$) with overwhelming probability by using an entropy argument (the same argument used in [30]). In particular, this holds for any (worst case) choice of $\mathbf{A}, \tilde{\mathbf{U}}, \tilde{\mathsf{m}}$, and only based on the random choice of $\mathbf{r} \xleftarrow{\$} \{0,1\}^{mQ_{\mathsf{id}}}$. The intuition is that, even if $\mathbf{r}' = \mathbf{G}^{-1}(\mathbf{s}\hat{\mathsf{m}}^{-1})$ depends on $\mathbf{s} = \mathbf{Ar}$, $\mathbf{s}$ is too small to reveal much information about the random $\mathbf{r}$. More precisely, we have that $\mathbf{H}_\infty(\mathbf{r} \mid \mathbf{r}') \geq \mathbf{H}_\infty(\mathbf{r} \mid \mathbf{Ar})$ because $\mathbf{r}'$ is chosen deterministically based on $\mathbf{s} = \mathbf{Ar}$. Due to the Lemma 1, we have that $\mathbf{H}_\infty(\mathbf{r} \mid \mathbf{Ar}) \geq \mathbf{H}_\infty(\mathbf{r}) - \mathsf{log}(|\mathcal{S}|)$, where $\mathcal{S}$ is the space of all possible $\mathbf{s}$. Since $\mathbf{s} \in \mathbb{Z}_q^n$, $|\mathcal{S}| = q^n$, and then $\mathsf{log}(|\mathcal{S}|) = \mathsf{log}(q^n) = \mathsf{log}((2^{\mathsf{log} q})^n) = n \log((2^{\log q})) = n \log q$. Regarding $\mathbf{H}_\infty(\mathbf{r})$, since $\mathbf{H}_\infty(X) := -\mathsf{log}\big(\max_x \Pr[X = x]\big)$, we have $\mathbf{H}_\infty(\mathbf{r}) = -\mathsf{log}\big(2^{-mQ}\big) = mQ \geq m$. Then, $\mathbf{H}_\infty(\mathbf{r} \mid \mathbf{r}') \geq \mathbf{H}_\infty(\mathbf{r}) - \mathsf{log}(\mathcal{S}) \geq m - n \log q = \omega(\log \lambda)$. Since we know that for random variables X, Y the optimal probability of an unbounded adversary guessing X given the correlated value Y is $2^{-\mathbf{H}_\infty(X|Y)}$, then $\Pr[\mathbf{r} = \tilde{\mathbf{U}}\mathbf{r}'] \leq 2^{-\mathbf{H}_\infty(\mathbf{r}|\mathbf{r}')} \leq 2^{-\omega(\log \lambda)} = \mathsf{negl}(\lambda)$.

Type 3 Forgery. Let $(\mathcal{P}^* := (f^*, \ell_1^*, \dots, \ell_t^*), \mathsf{m}^*, \sigma^* := (\mathsf{m}^*, \mathsf{z}^*, \mathsf{I}^*, \mathbf{U}^*, \mathbf{Z}^*))$ be a Type 3 forgery produced by $\mathcal{A}$ in the modified game such that there exists (at least) one label $\ell_j^* = (\mathsf{id}^*, \tau^*)$ such that $\mathsf{id}^* = \mathsf{id}_i$ but $\tau^* \notin \mathcal{T}_i$.[11] Actually, without loss of generality we can assume that there is exactly one of such labels;

[11] It is easy to see that the case in which id^* is new would imply the generation of a new $\mathbf{A}_{\mathsf{id}^*}$, which would make the verification equations hold with negligible probability (over the random choice of $\mathbf{A}_{\mathsf{id}^*}$).

if this is not the case, one could indeed redefine another adversary that makes more queries until it misses only this one. Note that for such a tag $\tau^* \notin \mathcal{T}_i$, $\mathcal{B}$ simulated $\mathbf{V}_{\mathsf{id}_i,\tau^*} = \mathbf{A}\mathbf{U}_{\mathsf{id}_i,\tau^*} + b_{i,\tau^*}\mathbf{G}$ for a randomly chosen bit $b_{i,\tau^*} \xleftarrow{\$} \{0,1\}$, that is perfectly hidden from $\mathcal{A}$.

By definition of Type 3 forgery, the tuple passes verification, and in particular check (4) $\sum_{\mathsf{id}\in\mathsf{I}^*} \mathbf{A}_{\mathsf{id}}\mathbf{Z}^*_{\mathsf{id}} + \mathsf{z}^* \cdot \mathbf{G} = \mathbf{V}^+ = \sum_{i=1}^{t} \gamma_i \cdot \mathbf{V}_{\ell^*_i}$ where the right hand side of the equation holds by construction of the verification algorithm. Moreover, let $\sigma = (\mathsf{m}, \mathsf{z}, \mathsf{I}, \mathbf{U}, \mathbf{Z})$ be the signature obtained by honestly applying Eval to the signatures corresponding to labels $\ell^*_1, \ldots, \ell^*_t$; in particular for the specific, missing, label ℓ^*_j $\mathcal{B}$ uses the values $\mathbf{U}_{\mathsf{id}_i,\tau^*}, b_{i,\tau^*}$ used to simulate $\mathbf{V}_{\mathsf{id}_i,\tau^*}$. Parsing $\mathbf{Z} := \{\mathbf{Z}_{\mathsf{id}}\}_{\mathsf{id}\in\mathsf{I}}$, notice that by correctness it holds $\mathsf{I} = \{\mathsf{id} : \mathsf{id} \in \mathcal{P}^*\}$ and $\sum_{\mathsf{id}\in\mathsf{I}} \mathbf{A}_{\mathsf{id}}\mathbf{Z}_{\mathsf{id}} + \mathsf{z}\cdot\mathbf{G} = \mathbf{V}^+$ where $\mathsf{z} = \sum_{i=1, i\neq j}^{t} \gamma_i \mathsf{m}_i + \gamma_j b_{i,\tau^*}$. Now, the observation is that every $\gamma_i \leq 2^d < q$, i.e., $\gamma_i \neq 0 \mod q$. Since b_{i,τ^*} is random and perfectly hidden to $\mathcal{A}$ we have that with probability $1/2$ it holds $\mathsf{z} \neq \mathsf{z}^*$.

Thus, if $\mathsf{z} \neq \mathsf{z}^*$, $\mathcal{B}$ combines the equalities on $\mathbf{V}^+$ to come up with an equation $\sum_{\mathsf{id}\in\mathsf{I}} \mathbf{A}_{\mathsf{id}}\tilde{\mathbf{Z}}_{\mathsf{id}} = \tilde{\mathsf{z}} \cdot \mathbf{G}$ where $\tilde{\mathsf{z}} = \mathsf{z} - \mathsf{z}^* \neq 0 \mod q$ and, for all $\mathsf{id} \in \mathsf{I}$, $\tilde{\mathbf{Z}}_{\mathsf{id}} = \mathbf{Z}^*_{\mathsf{id}} - \mathbf{Z}_{\mathsf{id}} \in \mathcal{U}$ such that $\|\tilde{\mathbf{Z}}_{\mathsf{id}}\|_\infty \leq \beta_{\mathsf{max}}$.

Finally, using the same technique as in the case of Type 2 forgeries, $\mathcal{B}$ can compute a vector $\mathbf{u}$ that is a solution of SIS with overwhelming probability, i.e., $\mathbf{A}\mathbf{u} = 0$. Therefore, we have proven that if an adversary $\mathcal{A}$ can break the MKHSig scheme with non negligible probability, then $\mathcal{C}$ can use such an $\mathcal{A}$ to break the SIS assumption for $\mathbf{A}$ with non negligible probability as well.

A Variant with Unbounded Tag Space in the Random Oracle Model. In this section, we show that the construction of multi-key homomorphic signatures of Section 4.2 can be easily modified in order to have short public keys and to support an unbounded tag space $\mathcal{T} = \{0,1\}^*$. Note that once arbitrary tags are allowed, the scheme also allows to handle *multiple datasets* for free. In fact, one can always extend tags to include the dataset name, i.e., simply redefine each tag τ as consisting of two substrings $\tau = (\Delta, \tau')$ where Δ is the dataset name and τ' the actual tag. The idea of modifying the scheme to support an unbounded tag space is simple and was also suggested in [30] for their construction. Instead of sampling matrices $\{\mathbf{V}_{\mathsf{id},1}, \ldots \mathbf{V}_{\mathsf{id},\mathsf{T}}\}$ in KeyGen, one can just choose a random string $r_{\mathsf{id}} \xleftarrow{\$} \{0,1\}^\lambda$ and define every $\mathbf{V}_{\mathsf{id},\tau} := \hat{\mathsf{H}}(r_{\mathsf{id}}, \tau)$ where $\hat{\mathsf{H}} : \{0,1\}^* \to \mathcal{V}$ is a hash function chosen in Setup (modeled as a random oracle in the proof). In all the remaining algorithms, every time one needs $\mathbf{V}_{\mathsf{id},\tau}$, this is obtained using $\hat{\mathsf{H}}$.

For this modified scheme, we also provide an idea of how the security proof of Theorem 1 has to be modified to account for these changes. The main change is the simulation of hash queries, which is done as follows.

Before phase (1), where $\mathcal{A}$ declares its queries, $\mathcal{B}$ simply answers every query $\hat{\mathsf{H}}(r, \tau)$ with a randomly chosen $\mathbf{V} \xleftarrow{\$} \mathcal{V}$. Afterwards, once $\mathcal{A}$ has declared all its queries, $\mathcal{B}$ chooses $r_{\mathsf{id}_1}, \ldots, r_{\mathsf{id}_Q} \xleftarrow{\$} \{0,1\}^\lambda$ and programs the random oracle so that, for all $\tau \in \mathcal{T}_i$, $\hat{\mathsf{H}}(r_{\mathsf{id}_i}, \tau) = \mathbf{V}_{\mathsf{id}_i,\tau}$ where $\mathbf{V}_{\mathsf{id}_i,\tau}$ is the same matrix generated in the phase (3) of the modified game. On the other hand, for all

$\tau \notin \mathcal{T}_i$, $\hat{\mathsf{H}}(r_{\mathsf{id}_i}, \tau) = \mathbf{V}_{\mathsf{id}_i,\tau}$ where $\mathbf{V}_{\mathsf{id}_i,\tau} = \mathbf{A}_{\mathsf{id}}\mathbf{U}_{\mathsf{id}_i,\tau} + b_{i,\tau}\mathbf{G}$ for a randomly chosen $\mathbf{U}_{\mathsf{id}_i,\tau} \xleftarrow{\$} \mathcal{U}$. All other queries $\hat{\mathsf{H}}(r, \tau)$ where $r \neq r_{\mathsf{id}_i}, \forall i \in [Q]$ are answered with random $\mathbf{V} \xleftarrow{\$} \mathcal{V}$. With this simulation, it is not hard to see that, from $\mathcal{A}$'s forgery $\mathcal{B}$ can extract a solution for SIS (except for some negligible probability that $\mathcal{A}$ guesses one of r_{id_i} before seeing it).

4.3 From a Single Dataset to Multiple Datasets

In this section, we present a generic transformation to convert a single-dataset MKHSig scheme into a scheme that supports multiple datasets. The intuition behind this transformation is similar to the one employed in [30] and implicitly used in [16,13], except that here we have to use additional techniques to deal with the multi-key setting. We combine a standard signature scheme $\mathsf{NH.Sig}$ (non-homomorphic) with a single dataset multi-key homomorphic signature scheme MKHSig'. The idea is that for every new dataset Δ, every user generates fresh keys of the multi-key homomorphic scheme MKHSig' and then uses the standard signature scheme $\mathsf{NH.Sig}$ to sign the dataset identifier Δ together with the generated public key. More precisely, in our transformation we assume to start with (single-dataset) multi-key homomorphic signature schemes in which the key generation algorithm can be split into two independent algorithms: KeyGen_1 that outputs some public parameters related to the identity id, and KeyGen_2 which outputs the actual keys. Differently than [30], in our scheme the signer does not need to sign the whole dataset at once, nor has to fix a bound N on the dataset size (unless such a bound is already contained in MKHSig').

In more details, let $\mathsf{NH.Sig} = (\mathsf{NH.KeyGen}, \mathsf{NH.Sign}, \mathsf{NH.Ver})$ be a standard (non-homomorphic) signature scheme, and let $\mathsf{MKHSig}' = (\mathsf{Setup}', \mathsf{KeyGen}', \mathsf{Sign}', \mathsf{Eval}', \mathsf{Ver}')$ be a single-dataset multi-key homomorphic signature scheme. We construct a multi-dataset multi-key homomorphic signature scheme $\mathsf{MKHSig} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Eval}, \mathsf{Ver})$ as follows.

$\mathsf{Setup}(1^\lambda)$. The setup algorithm samples parameters of the single-dataset multi-key homomorphic signature scheme, $\mathsf{pp}' \leftarrow \mathsf{Setup}'(1^\lambda)$, together with a description of a PRF $F : K \times \{0,1\}^* \to \{0,1\}^\rho$, and outputs $\mathsf{pp} = (\mathsf{pp}', F)$.

$\mathsf{KeyGen}(\mathsf{pp})$. The key generation algorithm runs $\mathsf{NH.KeyGen}$ to get $(\mathsf{pk}^{\mathsf{NH}}_{\mathsf{id}}, \mathsf{sk}^{\mathsf{NH}}_{\mathsf{id}})$, a pair of keys for the standard signature scheme. In addition, it runs KeyGen_1 to generate user-specific public parameters $\mathsf{pp}_{\mathsf{id}}$, and chooses a seed K_{id} for the PRF F. The final output is the vector $(\mathsf{sk}_{\mathsf{id}}, \mathsf{ek}_{\mathsf{id}}, \mathsf{vk}_{\mathsf{id}})$: where $\mathsf{sk}_{\mathsf{id}} = (\mathsf{sk}^{\mathsf{NH}}_{\mathsf{id}}, \mathsf{K}_{\mathsf{id}})$, $\mathsf{ek}_{\mathsf{id}} = (\mathsf{pp}_{\mathsf{id}})$ and $\mathsf{vk}_{\mathsf{id}} = (\mathsf{pk}^{\mathsf{NH}}_{\mathsf{id}}, \mathsf{pp}_{\mathsf{id}})$.

$\mathsf{Sign}(\mathsf{sk}_{\mathsf{id}}, \Delta, \ell, \mathsf{m})$. The signing algorithm proceeds as follows. First it samples the keys of the single-dataset multi-key homomorphic signature scheme by feeding randomness $F_{\mathsf{K}_{\mathsf{id}}}(\Delta)$ to KeyGen_2, i.e., it runs $\mathsf{KeyGen}_2(\mathsf{pp}; F_{\mathsf{K}_{\mathsf{id}}}(\Delta))$ to obtain the keys $(\mathsf{sk}^{\Delta}_{\mathsf{id}}, \mathsf{ek}^{\Delta}_{\mathsf{id}}, \mathsf{vk}^{\Delta}_{\mathsf{id}})$.[12] The algorithm then runs $\sigma' \leftarrow \mathsf{Sign}'(\mathsf{sk}^{\Delta}_{\mathsf{id}}, \ell, \mathsf{m})$, and uses the non-homomorphic scheme to sign the concatenation of the public key $\mathsf{vk}^{\Delta}_{\mathsf{id}}$ and the dataset identifier Δ, i.e., $\sigma^{\Delta}_{\mathsf{id}} \leftarrow \mathsf{NH.Sign}(\mathsf{sk}^{\mathsf{NH}}_{\mathsf{id}}, \mathsf{vk}^{\Delta}_{\mathsf{id}}|\Delta)$, The output is the tuple $\sigma := (\mathsf{I} = \{\mathsf{id}\}, \sigma', \mathsf{par}_\Delta)$ where $\mathsf{par}_\Delta = \{(\mathsf{ek}^{\Delta}_{\mathsf{id}}, \mathsf{vk}^{\Delta}_{\mathsf{id}}, \sigma^{\Delta}_{\mathsf{id}})\}$.

[12] Here we assume that a ρ-bits string is sufficient, otherwise it can always be stretched using a PRG.

Note that the use of the PRF allows every signer (having the same K_{id}) to generate the same keys of the scheme MKHSig' on the same dataset Δ.

$\mathsf{Eval}(f, \{(\sigma_i, \mathsf{EKS}_i)\}_{i \in [t]})$**.** For each $i \in [t]$, the algorithm parses every signature as $\sigma_i := (\mathsf{I}_i, \sigma'_i, \mathsf{par}_{\Delta,i})$ with $\mathsf{par}_{\Delta,i} = \{\mathsf{ek}^{\Delta}_{\mathsf{id}}, \mathsf{vk}^{\Delta}_{\mathsf{id}}, \sigma^{\Delta}_{\mathsf{id}}\}_{\mathsf{id} \in \mathsf{I}_i}$, and sets $\mathsf{EKS}'_i = \{\mathsf{ek}^{\Delta}_{\mathsf{id}}\}_{\mathsf{id} \in \mathsf{I}_i}$. It computes $\sigma' \leftarrow \mathsf{Eval}'(f, \{\sigma'_i, \mathsf{EKS}'_i\}_{i \in [t]})$, defines $\mathsf{I} = \cup_{i=1}^{t} \mathsf{I}_i$ and $\mathsf{par}_{\Delta} = \cup_{i=1}^{t} \mathsf{par}_{\Delta,i}$. The final output is $\sigma = (\mathsf{I}, \sigma', \mathsf{par}_{\Delta})$.

$\mathsf{Ver}(\mathcal{P}, \Delta, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}}, \mathsf{m}, \sigma)$**.** The verification algorithm begins by parsing the verification keys as $\mathsf{vk}_{\mathsf{id}} := (\mathsf{pk}^{\mathsf{NH}}_{\mathsf{id}}, \mathsf{pp}_{\mathsf{id}})$ for each $\mathsf{id} \in \mathsf{I}$, and also the signature as $\sigma = (\mathsf{I}, \sigma', \mathsf{par}_{\Delta})$ with $\mathsf{par}_{\Delta} = \{(\mathsf{ek}^{\Delta}_{\mathsf{id}}, \mathsf{vk}^{\Delta}_{\mathsf{id}}, \sigma^{\Delta}_{\mathsf{id}})\}_{\mathsf{id} \in \mathsf{I}}$. Then, it proceeds with two main steps. First, for each $\mathsf{id} \in \mathsf{I}$, it verifies the standard signature $\sigma^{\Delta}_{\mathsf{id}}$ on the public key of the single-dataset multi-key homomorphic scheme and the given dataset, i.e., it checks whether $\mathsf{NH.Ver}(\mathsf{pk}^{\mathsf{NH}}_{\mathsf{id}}, \mathsf{vk}^{\Delta}_{\mathsf{id}} | \Delta, \sigma^{\Delta}_{\mathsf{id}}) = 1, \forall\, \mathsf{id} \in \mathsf{I}$. If at least one of the previous equations is not satisfied, the algorithm returns 0, otherwise it proceeds to the second check and returns the output of $\mathsf{Ver}'(\mathcal{P}, \{\mathsf{pp}_{\mathsf{id}}, \mathsf{vk}^{\Delta}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}}, \mathsf{m}, \sigma')$.

Authentication Correctness. Correctness of the scheme substantially follows from the correctness of the regular signature scheme $\mathsf{NH.Sig}$, the single-dataset multi-hey homomorphic scheme MKHSig' and the PRF F.

Evaluation Correctness. Evaluation correctness follows directly from the correctness of the evaluation algorithm Eval' of the single-dataset MKHSig scheme, the correctness of $\mathsf{NH.Sig}$ and of the PRF.

Security. Intuitively, the security of the scheme follows from two main observations. First, no adversary is able to fake the keys of the single-dataset multi-key homomorphic signature scheme, due to the security of the standard signature scheme and the property of pseudo-random functions. Secondly, no adversary can tamper with the results of Eval for a specific dataset as this would correspond to breaking the security of the single-dataset MKHSig' scheme.

Theorem 2. *If F is a secure pseudo-random function,* $\mathsf{NH.Sig}$ *is an unforgeable signature scheme and* MKHSig' *is a secure single-dataset multi-key homomorphic signature scheme, then the* MKHSig *scheme for multiple datasets described in Section 4.3 is secure against adversaries that make static corruptions of keys and produce forgeries as in Definition 4.*

The full proof of Theorem 2 is given in [22].

5 Our Multi-Key Homomorphic MAC from OWFs

In this section, we describe our construction of a multi-key homomorphic authenticator with private verification keys and supporting the evaluation of low-degree arithmetic circuits. More precisely, for a computation represented by an arithmetic circuit of degree d and involving inputs from n distinct identities, the final authenticator has size $\binom{n+d}{d}$, that is bounded by $poly(n)$ (for constant d) or by $poly(d)$ (for constant n). Essentially, the authenticators of our scheme grow with the degree of the circuit and the number of distinct users involved in the computation, whereas their size remains *independent* of the total number of inputs

/ users. This property is particularly desirable in contexts that involve a small set of users each of which contributes with several inputs.

Although our multi-key homomorphic MAC supports less expressive computations than our homomorphic signatures of Section 4, the scheme comes with two main benefits. First, it is based on a simple, general assumption: it relies on pseudo-random functions and thus is secure only assuming existence of one-way functions (OWF). Second, the scheme is very intuitive and efficient: fresh MACs essentially consist only of two $\mathbb{F}_p$ field elements (where p is a prime of λ bits) and an identity identifier; after evaluation, the authenticators consist of $\binom{n+d}{d}$ elements in $\mathbb{F}_p$, and homomorphic operations are simply additions and multiplications in the multi-variate polynomial ring $\mathbb{F}_p[X_1, \ldots, X_n]$.

We describe the five algorithms of our scheme MKHMac below. We note that our solution is presented for single data set only. However, since it admits labels that are arbitrarily long strings it is straight-forward to extend the scheme for handling multiple data sets: simply redefine each tag τ as consisting of two substrings $\tau = (\Delta, \tau')$ where Δ is the dataset name and τ' the actual tag.

$\mathsf{Setup}(1^\lambda)$**.** The setup algorithm generates a λ-bit prime p and let the message space be $\mathcal{M} := \mathbb{F}_p$. The set of identities is $\mathsf{ID} = [\mathsf{C}]$ for some integer bound $\mathsf{C} \in \mathbb{N}$, while the tag space consists of arbitrary binary strings, i.e., $\mathcal{T} = \{0,1\}^*$. The set $\mathcal{F}$ of admissible functions is made up of all arithmetic circuits whose degree d is bounded by some polynomial in the security parameter. The setup algorithm outputs the public parameters pp which include the descriptions of $\mathcal{M}, \mathsf{ID}, \mathcal{T}, \mathcal{F}$ as in Section 3, as well as the description of a PRF family $F : \mathcal{K} \times \{0,1\}^* \to \mathbb{F}_p$ with seed space $\mathcal{K}$. The public parameters define also the authenticator space. Each authenticator σ consists of a pair (I, y) where $\mathsf{I} \subseteq \mathsf{ID}$ and y is in the C-variate polynomial ring $\mathbb{F}_p[X_1, \ldots, X_\mathsf{C}]$. More precisely, if C is set up as a very large number (e.g., $\mathsf{C} = 2^\lambda$) the polynomials y can still live in some smaller sub-rings of $\mathbb{F}_p[X_1, \ldots, X_\mathsf{C}]$.

$\mathsf{KeyGen}(\mathsf{pp})$**.** The key generation algorithm picks a random $x \xleftarrow{\$} \mathbb{F}_p^*$, a PRF seed $K \xleftarrow{\$} \mathcal{K}$, and outputs $(\mathsf{sk}, \mathsf{ek}, \mathsf{vk})$ where $\mathsf{sk} = \mathsf{vk} = (x, K)$ and ek is void.

$\mathsf{Auth}(\mathsf{sk}, \ell, \mathsf{m})$**.** In order to authenticate the message m with label $\ell = (\mathsf{id}, \tau) \in \mathsf{ID} \times \mathcal{T}$, the authentication algorithm produces an authenticator $\sigma = (\mathsf{I}, \mathsf{y})$ where $\mathsf{I} \subseteq \mathsf{ID}$ and $\mathsf{y} \in \mathbb{F}_p[X_\mathsf{id}] \subset \mathbb{F}_p[X_1, \ldots, X_\mathsf{C}]$. The set I is simply $\{\mathsf{id}\}$. The polynomial y is a degree-1 polynomial in the variable X_id such that $\mathsf{y}(0) = \mathsf{m}$ and $\mathsf{y}(x_\mathsf{id}) = F(K_\mathsf{id}, \ell)$. Note that the coefficients of $\mathsf{y}(X_\mathsf{id}) = y_0 + y_\mathsf{id} X_\mathsf{id} \in \mathbb{F}_p[X_\mathsf{id}]$ can be efficiently computed with the knowledge of x_id by setting $y_0 = \mathsf{m}$ and $y_\mathsf{id} = \frac{F(K_\mathsf{id}, \ell) - \mathsf{m}}{x_\mathsf{id}}$. Moreover, y can be compactly represented by only giving the coefficients $y_0, y_\mathsf{id} \in \mathbb{F}_p$.

$\mathsf{Eval}(f, \{\sigma_k\}_{k \in [t]})$**.** Given a t-input arithmetic circuit $f : \mathbb{F}_p^t \to \mathbb{F}_p$, and the t authenticators $\{\sigma_k := (\mathsf{I}_k, \mathsf{y}_k)\}_k$, the evaluation algorithm outputs $\sigma = (\mathsf{I}, \mathsf{y})$ obtained in the following way. First, it determines all the identities involved in the computation by setting $\mathsf{I} = \cup_{k=1}^t \mathsf{I}_k$. Then every polynomial y_k is "expanded" into a polynomial $\hat{\mathsf{y}}_k$, defined on the variables X_id corresponding to all the identities in I. This is done using the canonical embedding $\mathbb{F}_p[X_\mathsf{id} : \mathsf{id} \in \mathsf{I}_k] \hookrightarrow \mathbb{F}_p[X_\mathsf{id} : \mathsf{id} \in \mathsf{I}]$. It is worth noticing that the terms of $\hat{\mathsf{y}}_k$ that depend on variables in $\mathsf{I} \setminus \mathsf{I}_k$ have coefficient 0. Next, let $\hat{f} : \mathbb{F}_p[X_{\mathsf{id}} : \mathsf{id} \in \mathsf{I}]^t \to \mathbb{F}_p[X_{\mathsf{id}} : \mathsf{id} \in \mathsf{I}]$ be the arithmetic circuit corresponding to the given f, i.e., $\hat{f}$ is the same as f except that additions (resp. multiplications) in $\mathbb{F}_p$ are replaced by additions (resp. multiplications) over the polynomial ring $\mathbb{F}_p[X_{\mathsf{id}} : \mathsf{id} \in \mathsf{I}]$. Finally, y is obtained as $\mathsf{y} = \hat{f}(\hat{\mathsf{y}}_1, \ldots, \hat{\mathsf{y}}_t)$.

$\mathsf{Ver}(\mathcal{P}, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}}, \mathsf{m}, \sigma)$**.** Let $\mathcal{P} = (f, \ell_1, \ldots, \ell_t)$ be a labeled program where f is a degree-d arithmetic circuit and every label is of the form $\ell_k = (\mathsf{id}_k, \tau_k)$. Let $\sigma = (\mathsf{I}, \mathsf{y})$ where $\mathsf{I} = \{\bar{\mathsf{id}}_1, \ldots, \bar{\mathsf{id}}_n\}$ with $\bar{\mathsf{id}}_i \neq \bar{\mathsf{id}}_j$ for $i \neq j$. The verification algorithm outputs 1 (accept) if and only if the authenticator satisfies the following three checks. Otherwise it outputs 0 (reject).

$$\{\bar{\mathsf{id}}_1, \ldots, \bar{\mathsf{id}}_n\} = \{\mathsf{id} : \mathsf{id} \in \mathcal{P}\}, \tag{5}$$

$$\mathsf{y}(0, \ldots, 0) = \mathsf{m}\ , \tag{6}$$

$$\mathsf{y}(x_{\bar{\mathsf{id}}_1}, \ldots, x_{\bar{\mathsf{id}}_n}) = f(F(K_{\mathsf{id}_1}, \ell_1), \ldots, F(K_{\mathsf{id}_t}, \ell_t))\ . \tag{7}$$

In the remainder of the section, we discuss the efficiency and succinctness of our MKHMac and prove the correctness of our scheme. We conclude with the security analysis of the proposed MKHMac scheme.

Succinctness. Let us consider the case of an authenticator σ which was obtained after running Eval on a circuit of degree d and taking inputs from n distinct identities. Note that every σ consists of two elements: a set $\mathsf{I} \subseteq [\mathsf{C}]$ and a polynomial $\mathsf{y} \in \mathbb{F}_p[X_{\bar{\mathsf{id}}} : \bar{\mathsf{id}} \in \mathsf{I}]$.

For the set I, it is easy to see that $|\mathsf{I}| = n$ and I can be represented with $n \log \mathsf{C}$ bits. The other part of the authenticator, y, is instead an n-variate polynomial in $\mathbb{F}_p[X_{\bar{\mathsf{id}}_1}, \ldots, X_{\bar{\mathsf{id}}_n}]$ of degree d. Since the circuit degree is d, the maximum number of coefficients of y is $\binom{n+d}{d}$. More precisely, the total size of y depends on the particular representation of the multi-variate polynomial y which is chosen for implementation. In [22] we discuss some possible representations (further details can also be found in [38]). For example, when employing the *sparse representation* of polynomials, the size of y is bounded by $O(nt \log d)$ where t is the number of non-zero coefficients in y (note that in the worst case, a polynomial $\mathsf{y} \in \mathbb{F}_p[X_{\bar{\mathsf{id}}_1}, \ldots, X_{\bar{\mathsf{id}}_n}]$ of degree d has at most $t = \binom{n+d}{d}$ non-zero coefficients). Thus, setting $\log \mathsf{C} \approx \log p \approx \lambda$, we have that the size in bits of the authenticator σ is $|\sigma| \leq \lambda n + \lambda \binom{n+d}{d}$. Ignoring the security parameter, we have that $|\sigma| = \mathsf{poly}(n)$ when d is constant, or $|\sigma| = \mathsf{poly}(d)$ when n is constant.

Efficiency of Eval. In what follows, we discuss the cost of computing additions and multiplications over authenticators in our MKHMac scheme. Let $\sigma^{(i)} = (\mathsf{I}^{(i)}, \mathsf{y}^{(i)})$, for $i = 1, 2$ be two authenticators and consider the operation $\sigma = \mathsf{Eval}(g, \sigma^{(1)}, \sigma^{(2)})$ where g is a fan-in-2 addition or multiplication gate. In both cases the set I of identities of $\sigma = (\mathsf{I}, \mathsf{y})$ is obtained as the union $\mathsf{I} = \mathsf{I}^{(1)} \cup \mathsf{I}^{(2)}$ that can be computed in time $O(n)$, where $n = |\mathsf{I}|$, assuming the sets $\mathsf{I}^{(1)}, \mathsf{I}^{(2)}$ are ordered. Regarding the computation of y from $\mathsf{y}^{(1)}$ and $\mathsf{y}^{(2)}$, one has to first embed each y_i into the ring $\mathbb{F}_p[X_{\bar{\mathsf{id}}} : \bar{\mathsf{id}} \in \mathsf{I}]$, and then evaluate addition (resp. multiplication) over $\mathbb{F}_p[X_{\bar{\mathsf{id}}} : \bar{\mathsf{id}} \in \mathsf{I}]$. Again, the costs of these operations depend on the adopted representation [38,24].

Using the *sparse representation* of polynomials, expanding a y having t non-zero coefficients into an n-variate polynomial $\hat{\mathsf{y}}$ requires time at most $O(tn)$. To give an idea, such expansion indeed consists simply into inserting zeros in the correct positions of the exponent vectors of every non-zero monomial term of y. On the other hand, the complexity of operations (additions and multiplications) on polynomials using the *sparse representation* is usually estimated in terms of the number of monomial comparisons. The cost of such comparisons depends on the specific monomial ordering chosen, but is usually $O(n \log d)$, where n is the total number of variables and d is the maximum degree. Given two polynomials in sparse representation having t_1 and t_2 non-zero terms respectively, addition costs about $O(t_1 t_2)$ monomial comparisons (if the monomial terms are stored in sorted order the cost of addition drops to $O(t_1 + t_2)$), while multiplication requires to add (merge) t_2 intermediate products of t_1 terms each, and can be performed with $O(t_1 t_2 \log t_2)$ monomial comparisons [24].

Correctness. AUTHENTICATION CORRECTNESS. By construction, each fresh authenticator $\sigma = (\mathsf{I}, \mathsf{y})$ of a message m labeled by $\ell := (\mathsf{id}, \tau)$ is of the form $\mathsf{I} = \{\mathsf{id}\}$ and $\mathsf{y}(X_{\mathsf{id}}) := y_0 + y_{\mathsf{id}} X_{\mathsf{id}} = \mathsf{m} + \frac{F(K_{\mathsf{id}}, \ell) - \mathsf{m}}{x_{\mathsf{id}}} X_{\mathsf{id}}$. Thus the set I satisfies equation (5) since $\{\mathsf{id} : \mathsf{id} \in \mathcal{I}_\ell\} = \{\mathsf{id}\}$. The two last verification checks (6) and (7) are automatically granted for the identity program $\mathcal{I}_\ell$ because $\mathsf{y}(0) = y_0 = \mathsf{m}$ and $\mathsf{y}(x_{\mathsf{id}}) = \mathsf{m} + \frac{F(K_{\mathsf{id}}, \ell) - \mathsf{m}}{x_{\mathsf{id}}} x_{\mathsf{id}} = F(K_{\mathsf{id}}, \ell)$.

EVALUATION CORRECTNESS. The correctness of the Eval algorithm essentially comes from the structure of the multi-variate polynomial ring. We provide the detailed proof in the full version of the paper [22].

SECURITY. In what follows we prove the security of our scheme against adversaries that make static corruptions, and produce forgeries according to the following restrictions.

Definition 8 (Weak Forgery). *Consider an execution of the experiment described in Section 3,* $\mathsf{HomUF\text{-}CMA}_{\mathcal{A},\mathsf{MKHAut}}(\lambda)$ *where* $(\mathcal{P}^*, \Delta^*, \mathsf{m}^*, \sigma^*)$ *is the tuple returned by the adversary at the end of the experiment, with* $\mathcal{P}^* = (f^*, \ell_1^*, \ldots, \ell_t^*)$*,* Δ^* *a dataset identifier,* $\mathsf{m}^* \in \mathcal{M}$ *and* σ^* *an authenticator. First, we say that the labeled program* $\mathcal{P}^*$ *is* well-defined *on a list* L *if either one of the following two cases occurs:*

1. *There exists* $i \in [t]$ *such that* $(\ell_i^*, \cdot) \notin L$ *(i.e.,* $\mathcal{A}$ *never made a query with label* ℓ_i^**), and* $f^*(\{\mathsf{m}_j\}_{(\ell_j, \mathsf{m}_j) \in L} \cup \{\tilde{\mathsf{m}}_j\}_{(\ell_j, \cdot) \notin L})$ *outputs the same value for all possible choices of* $\tilde{\mathsf{m}}_j \in \mathcal{M}$*;*
2. L *contains the tuples* $(\ell_1^*, \mathsf{m}_1), \ldots, (\ell_t^*, \mathsf{m}_t)$*, for some messages* $\mathsf{m}_1, \ldots, \mathsf{m}_t$*.*

Then we say that $(\mathcal{P}^*, \Delta^*, \mathsf{m}^*, \sigma^*)$ *is a* weak forgery *if* $\mathsf{Ver}(\mathcal{P}^*, \Delta^*, \{\mathsf{vk}_{\mathsf{id}}\}_{\mathsf{id} \in \mathcal{P}^*}, \mathsf{m}^*, \sigma^*) = 1$ *and either one of the following conditions is satisfied:*

Type 1: L_{Δ^*} *was not initialized during the game (i.e.,* Δ^* *was never queried).*
Type 2: $\mathcal{P}^*$ *is well-defined on* L_{Δ^*} *but* $\mathsf{m}^* \neq f^*(\{\mathsf{m}_j\}_{(\ell_j, \mathsf{m}_j) \in L_{\Delta^*}} \cup \{0\}_{\ell_j \notin L_{\Delta^*}})$ *(i.e.,* m^* *is not the correct output of* $\mathcal{P}^*$ *when executed over previously authenticated messages).*
Type 3: $\mathcal{P}^*$ *is not well-defined on* L_{Δ^*}*.*

Although Definition 8 is weaker than our Definition 4, we stress that the above definition still protects the verifier from adversaries that try to cheat on the output of a computation. In more details, the difference between Definition 8 and Definition 4 is the following: if f^* has an input wire that has never been authenticated during the game (a Type 3 forgery in Definition 4), but f^* is *constant* with respect to such input wire, then the above definition does not consider it a forgery. The intuitive reason why such a relaxed definition still makes sense is that "irrelevant" inputs would not help in any case the adversary to cheat on the output of f^*. Definition 8 is essentially the multi-key version of the forgery definition used in previous (single-key) homomorphic MAC works, e.g., [11]. As discussed in [23] testing whether a program is well-defined may not be doable in polynomial time in the most general case (i.e., every class of functions). However, in [12] it is shown how this can be done efficiently via a probabilistic test in the case of arithmetic circuits of degree d over a finite field of order p such that $d/p < 1/2$. Finally, we notice that for our MKHMac Type 1 forgeries cannot occur as the scheme described here supports only one dataset.[13]

Theorem 3. *If F is a pseudo-random function then the multi-key homomorphic MAC described in Section 5 is secure against adversaries that make static corruptions of keys and produce forgeries as in Definition 8.*

Note that we can deal with corruptions via our generic result of Proposition 1. Therefore, it is sufficient to prove the security against adversaries that make no corruptions. The proof is done via a chain of games following this (intuitive) path. First, we rule out adversaries that make Type 3 forgeries. Intuitively, this can be done as the adversary has never seen one of the inputs of the computation, and in particular an input which can change the result. Second, we replace every PRF instance with a truly random function. Note that at this point the security of the scheme is information theoretic. Third, we change the way to answer verification queries that are candidates to be Type 2 forgeries. Finally, we observe that in this last game the adversary gains no information on the secret keys x_i and thus has negligible probability of making a Type 2 forgery. Due to space restrictions, the detailed and formal proofs appear in only in the full version [22].

6 Conclusions

In this paper, we introduced the concept of multi-key homomorphic authenticators, a cryptographic primitive that enables an untrusted third party to execute a function f on data authenticated using different secret keys in order to obtain a value certifying the correctness of f's result, which can be checked with knowledge of corresponding verification keys. In addition to providing suitable definitions, we also propose two constructions: one which is publicly verifiable

[13] As noted at the beginning of the section the extension to multiple datasets is straightforward given that tags are arbitrary strings. When such extension is applied it is easy to see that Type 1 forgeries are Type 3 ones in the underlying scheme.

and supports general boolean circuits, and a second one that is secretly verifiable and supports low-degree arithmetic circuits. Although our work does not address directly the problem of privacy, extensions of our results along this direction are possible, and we leave the details to future investigation. A first extension is defining a notion of context-hiding for multi-key HAs. Similarly to the single key setting, this property should guarantee that authenticators do not reveal non-trivial information about the computation's inputs. The second extension has to do with preventing the Cloud from learning the data over which it computes. In this case, we note that multi-key HAs can be executed on top of homomorphic encryption following an approach similar to that suggested in [21]. Finally, an interesting problem left open by our work is to find multi-key HA schemes where authenticators have size independent of the number of users involved in the computation.

Acknowledgements. This work was partially supported by the COST Action IC1306 through a STSM grant to Elena Pagnin, the European Union's H2020 Research and Innovation Programme under grant agreement 688722 (NEXTLEAP), the Spanish Ministry of Economy under project reference TIN2015-70713-R (DEDETIS) and a Juan de la Cierva fellowship to Dario Fiore, and by the Madrid Regional Government under project N-Greens (ref. S2013/ICE-2731). This work was also partially supported by the the People Programme (Marie Curie Actions) of the European Union's Seventh Framework Programme (FP7/2007-2013) under REA grant agreement no 608743, the VR grant PRECIS no 621-2014-4845 and the STINT grant Secure, Private & Efficient Healthcare with wearable computing no IB2015-6001.

References

1. S. Agrawal, D. Boneh, X. Boyen, and D. M. Freeman. Preventing pollution attacks in multi-source network coding. In P. Q. Nguyen and D. Pointcheval, editors, *PKC 2010*, volume 6056 of *LNCS*, pages 161–176. Springer, Heidelberg, May 2010.
2. M. Ajtai. Generating hard instances of lattice problems (extended abstract). In *28th ACM STOC*, pages 99–108. ACM Press, May 1996.
3. M. Ajtai. Generating hard instances of the short basis problem. In *Automata, Languages and Programming*, pages 1–9. Springer, 1999.
4. J. Alwen and C. Peikert. Generating shorter bases for hard random lattices. *Theory of Computing Systems*, 48(3):535–553, 2011.
5. M. Backes, D. Fiore, and R. M. Reischuk. Verifiable delegation of computation on outsourced data. In A.-R. Sadeghi, V. D. Gligor, and M. Yung, editors, *ACM CCS 13*, pages 863–874. ACM Press, Nov. 2013.
6. S. Benabbas, R. Gennaro, and Y. Vahlis. Verifiable delegation of computation over large datasets. In P. Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 111–131. Springer, Heidelberg, Aug. 2011.
7. N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In D. Boneh, T. Roughgarden, and J. Feigenbaum, editors, *45th ACM STOC*, pages 111–120. ACM Press, June 2013.

8. D. Boneh, D. Freeman, J. Katz, and B. Waters. Signing a linear subspace: Signature schemes for network coding. In S. Jarecki and G. Tsudik, editors, *PKC 2009*, volume 5443 of *LNCS*, pages 68–87. Springer, Heidelberg, Mar. 2009.
9. D. Boneh and D. M. Freeman. Homomorphic signatures for polynomial functions. In K. G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 149–168. Springer, Heidelberg, May 2011.
10. D. Boneh, C. Gentry, S. Gorbunov, S. Halevi, V. Nikolaenko, G. Segev, V. Vaikuntanathan, and D. Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In P. Q. Nguyen and E. Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 533–556. Springer, Heidelberg, May 2014.
11. D. Catalano and D. Fiore. Practical homomorphic MACs for arithmetic circuits. In T. Johansson and P. Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 336–352. Springer, Heidelberg, May 2013.
12. D. Catalano, D. Fiore, R. Gennaro, and L. Nizzardo. Generalizing homomorphic MACs for arithmetic circuits. In H. Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 538–555. Springer, Heidelberg, Mar. 2014.
13. D. Catalano, D. Fiore, and L. Nizzardo. Programmable hash functions go private: Constructions and applications to (homomorphic) signatures with shorter public keys. In R. Gennaro and M. J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 254–274. Springer, Heidelberg, Aug. 2015.
14. D. Catalano, D. Fiore, and B. Warinschi. Adaptive pseudo-free groups and applications. In K. G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 207–223. Springer, Heidelberg, May 2011.
15. D. Catalano, D. Fiore, and B. Warinschi. Efficient network coding signatures in the standard model. In M. Fischlin, J. Buchmann, and M. Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 680–696. Springer, Heidelberg, May 2012.
16. D. Catalano, D. Fiore, and B. Warinschi. Homomorphic signatures with efficient verification for polynomial functions. In J. A. Garay and R. Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 371–389. Springer, Heidelberg, Aug. 2014.
17. S. G. Choi, J. Katz, R. Kumaresan, and C. Cid. Multi-client non-interactive verifiable computation. In A. Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 499–518. Springer, Heidelberg, Mar. 2013.
18. K.-M. Chung, Y. Kalai, and S. P. Vadhan. Improved delegation of computation using fully homomorphic encryption. In T. Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 483–501. Springer, Heidelberg, Aug. 2010.
19. Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM Journal on computing*, 38(1):97–139, 2008.
20. D. Fiore and R. Gennaro. Publicly verifiable delegation of large polynomials and matrix computations, with applications. In T. Yu, G. Danezis, and V. D. Gligor, editors, *ACM CCS 12*, pages 501–512. ACM Press, Oct. 2012.
21. D. Fiore, R. Gennaro, and V. Pastro. Efficiently verifiable computation on encrypted data. In G.-J. Ahn, M. Yung, and N. Li, editors, *ACM CCS 14*, pages 844–855. ACM Press, Nov. 2014.
22. D. Fiore, A. Mitrokotsa, L. Nizzardo, and E. Pagnin. Multi-key homomorphic authenticators. *IACR Cryptology ePrint Archive*, 2016.
23. D. M. Freeman. Improved security for linearly homomorphic signatures: A generic framework. In M. Fischlin, J. Buchmann, and M. Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 697–714. Springer, Heidelberg, May 2012.

24. K. O. Geddes, S. R. Czapor, and G. Labahn. *Algorithms for Computer Algebra*. Kluwer Academic Publishers, Norwell, MA, USA, 1992.
25. R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In T. Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 465–482. Springer, Heidelberg, Aug. 2010.
26. R. Gennaro and D. Wichs. Fully homomorphic message authenticators. In K. Sako and P. Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 301–320. Springer, Heidelberg, Dec. 2013.
27. C. Gentry. Fully homomorphic encryption using ideal lattices. In M. Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.
28. C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In R. E. Ladner and C. Dwork, editors, *40th ACM STOC*, pages 197–206. ACM Press, May 2008.
29. S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. Delegating computation: interactive proofs for muggles. In R. E. Ladner and C. Dwork, editors, *40th ACM STOC*, pages 113–122. ACM Press, May 2008.
30. S. Gorbunov, V. Vaikuntanathan, and D. Wichs. Leveled fully homomorphic signatures from standard lattices. In R. A. Servedio and R. Rubinfeld, editors, *47th ACM STOC*, pages 469–477. ACM Press, June 2015.
31. S. D. Gordon, J. Katz, F.-H. Liu, E. Shi, and H.-S. Zhou. Multi-client verifiable computation with stronger security guarantees. In Y. Dodis and J. B. Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 144–168. Springer, Heidelberg, Mar. 2015.
32. R. Johnson, D. Molnar, D. X. Song, and D. Wagner. Homomorphic signature schemes. In B. Preneel, editor, *CT-RSA 2002*, volume 2271 of *LNCS*, pages 244–262. Springer, Heidelberg, Feb. 2002.
33. D. Micciancio. Almost perfect lattices, the covering radius problem, and applications to Ajtai's connection factor. *SIAM Journal on Computing*, 34(1):118–169, 2004. Preliminary version in STOC 2002.
34. D. Micciancio and C. Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 700–718. Springer, Heidelberg, Apr. 2012.
35. D. Micciancio and C. Peikert. Hardness of SIS and LWE with small parameters. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 21–39. Springer, Heidelberg, Aug. 2013.
36. D. Micciancio and O. Regev. Worst-case to average-case reductions based on Gaussian measures. In *45th FOCS*, pages 372–381. IEEE Computer Society Press, Oct. 2004.
37. B. Parno, M. Raykova, and V. Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In R. Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 422–439. Springer, Heidelberg, Mar. 2012.
38. J. van der Hoeven and G. Lecerf. On the bit-complexity of sparse polynomial and series multiplication. *Journal of Symbolic Computation*, 50:227 – 254, 2013.