The Simplest Multi-key Linearly Homomorphic Signature Scheme

Diego F. Aranha[1,2] and Elena Pagnin[1]

[1] Aarhus University, Aarhus, Denmark
dfaranha@eng.au.dk, elena@cs.au.dk
[2] University of Campinas, Campinas, Brazil

Abstract. We consider the problem of outsourcing computation on data authenticated by different users. Our aim is to describe and implement the simplest possible solution to provide data integrity in cloud-based scenarios. Concretely, our multi-key linearly homomorphic signature scheme (mklhs) allows users to upload signed data on a server, and at any later point in time any third party can query the server to compute a linear combination of data authenticated by different users and check the correctness of the returned result. Our construction generalizes Boneh et al.'s linearly homomorphic signature scheme (PKC'09 [7]) to the multi-key setting and relies on basic tools of pairing-based cryptography. Compared to existing multi-key homomorphic signature schemes, our mklhs is a conceptually simple and elegant direct construction, which trades-off privacy for efficiency. The simplicity of our approach leads us to a very efficient construction that enjoys significantly shorter signatures and higher performance than previous proposals. Finally, we implement mklhs using two different pairing-friendly curves at the 128-bit security level, a Barreto-Lynn-Scott curve and a Barreto-Naehrig curve. Our benchmarks illustrate interesting performance trade-offs between these parameters, involving the cost of exponentiation and hashing in pairing groups. We provide a discussion on such trade-offs that can be useful to other implementers of pairing-based protocols.

Keywords: Multi-key homomorphic signatures, cryptographic pairings, efficient software implementation.

1 Introduction

Outsourcing tasks and data is an increasing need in today's society. A common paradigm is to collect data, store and process it on remote servers and finally return an aggregated result. As an example, consider a fitness program where devices upload user data to a remote server, a service provider computes on the outsourced data and informs users with statistics on their average heart-beat rates, running speed and so on. The very same pattern applies also to a wide range of data coming from medical, financial or general measurement sources. Recent scandals have taught us that users should not blindly rely on *the cloud* or *trust* service providers not to misbehave. Therefore, to fully enjoy

the benefits of cloud-based solutions users must be able to somehow "protect" the tasks and data they outsource. To this end, researchers developed specific tools for retaining privacy, ensuring correctness and other desirable properties, such as Private Information Retrieval (PIR), Verifiable delegation of Computations (VC), (Fully) Homomorphic Encryption (FHE) and Signatures (FHS), to mention a few.

This work focuses on a special subset of Homomorphic Signature schemes, namely Multi-Key Homomorphic Signatures (MKHS). In a nutshell, a MKHS scheme enables a set of multiple signers to independently authenticate their data, upload data and signatures to a remote server, let any third party (usually the server) carry out sensible computations and output a "combined" signature vouching for the correctness of the outcome of the computation, even when this was carried on data authenticated by different users. We remark that, in this model, after uploading the authenticated data, the signers are not required to interact with the cloud or the verifier. While MKHS do not guarantee privacy of the outsourced data, they target integrity of the information the server provides to the verifiers. In more detail, homomorphic signatures are a valid defense against malicious manipulation of data since they not only guarantee that the data used in the computation was authenticated by each signer involved but also that the output of the computation is correct. In other words, the "combined" signature can be used to verify that what the cloud outputs to the verifier is indeed the answer to the desired computation on the database (and not some random authenticated record). Existing MKHS constructions are based on lattice techniques [13], zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARK) [23], or creative yet convoluted compilers [14,26]. In this work, we take multi-key homomorphic signatures as a case of study and propose a new scheme that we consider conceptually simpler than all existing proposals.

Our Contribution. We aim to provide a MKHS scheme that is simpler to understand and thus can be seen as an easy way to introduce this fairly new research area. Our scheme, mklhs, allows to authenticate data signed by independent signers in a concise way and also to authenticate the result of a linear combination of data signed with different secret keys. While we are restricted to linear functions only, our signatures are shorter than all existing constructions. Despite the lack of full-fledged properties, such as full homomorphism or context hiding, mklhs has desirable characteristics including better succinctness and performance than previous proposals and relying entirely on standard assumptions (Random Oracle Model). Our scheme is inspired by the signature scheme by Boneh *et al.* [7] for signing a vector space in linear network coding scenarios. We remove the tools that are network-coding related and use a simple trick to obtain the multi-key features. While this technique is known and implicitly used in other works (*e.g.,* [13,26]), here we make a clear explanation of it and show that a natural application of this technique already brings with a non-trivial construction. Performance-wise, mklhs only requires operations in the base curve for computing signatures or evaluating functions over authenticated data, while signature verification requires a product of pairings. We implement mklhs using

pairings defined over elliptic curves using two sets of parameters at the current 128-bit security level: a Barreto-Lynn-Scott [5] curve with embedding degree 12 over a 381-bit prime field (BLS12-381) used in ZCash [25], and a Barreto-Naehrig [6] curve defined over a 382-bit field (BN-382). Our implementations illustrate trade-offs among parameters which can be useful to other instantiations of pairing-based protocols. To the best of our knowledge, this constitutes the first concrete implementation of multi-key homomorphic signature schemes.

Related Work. The notion of homomorphic signature schemes (HS) was introduced by Johnson *et al.* in [20], together with a security model for HS and a concrete construction for redactable signatures. Intuitively, this first example of HS allows to erase part of the message as an homomorphic operation: given a message m and a signature σ of m, anyone can derive from σ a new signature σ' for any message m' obtained after redacting m. Subsequent proposals extended the kind of operations supported by HS. The main bulk of work comes with constructions of linearly homomorphic signatures for linear network coding [3,7,8,10]. More recently, Catalano *et al.* [11] address the question of HS for higher degree functions and show applications to efficient verifiable computation of polynomial functions. The first construction of (leveled) fully homomorphic signatures (FHS) is due to Gorbunov *et al.* [18]. The scheme presented in [18] is a lattice-based HS capable of evaluating arbitrary boolean circuits of bounded polynomial depth over signed data. However, none of the aforementioned schemes support computations on data signed by multiple clients.

Agrawal *et al.* [1] expand the horizon of applications by considering *multi-source* signatures in the context of network coding. A few years later, Fiore *et al.* [13] formalize the concept of multi-key homomorphic authenticators and provided the first constructions of a multi-key homomorphic signature scheme (MKHS) and of a multi-key homomorphic MAC. The MKHS in [13] is designed for boolean circuits of bounded depth and can be seen as a (non-trivial) extension of the FHS scheme in [18]. A limitation of this work is that the proposed MKHS scheme is not unforgeable if corrupted parties take part to the computation. Lai *et al.* [23] address this issue and show that MKHS unforgeable under corruption can be constructed using zk-SNARKs and indeed must rely on non-falsifiable assumptions. Fiore and Pagnin [14] put forward a generic compiler to empower any single-key FHS with multi-key features without adding any security assumptions on top of the ones required by the base scheme. While the intuition behind this compiler is quite simple, its formal description gets quickly entangled. Schabhüser *et al.* [26] recently proposed another generic compiler to obtain multi-key *linearly* homomorphic authenticators from any signature or MAC scheme. Their technique relies on asymmetric bilinear groups, is based on standard assumptions only, and achieves the non-trivial property of context hiding. In a nutshell, homomorphic signatures that are context hiding guarantee the privacy of the data input to the homomorphic evaluation, as evaluated authenticators do not leak any information about the input messages.

Our scheme provides the first direct construction of a multi-key linearly homomorphic signature scheme that is not based on a compiler. Albeit lacking

context hiding, our proposal is the simplest and most intuitive HS to enjoy multi-key features up to date. Suitable application scenarios are contexts involving public data where authenticity is preferable to confidentiality, e.g., processing pollution values recorded by sensors scattered around the city, simple statistic on public data such as weather forecast, among others. Moreover, compared to [26], we achieve concretely better succinctness: while the asymptotic bound is the same for both ($\mathcal{O}(t)$, for t signers involved in the computation), the constants hidden by the big-oh notation differ considerably. In terms of performance, our implementations show that mklhs outperforms [26] in all protocol operations (see Section 4 for further details).

2 Preliminaries

In this section, we recall the fundamental definitions of bilinear pairings and multi-key homomorphic signatures. In what follows, we use *choosing at random* or *randomly choosing* to refer to sampling from the given set according to the uniform distribution. In addition, we write $\mathsf{poly}(\lambda)$ to denote a polynomial function in the variable λ, and ε a negligible function (*i.e.,* $\varepsilon(\lambda) < 1/\mathsf{poly}(\lambda)$, for any function poly and large enough values of λ).

2.1 Bilinear Pairings and Security Assumptions

An *admissible bilinear pairing* is a non-degenerate efficiently-computable map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ defined over groups of prime order q. For efficiency, we assume here an *asymmetric* pairing constructed over an ordinary pairing-friendly curve with embedding degree k. In practice, groups $\mathbb{G}_1$ and $\mathbb{G}_2$ are then chosen as subgroups of points in elliptic curves E and d-degree twist E' defined over a finite field $\mathbb{F}_p$ and its extension $\mathbb{F}_{p^{k/d}}$, respectively, and $\mathbb{G}_T$ is a subgroup of the multiplicative group of the related finite field $\mathbb{F}^*_{p^k}$.

The core property of the map e is linearity in both arguments, allowing the construction of novel cryptographic schemes with security relying on the hardness of the *Discrete Logarithm Problem* (DLP) in $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$. Security of pairing-based protocols typically relies directly or indirectly on the hardness of solving the *Bilinear Diffie-Hellman* problem (BDHP) of computing $e(g,h)^{abc}$ given $g, g^a, g^b, g^c, h, h^a, h^b, h^c$ for $g \in \mathbb{G}_1$, $h \in \mathbb{G}_2$ and $a, b, c \in \mathbb{Z}^*_q$. The security of our particular scheme depends on the hardness of the *co-Computational Diffie-Hellman* problem (co-CDH) in the bilinear setting:

Definition 1 (co-CDH). *Given $g \in \mathbb{G}_1$ and $h, h^x \in \mathbb{G}_2$, compute $g^x \in \mathbb{G}_1$.*

2.2 Multi-Key Homomorphic Signatures

The seminal notions of homomorphic signatures consider only authenticated data and a function to be evaluated on the data and the signatures. While this is a direct approach, more recent work on homomorphic authenticators and

especially on multi-key variations thereof points out the need to authenticate data "in a context". Intuitively, we are not interested in verifying that a certain value m is the output of function f on *some* authenticated data, but rather that m is the output of f on *precisely* the data asked to be computed on. This linkability property allows us to put the values into a well-defined context. The cryptographic artifact that is used to formalize the intuition above is called a *labeled program*.

Definition 2 (Labeled Programs [17]). *A labeled program $\mathcal{P}$ is a tuple of form $(f, \ell_1, \ldots, \ell_n)$ where f is a function $f : \mathcal{M}^n \to \mathcal{M}$ that takes as input n messages and returns a single value, while $\ell_1, \ldots, \ell_n \in \{0,1\}^*$ are the n labels that identify each input of f.*

In this work, we follow the mainstream approach to include the identity of users in the labels. A meaningful definition of multi-key HS requires the signatures to be verified with respect to the set of keys used to sign the inputs to the computation. Our labels are of the form $\ell = (\mathsf{id}, \tau)$ where id is a user's identifier (we often refer to it as identity) and τ is a tag, a string used to uniquely identify a data item from user id. As a general rule, n denotes the number of inputs to the labeled program (essentially the number of messages involved in the computation), while t denotes the number of distinct entities contributing with data to the computation. Therefore it holds that $n \geq t$. For further details we refer the reader to [13,14,17].

Next, we present the definition of multi-key homomorphic signatures. For completeness, we include dataset identifiers Δ in the general definitions, even though these will be dropped in our construction (where they can trivially be included into the tags as shown in [13]). Intuitively, dataset identifiers enable users to authenticate data on different databases, or to sign a new message under the same label (but for a different Δ).

Definition 3 (Multi-Key Homomorphic Signatures [13]). *A multi-key homomorphic signature scheme* MKHS *is a tuple of five PPT algorithms* $\mathsf{MKHS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Eval}, \mathsf{Verify})$ *defined as follows.*

$\mathsf{Setup}(1^\lambda)$*: The setup algorithm takes as input the security parameter and outputs some public parameters* pp *including a description of an identity space* ID*, a tag space $\mathcal{T}$ a message space $\mathcal{M}$ and a set of admissible functions $\mathcal{F}$.*

$\mathsf{KeyGen}(\mathsf{pp})$*: The key generation algorithm takes as input the public parameters and outputs a pair of keys (*sk,pk*) to which is associated an identity* id*.*

$\mathsf{Sign}(\mathsf{sk}, \Delta, \ell, m)$*: The sign algorithm takes as input a secret key* sk*, a dataset identifier Δ, a label $\ell = (\mathsf{id}, \tau)$ for the message m, and it outputs a signature σ.*

$\mathsf{Eval}(\mathcal{P}, \{\mathsf{pk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}}, \sigma_1, \ldots, \sigma_n)$*: The evaluation algorithm takes as input a labeled program $\mathcal{P} = (f, \ell_1, \ldots, \ell_n)$ a set of public-keys and signatures, and outputs an homomorphic signature σ.*

$\mathsf{Verify}(\mathcal{P}, \Delta, \{\mathsf{pk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}}, m, \sigma)$*: The verification algorithm takes as input a labeled program $\mathcal{P}$, a dataset identifier Δ, the set of public keys defined by the labels in $\mathcal{P}$, a message m and an homomorphic signature σ. It outputs 0 (reject) or 1 (accept).*

In this work, we consider a special family of MKHS, namely schemes that support evaluation of solely linear functions. When referring to MKHS schemes we give for granted that the construction satisfies the properties of authentication correctness, evaluation correctness and compactness described below.

Authentication Correctness. Intuitively, this property states that every signature output by the Sign algorithm for a message m and label ℓ verifies successfully, with overwhelming probability, against m, ℓ and labeled program $\mathcal{I}$ corresponding to the identity function for label ℓ. Formally, a multi-key homomorphic signature satisfies authentication correctness if for all public parameters $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, any database identifier Δ, any key pair $(\mathsf{sk}_{\mathsf{id}}, \mathsf{pk}_{\mathsf{id}}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$, any label $\ell = (\mathsf{id}, \tau) \in \mathsf{ID} \times \mathcal{T}$, any message $m \in \mathcal{M}$ and any signature $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, \Delta, \ell, m)$, it holds that

$$\Pr\left[\mathsf{Verify}(\mathcal{I}_\ell, \Delta, \mathsf{pk}, m, \sigma) = 1\right] \geq 1 - \varepsilon \ .$$

Evaluation Correctness. Intuitively, this property states that if the signatures input to the Eval algorithm satisfy authentication correctness, then Eval outputs signatures that, with overwhelming probability, verify successfully for the appropriate value m and labeled program $\mathcal{P}'$ (seen as the composition of multiple labeled programs). Formally, a multi-key homomorphic signature satisfies evaluation correctness if

$$\Pr\left[\mathsf{MKHS.Verify}(\mathcal{P}', \Delta, \{\mathsf{pk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}'}, m', \sigma') = 1\right] \geq 1 - \varepsilon \ ,$$

where the equality holds for a fixed description of the public parameters $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, an arbitrary set of honestly generated keys $(\mathsf{sk}_{\mathsf{id}}, \mathsf{pk}_{\mathsf{id}}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$, a function $f : \mathcal{M}^n \to \mathcal{M}$, any dataset Δ, and any set of program/message/signature triples $\{(\mathcal{P}_i, m_i, \sigma_i)\}_{i\in[n]}$ such that $\mathsf{Verify}(\mathcal{P}_i, \Delta, \{\mathsf{pk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}_i}, m_i, \sigma_i) = 1$ for all $i \in [n]$, and $m' = g(m_1, \ldots, m_n)$, $\mathcal{P}' = g(\mathcal{P}_1, \ldots, \mathcal{P}_n)$, and $\sigma' \leftarrow \mathsf{Eval}(\mathcal{P}, \{\mathsf{pk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}_i}, \{\sigma_i\}_{i\in[n]})$.

Succinctness. This is a crucial property for MKHS as it rules out trivial constructions. One could define a MKHS scheme by using a standard (non-homomorphic) signature scheme, and set the Eval procedure to simply append the input signatures (and corresponding messages). Now, all the workload drops on the Verify algorithm, that checks the authentication correctness of each individual signature and its message, and afterwards performs the desired computation on the authenticated messages. What makes such a solution unattractive is that (i) the verifier needs to check the validity of n signatures (where n is the number of inputs to the function it wishes to evaluate) and (ii) the verifier essentially needs to compute the function itself. The succinctness property guarantees that the verifier only needs to perform *one* signature verification. Intuitively, this means that the size of the signature output by Eval for a labeled program $\mathcal{P}$ should be significantly smaller than n, the input size of $\mathcal{P}$, concretely, it should depend at most logarithmically in n, and linearly in the number of signers involved in the

computation. More formally, a multi-key homomorphic signature is succinct if the signature $\sigma \leftarrow \mathsf{Eval}(\mathcal{P}, \{\mathsf{pk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}_i}, \{\sigma_i\}_{i\in[n]})$ has size $|\sigma| = \mathsf{poly}(\lambda, t, \log n)$ where λ denotes the security parameter of the scheme, $t = |\mathsf{id} \in \mathcal{P}|$ denotes the number of distinct identities involved in the computation and n is the total number of inputs to $\mathcal{P}$.

Security. Finally, we present the security notion for multi-key homomorphic signatures. Our definition is equivalent to the one provided by Fiore *et al.* [13], however, we split the authentication query phase into two phases: a identity query one and a sign phase to improve readability and avoid long listing subcases.

Definition 4 (Homomorphic Unforgeability under Chosen Message Attack security experiment – HUFCMA).

Setup. *The challenger $\mathscr{C}$ runs $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ and returns pp to the adversary $\mathscr{A}$. In addition, $\mathscr{C}$ initiates three empty lists, the first one to keep track of already generated identities $L_{ID} \leftarrow \varnothing$, the second one for bookkeeping the dataset/labels/messages that will sign during the game $L_\sigma \leftarrow \varnothing$, and the third one for corrupted identities $L_{\mathsf{corr}} \leftarrow \varnothing$.*

Identity Queries. *The adversary can adaptively submit identity queries of the form $\mathsf{id} \in \mathsf{ID}$. Whenever $\mathsf{id} \notin L_{ID}$, it means that this is the first query with identity id, and $\mathscr{C}$ generates new keys for this identity by running $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$. Then it numbers the identity as id_d, where $d = |L_{ID}| + 1$, and adds id_d to the list of already generated identities $L_{ID} \leftarrow L_{ID} \cup \{(\mathsf{id}, d, \mathsf{sk}, \mathsf{pk})\}$. Finally, the challenger returns to $\mathscr{A}$ the public key pk. Whenever $\mathsf{id} \in L_{ID}$ the challenger interprets the query as a corruption query, so it updates the list of corrupted parties $L_{\mathsf{corr}} \leftarrow L_{\mathsf{corr}} \cup \mathsf{id}$ and retrieves the handle of information containing id from L_{ID}, i.e., $(\mathsf{id}, d, \mathsf{sk}, \mathsf{pk})$. Finally, the challenger returns to $\mathscr{A}$ the secret key sk corresponding to the queried identity.*

Sign Queries. *The adversary can adaptively submit queries of the form (Δ, ℓ, m), where Δ is a database identifier, $\ell = (\mathsf{id}, \tau)$ is a label in $\mathsf{ID} \times \mathcal{T}$ and $m \in \mathcal{M}$ is a message. The challenger ignores the query whenever $(\Delta, \ell, \cdot) \in L_\sigma$, i.e., the adversary has already asked a signature for label ℓ in the dataset Δ; or $(\mathsf{id}, \cdots) \notin L_{ID}$, i.e., the identity specified in $\ell = (\mathsf{id}, \tau)$ has not yet been generated. Otherwise, the challenger computes $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_{\mathsf{id}}, \Delta, \ell, m)$, updates the list $L_\sigma \leftarrow L_\sigma \cup (\Delta, \ell, m)$ and returns σ to the adversary.*

Forgery. *At the end of the game, the adversary $\mathscr{A}$ outputs a tuple*

$$(\mathcal{P}^*, \Delta^*, \{\mathsf{pk}^*_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}^*}, m^*, \sigma^*).$$

The experiment outputs $\mathsf{1}$ if the tuple returned by $\mathscr{A}$ is a forgery (defined below), and $\mathsf{0}$ otherwise.

Definition 5. *A* MKHS *scheme is said to be* unforgeable *if, for every PPT adversary $\mathscr{A}$, its advantage in winning the security game (HUFCMA) described before is negligible in the security parameter of the scheme, formally:*

$$\mathsf{Adv}_{\mathsf{MKHS},\mathscr{A}}[\lambda] = \Pr[\mathscr{A} \text{ wins the HUFCMA game for } \mathsf{MKHS}(\lambda) = 1] \leq \varepsilon.$$

Definition 6 (Forgery). *We consider an execution of HUFCMA where* $(\mathcal{P}^*, \Delta^*, \{\mathsf{pk}^*_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}^*}, m^*, \sigma^*)$ *is the tuple returned by* $\mathscr{A}$ *at the end of the experiment. Let* $\mathcal{P}^* = (f^*, \ell^*_1, \ldots, \ell^*_n)$*. The adversary outputs a successful forgery if*

$$\mathsf{Verify}(\mathcal{P}^*, \Delta^*, \{\mathsf{pk}^*_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}^*}, m^*, \sigma^*) = 1$$

and at least one of the following conditions hold:

Type-1 forgery*: the database* Δ^* *was never initialized during the game, i.e.,* $(\Delta^*, \cdot, \cdot) \notin L_\sigma$*.*

Type-2 forgery*: for all* $\mathsf{id} \in \mathcal{P}^*$*,* $\mathsf{id} \notin L_{\mathsf{corr}}$ *and* $(\Delta^*, \ell^*_i, m_i) \in L_\sigma$ *for all* $i \in [n]$*, but* $m^* \neq f^*(m_1, \ldots, m_n)$*.*

Type-3 forgery*: there exists (at least) one index* $i \in [n]$ *such that* ℓ^*_i *was never queried, i.e.,* $(\Delta^*, \ell^*_i, \cdot) \notin L_\sigma$ *and* $\mathsf{id}_i \notin L_{\mathsf{corr}}$ *is a non-corrupted identity.*

In all forgery types, the adversary tampers with the result of the computation by creating a signature σ^* that verifies an incorrect or un-initialized value m^*. More precisely, type-1 forgeries model cross-database attacks and are relevant only to constructions that consider multiple datasets. Type-2 forgeries model attack scenarios where the adversary intents to authenticate a value m^* that is not the correct output of $\mathcal{P}^*$ when executed over previously signed data. It is important to notice that all label-message pairs have been queried during the game. In contrast, type-3 forgeries model attack scenarios where at least one of the inputs to the labeled program $\mathcal{P}^*$ has never been initializated during the security game. In such cases, the adversary has no access to the signature of a message for a certain label ℓ^*_i. Thus, $\mathscr{A}$ has the freedom to choose a value m^*_i for the i-th input to the computation, conditioned to forging a valid signature for (ℓ^*_i, m^*_i).

In this work we want to provide a construction of a multi-key linearly homomorphic signature scheme that is as simple and intuitive as possible. To this end, we do not consider multiple datasets, and thus ignore database identifiers and type-1 forgeries.

3 Our Construction

We propose mklhs, a linearly homomorphic signature scheme that supports computations on data authenticated by different secret keys. Compared to existing constructions [26], our proposal is conceptually simpler and more direct.

Our scheme is inspired by the linearly homomorphic signature scheme by Boneh *et al.* [7] for signing vector spaces in the context of linear network coding. Concretely, we remove all of the network-coding related machinery from [7] and modify the resulting scheme to accommodate for homomorphic computations on signatures generated using different secret keys. The joint signature results unforgeable under the co-CDH assumption and achieves better succinctness (in terms of number of entries) than [26] and [13]. Similarly to [7], we work with Type-2 pairings for clarity and convenience. This gives us a homomorphism φ

from $\mathbb{G}_2$ to $\mathbb{G}_1$ that plays a central role in our security proof. However, standard tools from the literature can be used to adapt the security proof to the more efficient Type-3 pairing setting by tweaking the hardness assumption [12]. A hash function $H : \{0,1\}^* \to \mathbb{G}_1$ will also be needed to embed labels in the signatures.

3.1 Intuition

The main algorithms of our construction are detailed in Figure 1. In what follows, we provide an intuitive description. Our signatures are obtained by hashing the label corresponding to the desired message to a group element in $\mathbb{G}_1$. Subsequently, we multiply this group element by the generator of the elliptic curve group to the power of the message. Finally, we wrap the result by exponentiating to the secret key. For security, we also need to append the message to the signatures. While this approach precludes us from achieving context hiding features, it does not go against the basic requirements of a signature scheme, namely guaranteeing data integrity and not confidentiality. The homomorphic evaluation of signatures follows the same aggregation style as [7], with an additional routine to take care of the second component of the input signatures. In detail, for every distinct identity that contributed to the computation with a signature, we identify all of the partial inputs of this identity and homomorphically combine them into one single component. Finally, for verification, we perform two consistency checks: first we make sure that the contribution provided by each signer actually adds up to all the target message (line 3); then we check that each signer's contribution verifies, in a batch manner (lines 5 and 6).

The **Organize** subroutine used by the evaluation and verification algorithms is formalized in the pseudocode to the right. Given as input a set of elements each containing an identity identifier (*e.g.,* the signatures or the labels), it identifies what are the different identities, re-labels them according to the order of appearance, and outputs the ordered set of identities as well as sets of indexes corresponding to the element connected to each identity. Essentially, it translates statements like "without loss of generality, we assume that the input signatures are grouped per-user and that the identities involved have indexes from 1 to t".

Organize$(a_1, \dots, a_n)$

1 : $\mathsf{OrdID} \leftarrow \varnothing, j \leftarrow 0, t \leftarrow 0$
2 : **for** $i \in [n]$
3 : **parse** $a_i = (\mathsf{id}_i, *)$
4 : **if** $\mathsf{id}_i \notin \mathsf{OrdID}$
5 : $j \leftarrow |\mathsf{OrdID}| + 1$
6 : $\tilde{\mathsf{id}}_j \leftarrow \mathsf{id}_i$
7 : $\mathsf{OrdID} \leftarrow \mathsf{OrdID} \cup \{\tilde{\mathsf{id}}_j\}$
8 : $t \leftarrow |\mathsf{OrdID}|$
9 : **for** $j \in [n]$
10 : $\mathsf{I}_j \leftarrow \varnothing$
11 : **if** $\mathsf{id}_i == \tilde{\mathsf{id}}_j$
12 : $\mathsf{I}_j \leftarrow \mathsf{I}_j \cup \{i\}$
return $(t, \mathsf{OrdID}, \mathsf{I}_1, \dots, \mathsf{I}_t)$.

3.2 Security Analysis

The security of our scheme is supported by the main theorem below.

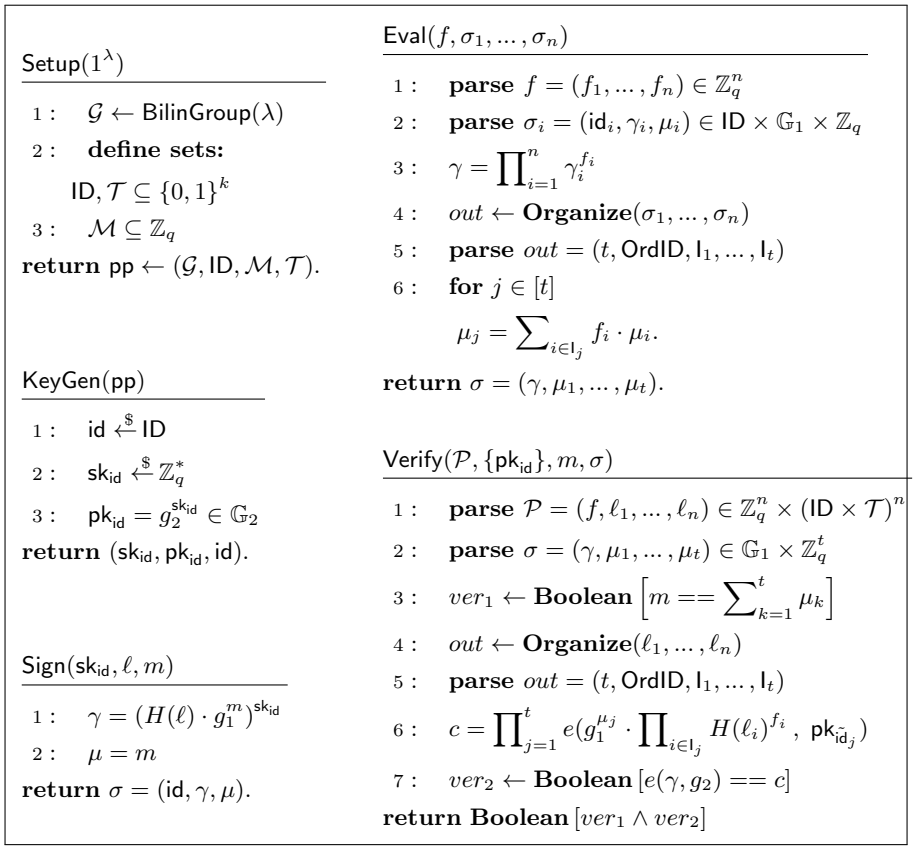

Fig. 1. Our mklhs construction.

Theorem 1. *The* mklhs *scheme is secure in the random oracle model assuming that the co-CDH problem is computationally infeasible. In detail, let $\mathscr{A}$ be a probabilistic polynomial-time adversary in the security experiment (HUFCMA) described in Section 2.2, then its advantage is bounded by*

$$\mathsf{Adv}_{\mathsf{mklhs},\mathscr{A}}[\lambda] \leq \frac{1}{2} \cdot \left[R_H + Q_{\mathsf{id}} \cdot \mathsf{Adv}_{\mathscr{B}}^{\mathsf{coCDH}[\lambda]}\right].$$

where $\mathscr{B}$ is a polynomial-time algorithm that solves a co-CDH instance with probability $\mathsf{Adv}_{\mathscr{B}}^{\mathsf{coCDH}}[\lambda]$, $R_H = \frac{1}{\mathsf{poly}(\lambda)}$ is determined by the prime number q corresponding to the order of the group $\mathbb{G}_1$ (the range of the hash function) and $Q_{\mathsf{id}} = \mathsf{poly}(\lambda)$ is the total number of identities generated during the game.

The proof flow works as follows. We begin by ruling out corruption queries using the generic result by Fiore *et al.* on the equivalence between multi-key homomorphic authenticators secure against adversaries that make *no corruption* queries and secure against adversaries that make *non-adaptive* corruption queries (Proposition 1 in [13]). Next, we consider an adversary that outputs

type-3 forgeries and show that it can only succeed with a negligible probability of $R_H = \frac{1}{q} = \frac{1}{\mathsf{poly}(\lambda)}$ corresponding to randomly guessing $H(\ell^*)$, the output of the hash function on the un-queried label. Finally, we exhibit a reduction from type-2 forgeries to the co-CDH problem that combines techniques for multi-key settings with a clever embedding of the co-CDH challenge into our signature scheme. In particular, modulo the multi-key factor, our reduction is conceptually simpler and more efficient (less probability to abort) than the one used in [7]. This improvement is mainly due to a technical choice: separating hash queries from sign queries, namely our adversary can perform hash queries only after the sign-query phase is over. In this way, we avoid to abort every time the reduction could not program the hash function (as it happens in [7]).

In light of the generic equivalence stated by Fiore *et al.* in Proposition 1 [13], we ignore corruption queries during the security game. Therefore, our initial game is the security game between the adversary $\mathscr{A}$ the challenger $\mathscr{C}$ described at the end of Section 2.2, where no corruptions are allowed, hash queries happen after the signing query phase and forgeries follow the Definition 6. We prove the security of our scheme in 2 steps:

(1) we bound the success probability of any PPT adversary that outputs **type-3** forgeries to $\frac{1}{q}$;

(2) for any PPT adversary that outputs **type-2** forgeries we show a reduction to a co-CDH instance with factor $\frac{1}{Q_{\mathsf{id}}}$ where Q_{id} is the total number of identities generated during the game (this loss is common to all multi-key homomorphic signatures schemes to date [13,26]).

Type-3 forgeries. Let $\mathscr{A}$ be a **type-3** forger. By definition, the adversarial output $(\mathcal{P}^*, m^*, \sigma^*)$ contains a label $\ell^* \in \mathcal{P}^*$ for which no signature or hash query has been performed. This means that in order to verify $(\mathcal{P}^*, m^*, \sigma^*)$ the challenger needs to generate a value for $H(\ell^*)$ on-the-fly. Since $\mathscr{C}$ acts as a Random Oracle on hash queries, the probability that the adversary outputs a valid type-3 forgery is at most equal to the probability of randomly guessing the value $H(\ell^*) \in \mathbb{G}_1$, *i.e.,* R_H. Thus

$$\mathsf{Prob}[\mathscr{A} \text{ outputs a valid type-3 forgery }] \leq R_H = \frac{1}{q}.$$

Given that $|\mathbb{G}_1| = q$ is a $\mathsf{poly}(\lambda)$-bit prime, we can make the above probability arbitrarily small.

Type-2 forgeries. Now consider the case of $\mathscr{A}$ be a **type-2** forger. We define a reduction $\mathscr{B}$ that turns any type-2 forger (against a specific identity) into solutions of an instance of the co-CDH problem. We recall that we require $\mathscr{A}$ to perform all sign queries before any hash query. In particular, after the first hash query $\mathscr{A}$ is no longer allowed to request new signatures.

Concretely, $\mathscr{B}$ takes as input a bilinear group $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, e, \varphi)$, two generators $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$ and a point $h = g_2^x \in \mathbb{G}_2$. The goal of the reduction

is to output an element $\omega \in \mathbb{G}_1$ such that $\omega = g_1^x$. Algorithm $\mathscr{B}$ simulates Setup, KeyGen, Sign and the hash function H of mklhs as follows.

Setup: $\mathscr{B}$ utilizes the group homomorphism φ to identify the image of g_2 in $\mathbb{G}_1$, *i.e.,* $\varphi(g_2) \in \mathbb{G}_1$. Then, $\mathscr{B}$ sets $\tilde{g}_1 = g_1^{\tilde{s}}\varphi(g_2)^{\tilde{t}}$ for randomly chosen $\tilde{s}, \tilde{t} \xleftarrow{\$} \mathbb{Z}_q$, and outputs the bilinear group $\mathcal{G}$ and the generators $\tilde{g}_1, g_2$. In addition, $\mathscr{B}$ chooses an index $\tilde{j} \xleftarrow{\$} [Q_{\mathsf{id}}]$ as a guess that $\mathscr{A}$ will make a type-2 forgery against the $\tilde{j}$-th identity in the system. We will refer to this special identity as the *target* identity. Finally, $\mathscr{B}$ initializes three empty lists L_{ID}, L_H and L_σ to track $\mathscr{A}$'s identity, hash and signature queries respectively.

Identity query: on input a query of the form $\mathsf{id} \in \mathsf{ID}$, $\mathscr{B}$ checks if $\mathsf{id} \in L_{ID}$ in which case it ignores the query. Otherwise, this is the first query with identity id. Let $i-1$ be the number of identities already present in L_{ID}, if $i \in [Q_{\mathsf{id}}] \setminus \tilde{j}$ the reduction produces a fresh pair of keys, when $i = \tilde{j}$ the reduction embeds h as the challenge public key. In detail, for $i \neq \tilde{j}$ the reduction creates a new user by choosing a random $\mathsf{sk}_i \xleftarrow{\$} \mathbb{Z}_q$, adding $(\mathsf{id}, i, \mathsf{sk}_i, \mathsf{pk}_i = g_2^{\mathsf{sk}_i})$ to L_{ID} and returning $\mathsf{pk}_{\mathsf{id}}$. For $i = \tilde{j}$ the reduction stores $(\tilde{\mathsf{id}}, \tilde{j}, \cdot, \mathsf{pk}_{\tilde{j}} = h)$ in L_{ID} and returns its co-CDH challenge piece $h \in \mathbb{G}_2$.

Sign query: on input a query of the form (ℓ, m), $\mathscr{B}$ checks whether $\mathscr{A}$ already queried the label (possibly on a different message), *i.e.,* $(\ell, \cdot) \in L_\sigma$, or the identity in $\ell = (\mathsf{id}, \tau)$ has not yet been generated, *i.e.,* $(\mathsf{id}, \cdot, \cdot, \cdot) \notin L_{ID}$. The reduction ignores the query if any of the two conditions above is met. Otherwise, there are three possible scenarios:

(a) this is the first sign query for the identity id and $\mathsf{id} \neq \tilde{\mathsf{id}}$,

(b) identity $\mathsf{id} \neq \tilde{\mathsf{id}}$ was already queried in combination to another label ℓ',

(c) the queried identity is the target identity, *i.e.,* $\mathsf{id} = \tilde{\mathsf{id}}$.

So long $i \neq \tilde{j}$, the reduction can retrieve $(\mathsf{id}, i, \mathsf{sk}_i, \mathsf{pk}_i)$ from L_{ID} and sign in a perfect manner using sk_i. Concretely, in case **(a)** this is the first sign query for the identity id, $\mathscr{B}$ chooses two random values $s_{\mathsf{id}}, t \xleftarrow{\$} \mathbb{Z}_q$ and sets

$$H(\ell) = g_1^{-s_{\mathsf{id}} \cdot m} \cdot \varphi(g_2)^t \tag{1}$$

$$\gamma = (H(\ell) \cdot \tilde{g}_1^m)^{\mathsf{sk}_i} \tag{2}$$

$$\mu = m. \tag{3}$$

In case **(b)**, $\mathscr{B}$ retrieves the identity's s_{id} value from an existing record in L_H with $\ell' = (\mathsf{id}, \tau')$ and computes (1), (2), (3) for a random $t \xleftarrow{\$} \mathbb{Z}_q$. In case **(c)**, *i.e.,* when $i = \tilde{j}$, the reduction can perfectly simulate the signature thanks to the special generator $\tilde{g}_1$ created in the setup phase. In detail, $\mathscr{B}$ chooses a random $t \xleftarrow{\$} \mathbb{Z}_q$, uses $\tilde{s}$ as the target identity's "random" $s_{\tilde{\mathsf{id}}}$ value and computes

$$H(\ell) = g_1^{-\tilde{s} \cdot m} \cdot \varphi(g_2)^t \tag{4}$$

$$\gamma = \varphi(h)^{t + \tilde{t} \cdot m} \tag{5}$$

$$\mu = m. \tag{6}$$

where we recall that $h = \mathsf{pk}_{\tilde{j}} \in \mathbb{G}_2$ was given to $\mathscr{B}$ by the co-CDH challenger.

In all cases, $\mathscr{B}$ does some bookkeeping by storing $(\ell, s_{\mathsf{id}}, t)$ in L_H and (ℓ, m, σ) in L_σ. Finally, $\mathscr{B}$ returns $\sigma = (\gamma, \mu)$ as the answer to $\mathscr{A}$'s sign query.

Hash query: let (ℓ) denote $\mathscr{A}$'s input to the hash oracle with $\ell = (\mathsf{id}, \tau)$. There are three possible scenarios:

(a) this is the first hash query for the identity id and $\mathsf{id} \neq \tilde{\mathsf{id}}$,

(b) identity $\mathsf{id} \neq \tilde{\mathsf{id}}$ was already queried in combination to another label ℓ',

(c) the queried identity is the target identity, *i.e.,* $\mathsf{id} = \tilde{\mathsf{id}}$.

In the first case, **(a)**, our reduction chooses two random values $s_{\mathsf{id}}, t \xleftarrow{\$} \mathbb{Z}_q$ and sets $H(\ell) = g_1^{-s_{\mathsf{id}} \cdot m} \varphi(g_2)^t$. To ensure consistency, in case **(b)**, our reduction retrieves s_{id} from any of the previous hash queries on id present in L_H and computes $H(\ell) = g_1^{-s_{\mathsf{id}} \cdot m} \varphi(g_2)^t$ for a random $t \xleftarrow{\$} \mathbb{Z}_q$. In the last case, **(c)**, our reduction uses the randomness generated during the setup phase and sets $s_{\tilde{\mathsf{id}}} = \tilde{s}$, $t \xleftarrow{\$} \mathbb{Z}_q$ and computes $H(\ell) = g_1^{-\tilde{s} \cdot m} \varphi(g_2)^t$.

In all cases, $\mathscr{B}$ stores $(\ell, s_{\mathsf{id}}, t)$ in L_H and returns $H(\ell) \in \mathbb{G}_1$.

It is easy to see that $\mathscr{B}$ simulates the security game in a perfect way, as we show at the end of this section. Assuming for now that the simulation works fine, we demonstrate how to extract a solution to the co-CDH problem from a type-2 forgery. Let $(\mathcal{P}^*, m^*, \sigma^*)$ be the output of the algorithm $\mathscr{A}$ at the end of its interaction with $\mathscr{B}$. Since we are dealing with a type-2 forger it must be the case that the labeled program $\mathcal{P}^* = (f^*, \ell_1, \ldots, \ell_n)$ is well-defined and

$$\begin{cases} \mathsf{Verify}(\mathcal{P}^*, \{\mathsf{pk}_i\}_{i \in \mathcal{P}^*}, m^*, \sigma^*) = 1 \\ m^* \neq f^*(m_1, \ldots, m_t) \end{cases}$$

where m_i is the message queried by $\mathscr{A}$ for the label ℓ_i during the interaction with $\mathscr{B}$. Without loss of generality, we assume the identities involved in $\mathcal{P}^*$ be $\mathsf{id}_1, \mathsf{id}_2, \ldots, \mathsf{id}_t$ and the messages be ordered per-party. In particular, the first $k_1 > 0$ messages belong to id_1, the subsequent $k_2 > 0$ messages belong to id_2 and so on, so that $\sum_{i=1}^{t} k_i = n$ (the number of inputs to f). Define f_i^* to be the labeled program $\mathcal{P}^*$ restricted to the inputs (labels) of identity id_i, so that $f^* = f_1^* + f_2^* + \ldots + f_t^*$ and $k = \sum_{i=1}^{k_{\tilde{j}-1}} k_i$ to be the last index before the messages by $\mathsf{id}_{\tilde{j}}$ are input. Our reduction $\mathscr{B}$ looks for type-2 forgeries against the user $\mathsf{id}_{\tilde{j}}$, i.e, forgeries that satisfy

$$\begin{cases} \mathsf{id}_{\tilde{j}} \in \mathcal{P}^* \\ \mu_{\tilde{j}} \neq f_{\tilde{j}}^*(m_{k+1}, \ldots, m_{k+k_{\tilde{j}}}) \end{cases}$$

The above conditions ensure that the identity $\tilde{j}$ used to embed the co-CDH challenge is present in the labeled program *and* that the corresponding entry in $\sigma^*$is a type-2 forgery. The reduction aborts every time at least one condition is not satisfied (this happens with probability $\frac{1}{Q_{\mathsf{id}}}$ corresponding to the event $\mathscr{B}$ made the wrong guess for the target identity). Otherwise, $\mathscr{B}$ extracts its output to the co-CDH challenger from the type-2 forgery against $\mathsf{id}_{\tilde{j}}$ as follows. First, $\mathscr{B}$ removes from $\sigma^* = (\gamma^*, \mu_1^*, \ldots, \mu_t^*)$ the contributions by all other parties and

the parts of the forgery that depend $\varphi(h)$:

$$K_0 = \frac{\gamma^*}{\prod_{j\in[t]\setminus\tilde{j}} \left(\prod_{i=k_{j-1}+1}^{k_j} H(\ell_i)^{f_i}\right)^{\mathsf{sk}_j} \cdot \tilde{g}_1^{\mu_j^* \cdot \mathsf{sk}_j}} \cdot \frac{1}{\prod_{i=k+1}^{k_{\tilde{j}}} \varphi(h)^{t_i} \cdot \varphi(h)^{\tilde{t}\cdot\mu_{\tilde{j}}^*}}$$

Note that the left most denominator removes from γ^* all contributions by the identities $\mathsf{id}_i \neq \mathsf{id}_{\tilde{j}}$. The right most denominator is constructed with the randomness values present in L_H for the labels related to $\mathsf{id}_{\tilde{j}}$ and the messages m_i that were queried together with those labels and stored in L_σ (note that the m_i exist since this is a type-2 forgery). Let $y = \tilde{s} \cdot (\mu_{\tilde{j}} - f_{\tilde{j}}^*(m_{k+1}, \ldots, m_{k+k_{\tilde{j}}}))$ then $\mathscr{B}$ output to its co-CDH challenger is

$$K_1 = K_0^{\left(\frac{1}{y}\right)}.$$

It is straightforward to check that $K_1 = g_1^x$. The way we extract the co-CDH solution is indeed a generalization of Boneh *et al.*'s technique [7] to multi-key signatures. Thus, our reduction transforms type-2 forgeries into a solution to the co-CDH problem unless it aborts, which leads us to

$$\mathsf{Adv}_{\mathscr{B}}^{\mathsf{co-CDH}}(\lambda) \geq \frac{1}{Q_{\mathsf{id}}} \cdot \mathsf{Prob}[\mathscr{A} \text{ outputs a valid type-2 forgery }].$$

Combining the bounds we proved on type-3 and type-2 forgeries we obtain:

$$\mathsf{Adv}_{\mathsf{mklhs},\mathscr{A}}(\lambda) \leq \frac{1}{2} \cdot \left[R_H + Q_{\mathsf{id}} \cdot \mathsf{Adv}_{\mathscr{B}}^{\mathsf{co-CDH}}(\lambda)\right],$$

that proves the security of our mklhs scheme.

Correctness of the simulation. In what follows we argue that our reduction simulates the answers to signatures and hash queries in a perfect way.

First, we observe that the element $\tilde{g}_1$ constructed by $\mathscr{B}$ is a generator of $\mathbb{G}_1$ with overwhelming probability and that $\mathsf{pk}_{\tilde{j}} = h$ is distributed identically to the public key produced by the real KeyGen algorithm.

Second, the responses to all hash queries are uniformly random in $\mathbb{G}_1$, thus simulate the behavior of any cryptographic hash function. Indeed it is possible to rewrite $H(\ell)$ as

$$H(\ell) = g_1^{-s_{\mathsf{id}} \cdot m} \cdot \varphi(g_2)^t = g_1^{-s_{\mathsf{id}} \cdot m + t \cdot r}$$

where r is a fixed, unknown, value that depends on the homomorphism φ. Clearly, the $(t \cdot r)$ component in the exponent ensures that the hash values fall back to the uniform distribution.

Third, on all identities other than the target $\mathsf{id}_{\tilde{j}}$ our reduction behaves as in the real scheme.

Finally, we prove that the signatures output by $\mathscr{B}$ for the target identity $\mathsf{id}_{\tilde{j}}$ are identical to the signatures that would be output by the real Sign algorithm

given the public key $\mathsf{pk}_{\tilde{j}} = h$, the public parameters output by $\mathscr{B}$ and the answers to hash queries produced by our reduction. In detail, it suffices to show that, for every pair of label and message the simulated signature output by the $\mathscr{B}$ algorithm equals the real signature on ℓ and m, *i.e.,* the output of $\mathsf{Sign}(\mathsf{sk}_{\tilde{j}}, \ell, m)$ as in Figure 1. We show the equality only for the γ part of the signature, as the μ part is trivial. By construction, the output of Sign on the given inputs is:

$$\begin{aligned}(H(\ell) \cdot \tilde{g}_1^m)^{\mathsf{sk}_{\tilde{j}}} &= \left(g_1^{-\tilde{s}\cdot m}\varphi(g_2)^t \cdot \left[g_1^{\tilde{s}}\varphi(g_2)^{\tilde{t}}\right]^m\right)^{\mathsf{sk}_{\tilde{j}}} \\ &= \left(g_1^{-\tilde{s}\cdot m+\tilde{s}\cdot m} \cdot \varphi(g_2)^{t+\tilde{t}\cdot m}\right)^{\mathsf{sk}_{\tilde{j}}} \\ &= \varphi(h)^{t+\tilde{t}\cdot m}\end{aligned}$$

where the left-hand-side is the real signature and the final right-hand-side term is the simulated one.

4 Performance Evaluation

Our proposed scheme is well-tailored to asymmetric pairings, as most operations happen in $\mathbb{G}_1$. Key generation is the only important operation involving arithmetic in $\mathbb{G}_2$, but it is assumed to happen offline and only once per user. In detail, it requires just one fixed-base exponentiation in $\mathbb{G}_2$ involving the generator g_2 and the private key. Computing a signature involves hashing the label to $\mathbb{G}_1$, performing a fixed-based exponentiation of a generator $g_1 \in \mathbb{G}_1$ to compute g_1^m, a point addition to compute $H(\ell) \cdot g_1^m$, and a variable-base exponentiation using the private key to finally compute $(H(\ell) \cdot g_1^m)^{\mathsf{sk}}$. Hence, hashing to $\mathbb{G}_1$ and variable-base exponentiations dominate the execution time for generating signatures. The performance-critical operation in the homomorphic evaluation of mklhs is the multi-exponentiation in $\mathbb{G}_1$ that corresponds to the computation of $\gamma = \prod_{i=1}^{n} \gamma_i^{f_i}$. Finally, verifying a multi-key homomorphic signature for computations involving t signers involves mostly hashing to $\mathbb{G}_1$, fixed-base exponentiations in $\mathbb{G}_1$ and a product of t pairings.

4.1 Implementation

We implemented mklhs within the RELIC library [2] using two sets of supported parameters at the 128-bit security level. The first choice is the curve BLS12-381 with embedding degree $k = 12$ and 255-bit prime-order subgroup used in the ZCash cryptocurrency [25]. The second choice is a prime-order BN curve defined over a 382-bit field (BN-382). These choices are motivated by the recent attacks against the DLP over $\mathbb{G}_T$ [22] and are supported by the analysis in [24]. Although recent analysis has point out that even larger parameters may be needed for 128-bit security [4], our main performance observations should still hold. The two curves are defined over a prime field $\mathbb{F}_p$ such that $p \equiv 3 \pmod 4$, providing an efficient towering for representing $\mathbb{F}_{p^k}$ and efficient extraction of square roots. RELIC provides Assembly acceleration for Intel 64-bit platforms for both curves using a shared codebase, which means that finite field arithmetic is

implemented using essentially the same techniques, allowing for fair comparisons across different curves and protocols. The resulting code is publicly available in the library repository.

The top two blocks of Table 1 list the curves parametrization, while the bottom block displays the concrete instantiation of z_0 used in our evaluation. Both curves have the same format $E : y^2 = x^3 + b$ when represented in short Weierstraß equation, therefore the same arithmetic optimizations apply to both curves. A significant difference comes with the length of parameters, such as the group size q and cofactor $h = \#E/q$, which offer interesting performance trade-offs: hashing and membership testing in $\mathbb{G}_1$ are faster on the BN curve due to the prime order (no need to clear cofactors); while exponentiations are more efficient on the BLS12 curve due to shorter exponents. We could consider using the short exponent optimization for the BN curve, but these would violate security assumptions about the distribution of exponents.

Table 1. Parametrization and concrete parameters for the BN and BLS12 pairing-friendly curves used in our implementation.

BN curves: $k = 12$, $\rho \approx 1$	
$p(z)$	$36z^4 + 36z^3 + 24z^2 + 6z + 1$
$q(z)$	$36z^4 + 36z^3 + 18z^2 + 6z + 1$
$t(z)$	$6z^2 + 1$
BLS12 curves: $k = 12$	
$p(z)$	$(z-1)^2(z^4 - z^2 + 1)/3 + z$
$q(z)$	$z^4 - z^2 + 1$
$t(z)$	$z + 1$

E	b	z_0	$\lceil \log_2 p \rceil$	$\lceil \log_2 q \rceil$	$\lceil \log_2 h \rceil$
BN-382	2	$-(2^{94} + 2^{78} + 2^{67} + 2^{64} + 2^{48} + 1)$	382	382	1
BLS12-381	4	$-(2^{63} + 2^{62} + 2^{60} + 2^{57} + 2^{48} + 2^{16})$	381	255	126

We also took side-channel resistance into consideration. The most critical procedure in our protocol from the point of view of implementation security is the signature generation. This is due to the fact that all other recurrent procedures are publicly evaluated and do not involve secret or sensitive information. In the signature generation, the fixed-base exponentiation does not need counter measures because messages are assumed to be public. Therefore, the variable-base exponentiation involving sk is the only operation requiring protection from side-channel leakage. Our pairing-friendly curves support efficient endomorphisms that we exploit to speed up the exponentiation [16]. Moreover, we combine the GLV scalar decomposition method for the private key [16] with the constant-time exponentiation based on regular recoding of exponents [21] to implement a side-channel resistant signing procedure. The GLV decomposition process was implemented in variable time and is assumed to be executed during key generation,

after which only the subscalars are needed. The constant-time implementation of exponentiation has the typical constant-time countermeasures, with lookup in precomputed tables performed by linear scanning across the entire table and branchless programming to replace all branches with logical operations.

Hashing to pairing groups is commonly implemented using a heuristic try-and-increment approach, where the string is hashed to the x-coordinate of a point and incremented until a suitable y-coordinate is found after extracting a square root using the curve equation. The point (x, y) is then multiplied by the curve cofactor to guarantee that the resulting point is in the right q-order subgroup. A problem with this approach is that it lacks any proper guarantees about the output distribution. In particular, it does not satisfy the requirements of a random oracle assumed in our scheme. We instead employ a Shallue–van de Woestijne (SW) [27] encoding of strings that is indifferentiable from a random oracle. Concretely, we employ versions of the approach customized to pairing-friendly curves belonging to the BN [15] and BLS12 families [28]. For BLS12 curves, we implemented the simpler version recently proposed in [28] with the optimization of replacing the cofactor multiplication inside hashing by the value $(1-z_0)$ with low Hamming weight. Homomorphic evaluation of linear functions over signatures was implemented with standard binary multi-exponentiation techniques, because the coefficients are assumed to be small. We implemented products of pairings using the conventional interleaving techniques to eliminate point doublings in the pairing computation and share the final exponentiation [19].

For comparison to the related work by Schabhüser *et al.* [26], we also implemented their scheme under similar constraints. This protocol requires an underlying conventional signature algorithm, for which we chose the pairing-based Boneh-Lynn-Schacham (BLS) [9] short signature scheme to rely on the same set of parameters. Computation of BLS signatures was implemented in constant-time in the same way as mklhs, because it depends on the secrecy of a long-term key. All other operations were implemented in variable-time for better performance, since they rely only on public or ephemeral data.

4.2 Experimental results and discussion

We benchmarked our implementation on a high-end Intel Core i7-6700K Skylake processor running at 4.0GHz. We turned off HyperThreading and TurboBoost to reduce noise in the benchmarks. The benchmarking dataset was created by generating 10 different users, who sign 16 individual messages each using 16 different labels. These signatures are collected by a third party, which homomorphically evaluates a linear function f composed of 16 random 32-bit integer coefficients across all signatures. The resulting multi-key signature is then verified. Each procedure is executed 10^4 times and the timings for mklhs and Schabhüser *et al.* instantiated both with curves BN-382 and BLS12-381 are presented in Table 2. The figures are amortized per user, so similar performance improvements are expected for datasets of different size.

Our scheme is clearly more efficient than Schabhüser *et al.* and trades off advanced security properties (such as context hiding) in favor of performance

Table 2. Efficiency comparison of our mklhs with Schabhüser *et al.*'s scheme. We display execution times in clock cycles (cc) for each of the main algorithms in the protocols. The dataset includes homomorphic evaluation and verification of 16 signatures computed by 10 users each. Figures represent the average of 10^4 executions, and fastest timings are typeset in **bold**. Timings for signature computation refer to individual signatures, and timings for homomorphic evaluation and verification are amortized per user. Individual and combined signature sizes are also displayed for reference, assuming point compression of $\mathbb{G}_1$ and $\mathbb{G}_2$ elements.

	Protocol			
	Schabhüser *et al.* [26]		mklhs	
Operation	BN-382	BLS12-381	BN-382	BLS12-381
Key Generation (cc)	42,071,481	28,575,291	879,258	**592,454**
Signature computation (cc)	5,643,798	4,137,016	1,552,369	**1,257,424**
Homomorphic Evalulation (cc)	3,346,135	3,349,471	666,542	**661,664**
Verification (cc)	14,702,224	12,770,063	10,814,364	**10,710,053**
Individual signature (bytes)	384	384	96	80
Combined sigs for t signers (bytes)	$144 + 240t$	$144 + 240t$	$48(1 + t)$	$48 + 32t$

and compactness. Computing an individual signature in mklhs is more than 3 times faster than in [26]. The homomorphic evaluation of mklhs is also much more efficient (around 5 times faster), since it does not require exponentiations in $\mathbb{G}_2$ to preserve the homomorphism. Verification is up to 26% faster in our proposal, depending on curve choice. Comparing the two curves in mklhs, we have the following observations. Key generation and signing are faster on BLS12-381 due to shorter exponents. BN-382 becomes competitive for homomorphic evaluation with same-length coefficients, and verification due to faster hashing but slower exponentiations. Protocols relying heavier on hashing are expected to further benefit from choosing BN-382 as the underlying pairing-friendly curve.

5 Conclusion

We presented mklhs, a novel multi-key linearly homomorphic signature scheme that addresses the problem of outsourcing computation on data authenticated by independent users. Our construction relies solely on standard assumptions (co-CDH and ROM), is conceptually simpler than existing proposals, and enjoys the most succinct signatures up to date with just 1 element in $\mathbb{G}_1$ and t elements in $\mathbb{Z}_q$ (t is the number of distinct signers). In addition, mklhs is the first direct construction of a multi-key linearly homomorphic signature scheme not based on a compiler. We use standard tools of pairing-based cryptography and design extremely efficient algorithms for signing, combining and verifying.

Compared to existing schemes, our mklhs scheme substantially improves execution times for most relevant operations in a multi-key homomorphic signature scheme, outperforming existing proposals by several times. Valuable pointers to further improvement include designing a context hiding version of mklhs and extending homomorphic computations to a wider and more expressive set of

functions. From the implementation point of view, instantiating the scheme with other parameters would also allow a better view of performance trade-offs across different curves.

Acknowledgments. This work was partly funded by: the European Research Council (ERC) under the European Unions's Horizon 2020 research and innovation programme under grant agreement No 669255 (MPCPRO); and the Concordium Blockchain Research Center at Aarhus University, Denmark. The first author is also affiliated to the DIGIT Centre for Digitalisation, Big Data and Data Analytics at Aarhus University.

References

1. S. Agrawal, D. Boneh, X. Boyen, and D. M. Freeman. Preventing pollution attacks in multi-source network coding. In *International Workshop on Public Key Cryptography*, pages 161–176. Springer, 2010.
2. D. F. Aranha and C. P. L. Gouvêa. RELIC is an Efficient LIbrary for Cryptography. `https://github.com/relic-toolkit/relic`.
3. N. Attrapadung, B. Libert, and T. Peters. Efficient completely context-hiding quotable and linearly homomorphic signatures. In *International Workshop on Public Key Cryptography*, pages 386–404. Springer, 2013.
4. R. Barbulescu and S. Duquesne. Updating key size estimations for pairings. *Journal of Cryptology*, Jan 2018.
5. P. S. Barreto, B. Lynn, and M. Scott. Constructing elliptic curves with prescribed embedding degrees. In *International Conference on Security in Communication Networks*, pages 257–267. Springer, 2002.
6. P. S. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. In *International Workshop on Selected Areas in Cryptography*, pages 319–331. Springer, 2005.
7. D. Boneh, D. Freeman, J. Katz, and B. Waters. Signing a linear subspace: Signature schemes for network coding. In *International Workshop on Public Key Cryptography*, pages 68–87. Springer, 2009.
8. D. Boneh and D. M. Freeman. Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In *International Workshop on Public Key Cryptography*, pages 1–16. Springer, 2011.
9. D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 514–532. Springer, 2001.
10. D. Catalano, D. Fiore, and B. Warinschi. Efficient network coding signatures in the standard model. In *International Workshop on Public Key Cryptography*, pages 680–696. Springer, 2012.
11. D. Catalano, D. Fiore, and B. Warinschi. Homomorphic signatures with efficient verification for polynomial functions. In *International Cryptology Conference*, pages 371–389. Springer, 2014.
12. S. Chatterjee and A. Menezes. On cryptographic protocols employing asymmetric pairings—the role of ψ revisited. *Discrete Applied Mathematics*, 159(13):1311–1322, 2011.

13. D. Fiore, A. Mitrokotsa, L. Nizzardo, and E. Pagnin. Multi-key homomorphic authenticators. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 499–530. Springer, 2016.
14. D. Fiore and E. Pagnin. Matrioska: a compiler for multi-key homomorphic signatures. In *International Conference on Security and Cryptography for Networks*, pages 43–62. Springer, 2018.
15. P.-A. Fouque and M. Tibouchi. Indifferentiable hashing to Barreto–Naehrig curves. In *International Conference on Cryptology and Information Security in Latin America*, pages 1–17. Springer, 2012.
16. R. P. Gallant, R. J. Lambert, and S. A. Vanstone. Faster point multiplication on elliptic curves with efficient endomorphisms. In *Annual International Cryptology Conference*, pages 190–200. Springer, 2001.
17. R. Gennaro and D. Wichs. Fully homomorphic message authenticators. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 301–320. Springer, 2013.
18. S. Gorbunov, V. Vaikuntanathan, and D. Wichs. Leveled fully homomorphic signatures from standard lattices. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 469–477. ACM, 2015.
19. R. Granger and N. P. Smart. On computing products of pairings. *IACR Cryptology ePrint Archive*, 2006:172, 2006.
20. R. Johnson, D. Molnar, D. Song, and D. Wagner. Homomorphic signature schemes. In *Cryptographers' Track at the RSA Conference*, pages 244–262. Springer, 2002.
21. M. Joye and M. Tunstall. *Fault analysis in cryptography*, volume 147. Springer, 2012.
22. T. Kim and R. Barbulescu. Extended tower number field sieve: A new complexity for the medium prime case. In *Annual International Cryptology Conference*, pages 543–571. Springer, 2016.
23. R. W. Lai, R. K. Tai, H. W. Wong, and S. S. Chow. Multi-key homomorphic signatures unforgeable under insider corruption. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 465–492. Springer, 2018.
24. A. Menezes, P. Sarkar, and S. Singh. Challenges with assessing the impact of NFS advances on the security of pairing-based cryptography. In *International Conference on Cryptology in Malaysia*, pages 83–108. Springer, 2016.
25. E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from Bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE, 2014.
26. L. Schabhüser, D. Butin, and J. Buchmann. Context hiding multi-key linearly homomorphic authenticators. In *Cryptographers' Track at the RSA Conference*, pages 493–513. Springer, 2019.
27. A. Shallue and C. E. van de Woestijne. Construction of rational points on elliptic curves over finite fields. In *International Algorithmic Number Theory Symposium*, pages 510–524. Springer, 2006.
28. R. S. Wahby and D. Boneh. Fast and simple constant-time hashing to the BLS12-381 elliptic curve. Cryptology ePrint Archive, Report 2019/403, 2019. `https://eprint.iacr.org/2019/403`.