Matrioska: A Compiler for Multi-key Homomorphic Signatures

Dario Fiore[1] and Elena Pagnin[2(✉)]

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

Abstract. Multi-Key Homomorphic Signatures (MK-HS) enable clients in a system to sign and upload messages to an untrusted server. At any later point in time, the server can perform a computation C on data provided by t different clients, and return the output y and a short signature $\sigma_{C,\mathsf{y}}$ vouching for the correctness of y as the output of the function C on the signed data. Interestingly, MK-HS enable verifiers to check the validity of the signature using solely the public keys of the signers whose messages were used in the computation. Moreover, the signatures $\sigma_{C,\mathsf{y}}$ are succinct, namely their size depends at most linearly in the number of clients, and only logarithmically in the total number of inputs of C. Existing MK-HS are constructed based either on standard assumptions over lattices (Fiore *et al.* ASIACRYPT'16), or on non-falsifiable assumptions (SNARKs) (Lai *et al.*, ePrint'16). In this paper, we investigate connections between single-key and multi-key homomorphic signatures. We propose a generic compiler, called Matrioska, which turns any (sufficiently expressive) single-key homomorphic signature scheme into a multi-key scheme. Matrioska establishes a formal connection between these two primitives and is the first alternative to the only known construction under standard falsifiable assumptions. Our result relies on a novel technique that exploits the homomorphic property of a single-key HS scheme to compress an arbitrary number of signatures from t different users into only t signatures.

1 Introduction

Consider a scenario where a user Alice uploads a collection of data items $x_1, \ldots, x_{\mathsf{n}}$ to an untrusted server. Later on, the server executes a computation $\mathcal{P}$ on this data and sends the result $\mathsf{y} = \mathcal{P}(x_1, \ldots, x_{\mathsf{n}})$ to another user Bob. *How can Bob be sure that* y *is the correct result obtained by running* $\mathcal{P}$ *on Alice's data?*

A trivial solution to this problem could be obtained by employing digital signatures: Alice could sign each data item x_i and send to the server the signatures $\sigma_1, \ldots, \sigma_{\mathsf{n}}$. Next, to convince Bob, a server can send along with y the original inputs with their signatures, and Bob should check that $\mathsf{y} = \mathcal{P}(x_1, \ldots, x_{\mathsf{n}})$ and that each σ_i is a valid signature for x_i. While this solution solves the above

D. Catalano and R. De Prisco (Eds.): SCN 2018, LNCS 11035, pp. 43–62, 2018.
https://doi.org/10.1007/978-3-319-98113-0_3

security concern, it has a clear efficiency drawback: it requires communication between the server and the verifier Bob that is *linear* in the input size of $\mathcal{P}$. This cost is undesirable and can even be unacceptable if Bob is cannot store the $x_1, \ldots, x_{\mathsf{n}}$.

Homomorphic Signatures. A solution to the above problem that achieves both security and efficiency can be obtained by using *homomorphic signatures* (HS). With this primitive, Alice can use her secret key to sign $x_1, \ldots, x_{\mathsf{n}}$ and sends the signed data items to the server. The server can use a special procedure Eval that, on input a program $\mathcal{P}$ and a collection of signatures $\sigma_1, \ldots, \sigma_{\mathsf{n}}$, outputs a signature $\sigma_{\mathcal{P},\mathsf{y}}$. Given Alice's public key and a triple $(\mathcal{P}, \mathsf{y}, \sigma_{\mathcal{P},\mathsf{y}})$, Bob (or anyone else) can get convinced that y is the correct output of $\mathcal{P}$ on inputs $(x_1, \ldots, x_{\mathsf{n}})$ signed by Alice. Very informally, homomorphic signatures are secure in the sense that an untrusted server (without knowing Alice's secret key) must not be able to convince the verifier of a false result. An additional property that makes this cryptographic primitive interesting and non-trivial is that signatures must be *succinct*. This means that the size of $\sigma_{\mathcal{P},\mathsf{y}}$ must be significantly smaller than $\mathcal{P}$'s input size, *e.g.*, $size(\sigma_{\mathcal{P},\mathsf{y}}) = O(\log \mathsf{n})$.

The notion of homomorphic signatures was proposed by Desdmedt [16] and first formalized by Johnson *et al.* [24]. Boneh *et al.* [4] proposed the first scheme for computing linear functions over signed vectors and showed an application to preventing pollution attacks in linear network coding. Following [4], a long series of works (e.g., [1,2,6,8,9,11–13,15,19,20,26]) addressed the problem of constructing linearly-homomorphic signatures obtaining new schemes that improved on multiple fronts, such as efficiency, security, and privacy. A few more works addressed the problem of constructing schemes for more expressive functionalities [5,7,14,23]. Boneh and Freeman [5] proposed the first scheme for polynomial functions based on lattices, which was later improved by Catalano, Fiore and Warinschi [14] based on multilinear maps. In 2015, Gorbunov, Vaikuntanathan and Wichs [23] constructed the first HS scheme for arbitrary circuits of bounded depth from standard lattices.

Multi-key Homomorphic Signatures. In a recent work, Fiore *et al.* [17] initiated the study of *multi-key homomorphic signatures* (MK-HS). In a nutshell, MK-HS are homomorphic signatures that allow for computing on data signed using different secret keys. This capability extends that one of previously known homomorphic signatures, and is useful in all those applications where one wants to compute on data provided (and signed) by multiple users. In addition to formally defining the notion of multi-key homomorphic signatures, Fiore *et al.* proposed a construction of MK-HS based on lattices that supports bounded depth circuits. Their scheme is obtained by extending the techniques of the single-key scheme of Gorbunov *et al.* [23]. Another recent work by Lai *et al.* [25] shows how to build an MK-HS using SNARKs and digital signatures. However, since SNARKs are likely to be based on non-falsifiable assumptions [22], the resulting MK-HS also relies on non standard assumptions.

1.1 Our Contribution

In this work, we continue the study of multi-key homomorphic signatures. Our main interest is to identify connections between multi-key homomorphic signatures and their single-key counterpart. In particular, we provide the first generic method to construct multi-key homomorphic signatures from (sufficiently expressive) single-key HS schemes. Our main contribution is a compiler, called Matrioska, that yields the following result:

Theorem 1 (Informal). *Let* HS *be a homomorphic signature scheme for circuits of polynomial size. Then, for a constant* t *representing the number of distinct keys involved in a computation, there exists a multi-key homomorphic signature scheme* $\mathsf{MKHS}(\mathsf{HS}, t)$ *for circuits of polynomial size. Furthermore, if* HS *has signatures bounded by a fixed polynomial* $p(\lambda)$*,* $\mathsf{MKHS}(\mathsf{HS}, t)$ *has signatures bounded by* $t \cdot p(\lambda)$*.*

Our result essentially shows that for a sufficiently expressive class of functions multi-key and single-key homomorphic signatures are equivalent. Our construction is the first to establish a formal connection between these two primitives without resorting to powerful primitives such as SNARKs which only yield constructions from non-falsifiable assumptions. Also, we propose a new methodology to construct MK-HS, which is the first alternative to the only known construction from standard assumptions [17]. In particular, while the techniques in [17] are specific to an algebraic lattice setting, our construction works in a generic fashion and as such it will allow to immediately obtain new MK-HS schemes from any future proposal of single-key HS.

Our MK-HS construction is quite involved and its efficiency is, admittedly, theoretical. In particular, in order to support circuits of (polynomial) size s, we need to start from a single-key HS scheme that supports circuits of size $s^{c_s{}^{t-1}}$, where t is the number of distinct keys involved in the computation and c_s is some constant that depends on the single-key HS scheme. Therefore our generic construction generates multi-key homomorphic signature schemes that can support computations among a constant number of keys (*i.e.,* users) only.

Nevertheless, our MK-HS scheme has succinct signatures that have size $t \cdot p(\lambda)$, which is non-trivial as it is independent of the total number of inputs involved in the computation. Indeed, even in the multi-key setting a trivial solution to build MK-HS from digital signatures (and even from HS) would require communication linear in the total number of inputs of a computation, i.e., $O(\mathsf{n} \cdot t)$, assuming each user provides n inputs.

An Overview of Our Techniques. The main challenge in constructing an MK-HS scheme generically from a single-key one is to obtain a construction with succinct signatures. In particular, obtaining succinctness requires some mechanism to "compress" $\mathsf{n} \cdot t$ signatures into some information that can at most depend linearly on $\log \mathsf{n}$ and t. While single-key HS allow for compressing signatures pertaining to the same key, this property seems of no utility when one needs to compute on signatures pertaining to different keys, if nothing about

their structure can be assumed.[1] To overcome this challenge, we devise a novel technique that allows us to compress $\mathsf{n} \cdot t$ signatures from t different users into t signatures; for this we show how to use the homomorphic property of the single-key HS scheme in order to inductively "prove" that the signatures of the first i users verify correctly on the corresponding inputs.

In what follows we illustrate the core idea of our technique considering, for simplicity, the two-client case $t = 2$, and assuming each users contributes to the computation with n inputs.

Let $C : \{0,1\}^{2\cdot\mathsf{n}} \to \{0,1\}$ be the circuit we wish to evaluate. Given the messages $\mathsf{m}_1, \ldots \mathsf{m}_\mathsf{n}$ by user id_1 and $\mathsf{m}_{\mathsf{n}+1}, \ldots \mathsf{m}_{2\cdot\mathsf{n}}$ by user id_2, we wish to authenticate the output of $\mathsf{y} = C(\mathsf{m}_1, \ldots, \mathsf{m}_{2\cdot\mathsf{n}})$. Let σ_i be the signature for the message m_i; in particular the first n signatures and the last n signatures are associated to different secret keys.

The initial step is to construct a $(2 \cdot \mathsf{n})$-input circuit E_0 such that $E_0(x_1, \ldots, x_{2\mathsf{n}}) = 1$ iff $C(x_1, \ldots, x_{2\mathsf{n}}) = \mathsf{y}$. Second, define a new circuit $E_1 : \{0,1\}^\mathsf{n} \to \{0,1\}$ that is E_0 with the last n inputs hardwired: $E_1(x_1, \ldots, x_\mathsf{n}) = E_0(x_1, \ldots, x_\mathsf{n}, \mathsf{m}_{\mathsf{n}+1}, \ldots, \mathsf{m}_{2\mathsf{n}})$. Now E_1 is a circuit that has inputs by a single client only, thus we can run $\hat{\sigma}_1 \leftarrow \mathsf{HS.Eval}(E_1, \mathsf{pk}_1, \sigma_1, \ldots, \sigma_\mathsf{n})$. By the correctness of the single-key homomorphic signature scheme it must hold $\mathsf{HS.Verify}(E_1, \mathsf{pk}_1, \hat{\sigma}_1, 1) = 1$. At this point, we already compressed the signatures $\sigma_1, \ldots, \sigma_\mathsf{n}$ into a single signature $\hat{\sigma}_1$. This is however not yet sufficient for succinctness because verifying $\hat{\sigma}_1$ requires the circuit E_1, which in turn requires to transmit to the verifier n messages $(\mathsf{m}_{\mathsf{n}+1}, \ldots, \mathsf{m}_{2\mathsf{n}})$ to let him reconstruct E_1.

This is where the inductive reasoning, and our new technique, begins. Very intuitively, we use the signatures of the second user to "prove" that $\mathsf{HS.Verify}(E_1, \mathsf{pk}_1, \hat{\sigma}_1, 1) = 1$, without letting the verifier run this verification explicitly. Let us see $\mathsf{H} = \mathsf{HS.Verify}((E_1, (\tau_1, \ldots, \tau_\mathsf{n})), \mathsf{pk}_1, \hat{\sigma}_1, 1)$ as a binary string with the description of a (no input) circuit. Look for the bits of H where the values $\mathsf{m}_{\mathsf{n}+1}, \ldots, \mathsf{m}_{2\mathsf{n}}$ are embedded. We can define a new circuit description E_2 that is the same as H except that the hardwired values $\mathsf{m}_{\mathsf{n}+1}, \ldots, \mathsf{m}_{2\mathsf{n}}$ are replaced with input gates. Thus E_2 is an n-input circuit satisfying $E_2(\mathsf{m}_{\mathsf{n}+1}, \ldots, \mathsf{m}_{2\mathsf{n}}) = \mathsf{HS.Verify}(E_1, \mathsf{pk}_1, \hat{\sigma}_1, 1)$, which returns 1 by correctness of HS.

Now, the crucial observation is that E_2 is a circuit on inputs by the second client only. Thus, we can run $\hat{\sigma}_2 \leftarrow \mathsf{HS.Eval}(E_2, \mathsf{pk}_2, \sigma_{\mathsf{n}+1}, \ldots, \sigma_{2\mathsf{n}})$. By the correctness of the HS scheme, $\mathsf{HS.Verify}(E_2, \mathsf{pk}_2, \hat{\sigma}_2, 1) = 1$. Note that E_2 does not contain any of the messages $\mathsf{m}_1, \ldots, \mathsf{m}_{2\cdot\mathsf{n}}$ hardwired; in particular E_2 is completely determined by C, y, pk_1, $\hat{\sigma}_1$ and a description of $\mathsf{HS.Verify}$. Hence, given $(\hat{\sigma}_1, \hat{\sigma}_2)$ the verifier can reconstruct E_2 and check if $\mathsf{HS.Verify}(E_2, \mathsf{pk}_2, \hat{\sigma}_2, 1) = 1$. Intuitively, this proves that for some messages signed by the second user $E_2(\mathsf{m}_{\mathsf{n}+1}, \ldots, \mathsf{m}_{2\mathsf{n}}) = 1$. By the correctness of HS, this in turn implies $E_1(\mathsf{m}_1, \ldots, \mathsf{m}_\mathsf{n}) = 1$ for some messages signed by the first user; and by construction of E_1 the latter implies $C(\mathsf{m}_1, \ldots, \mathsf{m}_{2\mathsf{n}}) = \mathsf{y}$.

[1] This is the case if one aims for a generic single-key to multi-key construction. In contrast, knowing for example the algebraic structure of signatures can be of help, as exploited in [17].

Our compiler, extends the above idea to multiple users, showing that at each step i the problem consists in proving correctness of a computation E_{i-1} that depends only on the inputs of user i, while inputs of users $> i$ are hardwired into it. This means that a progressive application of this idea lets the hardwired inputs progressively disappear up to the point of obtaining a circuit E_t which has no input hardwired and thus can be reconstructed by the verifier. This is the only computation explicitly checked by the verifier. By construction, E_t encodes the nested execution of several single-key HS verifications (from which our compiler's name "Matrioska"), and validity of E_t implicitly implies that each E_i returns 1 (even if the verifier does not know E_i itself). In this description we favor intuition to precision. A detailed presentation can be found in Sect. 3.

2 Preliminaries

Notation. The security parameter of our schemes is denoted by λ. For any $n \in \mathbb{N}$, we use $[n]$ to denote the set $[n] := \{1, \ldots, n\}$. The symbol lg denotes the logarithm in base 2; $||$ denotes the string concatenation, *e.g.,* $(00)||(10) = (0010)$; bold font letters, *e.g.,* $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_{\mathsf{n}})$, denote vectors. A function $\epsilon(\lambda)$ is said negligible in λ (denoted as $\epsilon(\lambda) = \mathsf{negl}(\lambda)$) if $\epsilon(\lambda) = O(\lambda^{-c})$ for every constant $c > 0$. Also, we often write $\mathsf{poly}(\cdot)$ to denote a function that can be expressed as a polynomial.

2.1 Circuits

We use a modeling of circuits similar to the one in [3]. We define circuits as 6-tuples $C = (\mathsf{n}, \mathsf{u}, \mathsf{q}, \mathsf{L}, \mathsf{R}, \mathsf{G})$. The value $\mathsf{n} \geq 1$ denotes the number of inputs to the circuit, $\mathsf{u} \geq 1$ is the number of outputs and $\mathsf{q} \geq 1$ is the number of gates. Let w denote the total number of wires in the circuit. For the circuits considered in this work $\mathsf{w} = \mathsf{n} + \mathsf{q}$. The functions L and R define respectively the left and right input wire to any given gate $g \in [\mathsf{q}]$, formally, $\mathsf{L}, \mathsf{R} : [\mathsf{q}] \rightarrow [\mathsf{w}] \cup \{0\}$. Finally, $\mathsf{G} : [\mathsf{q}] \rightarrow \{0,1\}$ encodes the gates by mapping each gate $g \in [\mathsf{q}]$ into a single bit G_g. In our construction we treat circuit descriptions C as binary strings. Similarly to [3], the size of our circuit description is quasi-linear in the number of wires: $|C| \in O(\mathsf{w} \lg(\mathsf{w}))$. Differently from [3], we number gates from 1 to q (instead of from $\mathsf{n}+1$ to $\mathsf{n}+\mathsf{q}$) and label the outgoing wire of a gate g as $g + \mathsf{n}$. Moreover, we introduce the 0 wire to denote *constant output* gates, *e.g.,* no-input gates or gates that have the same output independently of the input values, and allow for a gate to have the same left and right input, *i.e.,* $\mathsf{L}(g) \leq \mathsf{R}(g) < g + \mathsf{n}$. The largest component in the string C is the descriptions of the function L (and R), that is a sequence of q values in $[\mathsf{w}] \cup \{0\}$, therefore $|\mathsf{L}| = |\mathsf{R}| = \mathsf{q} \lg(\mathsf{w}+1)$. Hence, for a fixed and reasonable encoding it holds $|C| \in O(\mathsf{w} \lg(\mathsf{w}))$.

As an example of a circuit consider the following EQ^{y} circuit (that will be used in our generic compiler) $EQ^{\mathsf{y}} = \big(1, 1, 5, (01134), (02325), (\mathsf{y}, 1, 1, 1, 1)\big)$.

We explain the procedure to evaluate a 1-output, n-input circuit and refer the reader to [18] for the general case. Given $(x_1, \ldots, x_{\mathsf{n}})$ and the circuit description $C = (\mathsf{n}, 1, \mathsf{q}, \mathsf{L}, \mathsf{R}, \mathsf{G})$, compute $\mathsf{y} = C(x_1, \ldots, x_{\mathsf{n}})$ as follows. Retrieve the label of

the left and right input wires to gate $g = i$, for $i = 1, 2, \ldots, \mathsf{q}$. Let $l \leftarrow \mathsf{L}(i)$ and $r \leftarrow \mathsf{L}(i)$. Create a new variable $x_{\mathsf{n}+i} \in \{0,1\}$. If $l = 0 = r$, g is a constant gate, assign $x_{\mathsf{n}+i} \leftarrow \mathsf{G}(i)$. Otherwise, by definition $l \neq 0 \neq r$, retrieve the values x_l and x_r, and return $x_{\mathsf{n}+i} \leftarrow x_l$ if $\mathsf{G}(i) = 0$, or $x_{\mathsf{n}+i} \leftarrow \mathsf{NAND}(x_l, x_r)$ if $\mathsf{G}(i) = 1$. The output is $x_{\mathsf{n}+\mathsf{q}} = \mathsf{y} = C(x_1, \ldots, x_{\mathsf{n}})$.

Another interesting operation on circuits is circuit composition. Given two circuits, C_1 and C_2, we say that C_1 is *composable* with C_2 if $\mathsf{u}_1 = \mathsf{n}_2$. Intuitively, composition connects each output wire of C_1 with one input wire of C_2. We denote the circuit composition as $C_3 = C_1 \triangleright C_2$. The resulting circuit $C_3 = (\mathsf{n}_3, \mathsf{u}_3, \mathsf{q}_3, \mathsf{L}_3, \mathsf{R}_3, \mathsf{G}_3)$ is defined as: $\mathsf{n}_3 = \mathsf{n}_1$, $\mathsf{u}_3 = \mathsf{u}_2$, $\mathsf{q}_3 = \mathsf{q}_1 + \mathsf{q}_2$. Let w_i be the number of wires in C_i, then

$$\mathsf{L}_3 = \begin{cases} \mathsf{L}_1(i) & \text{for } i \in [\mathsf{w}_1] \\ 0 & \text{for } i \in [\mathsf{w}_1 + \mathsf{w}_2] \setminus [\mathsf{w}_1] \text{ and } \mathsf{L}_2(i - \mathsf{w}_1) = 0 \\ \mathsf{L}_2(i - \mathsf{w}_1) + \mathsf{w}_1 - \mathsf{u}_1 & \text{for } i \in [\mathsf{w}_1 + \mathsf{w}_2] \setminus [\mathsf{w}_1] \text{ and } \mathsf{L}_2(i - \mathsf{w}_1) \neq 0 \end{cases}$$

Note that the entries of L_3 that are set to 0 preserve constant output gates. The right-input function R_3 is defined analogously. The right-input function R_3 is defined analogously. Finally, $\mathsf{G}_3 = \mathsf{G}_1 || \mathsf{G}_2$.

2.2 Multi-key Homomorphic Signatures

We start by recalling the notion of labeled programs of Gennaro and Wichs [21].

Labeled Programs [21]. A labeled program $\mathcal{P}$ is a tuple $(C, \ell_1, \ldots, \ell_t)$, such that $C : \mathcal{M}^t \to \mathcal{M}$ is a function of t variables (*e.g.,* a circuit) and $\ell_i \in \{0,1\}^*$ is a label for the i-th input of C. Labeled programs can be composed as follows: given $\mathcal{P}_1, \ldots, \mathcal{P}_n$ and a function $G : \mathcal{M}^n \to \mathcal{M}$, the composed program $\mathcal{P}^*$ is the one obtained by evaluating G on the outputs of $\mathcal{P}_1, \ldots, \mathcal{P}_n$, and it is denoted as $\mathcal{P}^* = G(\mathcal{P}_1, \ldots, \mathcal{P}_n)$. The labeled inputs of $\mathcal{P}^*$ are all the distinct labeled inputs of $\mathcal{P}_1, \ldots, \mathcal{P}_n$ (all the inputs with the same label are grouped together and considered as a unique input of $\mathcal{P}^*$).

We recall the definitions of Fiore *et al.* [17] for multi-key homomorphic authenticators, adapted to the case of signature schemes only. Following [17], we consider labels where $\ell = (\mathsf{id}, \tau)$, such that id is a given client identity and τ is a tag which refers to the client's input data. To ease the reading, we use the compact and improper notation $\mathsf{id} \in \mathcal{P}$ meaning that there exists at least one index label ℓ in the description of $\mathcal{P} = (C, (\ell_1, \ldots, \ell_{\mathsf{n}}))$ such that $\ell = (\mathsf{id}, \tau)$ for some string τ.

Definition 1 (Multi-key Homomorphic Signature [17]). *A multi-key homomorphic signature scheme* MKHS *is a tuple of five PPT algorithms* $\mathsf{MKHS} = (\mathsf{MKHS.Setup}, \mathsf{MKHS.KeyGen}, \mathsf{MKHS.Sign}, \mathsf{MKHS.Eval}, \mathsf{MKHS.Verify})$ *that satisfies the properties of* authentication correctness, evaluation correctness, succinctness *and* security. *The algorithms are defined as follows:*

$\mathsf{MKHS.Setup}(1^\lambda)$. *The setup algorithm takes as input the security parameter* λ *and outputs some public parameters* pp *including a description of an identity space* ID*, a tag space* $\mathcal{T}$ *(these implicitly define the label space* $\mathcal{L} = \mathsf{ID} \times \mathcal{T}$*), a message space* $\mathcal{M}$ *and a set of admissible functions* $\mathcal{F}$*. The* pp *are input to all the following algorithms, even when not specified.*

$\mathsf{MKHS.KeyGen}(\mathsf{pp})$. *The key generation algorithm takes as input the public parameters and outputs a pair of keys* $(\mathsf{sk}, \mathsf{pk})$*, where* sk *is a secret signing key, while* pk *is the public evaluation and verification key.*

$\mathsf{MKHS.Sign}(\mathsf{sk}, \Delta, \ell, \mathsf{m})$. *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*.*

$\mathsf{MKHS.Eval}(\mathcal{P}, \Delta, \{(\sigma_i, \mathsf{pk}_{\mathsf{id}_i})\}_{i\in[\mathsf{n}]})$. *The evaluation algorithm takes as input a labeled program* $\mathcal{P} = (C, (\ell_1, \ldots, \ell_\mathsf{n}))$*, where* C *is an* n*-input circuit* $C : \mathcal{M}^\mathsf{n} \longrightarrow \mathcal{M}$*, a dataset identifier* Δ *and a set of signature and public-key pairs* $\{(\sigma_i, \mathsf{pk}_{\mathsf{id}_i})\}_{i\in[\mathsf{n}]}$*. The output is an homomorphic signature* σ*.*

$\mathsf{MKHS.Verify}(\mathcal{P}, \Delta, \{\mathsf{pk}_\mathsf{id}\}_{\mathsf{id}\in\mathcal{P}}, \mathsf{m}, \sigma)$. *The verification algorithm takes as input a labeled program* $\mathcal{P} = (C, (\ell_1, \ldots, \ell_\mathsf{n}))$*, a dataset identifier* Δ*, the set of public keys* $\{\mathsf{pk}_\mathsf{id}\}_{\mathsf{id}\in\mathcal{P}}$ *corresponding to those identities* id *involved in* $\mathcal{P}$*, a message* m *and an homomorphic signature* σ*. It outputs* 0 *(reject) or* 1 *(accept).*

Remark 1 (Single/Multi-Hop Evaluation). Similarly to fully homomorphic encryption, we call a (multi-key) homomorphic signature i-Hop if the Eval algorithm can be executed on its own outputs up to i times. We call *single-hop* a scheme where Eval can be executed only on fresh signatures, i.e., generated by Sign, whereas a multi-hop scheme is a scheme that is i-Hop for all i.

Authentication Correctness. A multi-key homomorphic signature satisfies authentication correctness if for all public parameters $\mathsf{pp} \leftarrow \mathsf{MKHS.Setup}(1^\lambda)$, any key pair $(\mathsf{sk}_\mathsf{id}, \mathsf{pk}_\mathsf{id}) \leftarrow \mathsf{MKHS.KeyGen}(\mathsf{pp})$, any dataset identifier Δ, any label $\ell = (\mathsf{id}, \tau) \in \mathcal{L}$, any message $\mathsf{m} \in \mathcal{M}$ and any signature $\sigma \leftarrow \mathsf{MKHS.Sign}(\mathsf{sk}, \Delta, \ell, \mathsf{m})$, it holds that $\Pr[\mathsf{MKHS.Verify}(\mathcal{I}_\ell, \Delta, \mathsf{pk}, \mathsf{m}, \sigma) = 1] \geq 1 - \mathsf{negl}$.

Evaluation Correctness. A multi-key homomorphic signature satisfies evaluation correctness if $\Pr[\mathsf{MKHS.Verify}(\mathcal{P}', \Delta, \{\mathsf{pk}_\mathsf{id}\}_{\mathsf{id}\in\mathcal{P}'}, \mathsf{m}', \sigma') = 1] \geq 1 - \mathsf{negl}$ where the equality holds for a fixed description of the public parameters $\mathsf{pp} \leftarrow \mathsf{MKHS.Setup}(1^\lambda)$, an arbitrary set of honestly generated keys $\{(\mathsf{sk}_\mathsf{id}, \mathsf{pk}_\mathsf{id})\}_{\mathsf{id}\in\tilde{\mathsf{ID}}}$ for some $\tilde{\mathsf{ID}} \subseteq \mathsf{ID}$, with $|\tilde{ID}| = t$, a dataset identifier Δ, a function $C : \mathcal{M}^\mathsf{n} \to \mathcal{M}$, and any set of program/message/signature triples $\{(\mathcal{P}_i, \mathsf{m}_i, \sigma_i)\}_{i\in[\mathsf{n}]}$ such that $\mathsf{MKHS.Verify}(\mathcal{P}_i, \Delta, \{\mathsf{pk}_\mathsf{id}\}_{\mathsf{id}\in\mathcal{P}_i}, \mathsf{m}_i, \sigma_i) = 1$ for all $i \in [\mathsf{n}]$, and $\mathsf{m}' = g(\mathsf{m}_1, \ldots, \mathsf{m}_\mathsf{n})$, $\mathcal{P}' = g(\mathcal{P}_1, \ldots, \mathcal{P}_\mathsf{n})$, and $\sigma' = \mathsf{Eval}(C, \{(\sigma_i, PK_i)\}_{i\in[n]})$ where $PK_i = \{\mathsf{pk}_\mathsf{id}\}_{\mathsf{id}\in\mathcal{P}_i}$.

Succinctness. Succinctness is one of the crucial properties that make multi-key homomorphic signatures an interesting primitive. Intuitively, a MKHS scheme is succinct if the size of every signature depends only logarithmically on the size

of a dataset. More formally, let $\mathsf{pp} \leftarrow \mathsf{MKHS.Setup}(1^\lambda)$, $\mathscr{P} = (C, (\ell_1, \ldots, \ell_\mathsf{n}))$ with $\ell_i = (\mathsf{id}_i, \tau_i)$, $(\mathsf{sk}_\mathsf{id}, \mathsf{pk}_\mathsf{id}) \leftarrow \mathsf{MKHS.KeyGen}(\mathsf{pp})$ for all $\mathsf{id} \in [n]$. and $\sigma_i \leftarrow \mathsf{MKHS.Sign}(\mathsf{sk}_{\mathsf{id}_i}, \Delta, \ell_i, \mathsf{m}_i)$, for all $i \in [\mathsf{n}]$, then MKHS has succinct signatures if there exists a fixed polynomial $\mathsf{poly}(\cdot)$ such that $size(\sigma) = \mathsf{poly}(\lambda, t, \log \mathsf{n})$ where $\sigma = \mathsf{MKHS.Eval}(\mathscr{P}, \{(\sigma_i, \mathsf{pk}_{\mathsf{id}_i})\}_{i \in [n]})$.

Security. We adopt Fiore *et al.*'s security model [17]. Very intuitively, a multi-key homomorphic signature scheme is secure if the adversary, who can request to multiple users signatures on messages of its choice, can produce only signatures that are either the ones it received, or ones that are obtained by correctly executing the Eval algorithm. In addition, in the multi-key setting the adversary is also allowed to corrupt users but this shall not affect the integrity of computations performed on data signed by other (un-corrupted) users of the system. Formally, we define the security experiment below.

Setup. The challenger $\mathscr{C}$ runs $\mathsf{MKHS.Setup}(1^\lambda)$ and sends the public parameters pp to the adversary $\mathscr{A}$.

Sign Queries. The adversary 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 $\mathsf{m} \in \mathcal{M}$ is a message. The challenger answers performing all the 1–4 checks below:

1. If (ℓ, m) is the first query for the dataset Δ, the challenger initializes an empty list $L_\Delta = \emptyset$.
2. If $(\Delta, \ell, \mathsf{m})$ is the first query with identity id, the challenger generates the keys for that identity: $(\mathsf{sk}_\mathsf{id}, \mathsf{pk}_\mathsf{id}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$. and proceeds to step 3.
3. If $(\Delta, \ell, \mathsf{m})$ is such that $(\ell, \mathsf{m}) \notin L_\Delta$, the challenger computes $\sigma \leftarrow \mathsf{MKHS.Sign}(\mathsf{sk}_\mathsf{id}, \Delta, \ell, \mathsf{m})$ (this is possible since $\mathscr{C}$ has already generated the keys for the identity id). Then the challenger updates the list $L_\Delta \leftarrow L_\Delta \cup (\ell, \mathsf{m})$ and returns $(\sigma, \mathsf{pk}_\mathsf{id})$ to $\mathscr{A}$.
4. If $(\Delta, \ell, \mathsf{m})$ is such that $(\ell, \cdot) \notin L_\Delta$, that is, the adversary had already made a query $(\Delta, \ell, \mathsf{m}')$ for some message m', the challenger ignores the query. Note that for a given (Δ, ℓ) pair only one message can be obtained.

Corruption Queries. At the beginning of the game, the challenger initialises an empty list $\mathsf{L}_\mathsf{corr} = \emptyset$ of corrupted identities. During the game, the adversary can adaptively perform corruption queries by sending $\mathsf{id} \in \mathsf{ID}$ to the challenger. If $\mathsf{id} \notin \mathsf{L}_\mathsf{corr}$ the challenger updates the list $\mathsf{L}_\mathsf{corr} \leftarrow \mathsf{L}_\mathsf{corr} \cup \mathsf{id}$ and answers the query with the pair $(\mathsf{sk}_\mathsf{id}, \mathsf{pk}_\mathsf{id})$ generated using KeyGen (if not done before). If $\mathsf{id} \in \mathsf{L}_\mathsf{corr}$ the challenger replies with keys $(\mathsf{sk}_\mathsf{id}, \mathsf{pk}_\mathsf{id})$ assigned to id before.

Forgery. At the end of the game, $\mathscr{A}$ outputs a tuple $(\mathscr{P}^*, \Delta^*, \{\mathsf{pk}^*_\mathsf{id}\}_{\mathsf{id} \in \mathscr{P}^*}, \mathsf{y}^*, \sigma^*)$. The experiment outputs 1 if the tuple returned by $\mathscr{A}$ is a forgery (defined below), and 0 otherwise.

A MK-HS scheme MKHS is *unforgeable* if for every PPT adversary $\mathscr{A}$, its advantage $\mathrm{Adv}_{\mathscr{A}^{\mathsf{MKHS}}}(\lambda) = \Pr[\mathsf{MK\text{-}HomUF\text{-}CMA}_{\mathscr{A},\mathsf{MKHS}}(\lambda) = 1]$ is $\mathsf{negl}(\lambda)$.

Definition 2 (Forgery). *We consider an execution of* MK-HomUF-CMA *where* $(\mathcal{P}^*, \Delta^*, \{\mathsf{pk}^*_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}^*}, \mathsf{y}^*, \sigma^*)$ *is the tuple returned by* $\mathcal{A}$ *at the end of the experiment. Let* $\mathcal{P}^* = (C^*, \ell^*_1, \ldots, \ell^*_{\mathsf{n}})$. *The adversary's output is said to be a successful forgery against the multi-key homomorphic signature scheme if:* $\mathsf{MKHS.Verify}(\mathcal{P}^*, \Delta^*, \{\mathsf{pk}^*_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}^*}, \mathsf{y}^*, \sigma^*) = 1$ *and at least one of the following conditions hold:*

Type-1 forgery*: the dataset* Δ^* *was never initialised.*

Type-2 forgery*: for all* $\mathsf{id} \in \mathcal{P}^*$, $\mathsf{id} \notin \mathsf{L}_{\mathsf{corr}}$ *and* $(\ell^*_i, \mathsf{m}_i) \in L_{\Delta^*}$ *for all* $i \in [\mathsf{n}]$, *but* $\mathsf{y}^* \neq C^*(\mathsf{m}_1, \ldots, \mathsf{m}_{\mathsf{n}})$.

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

Non-adaptive Corruption Queries. We also recall a proposition given in [17], which shows that it is sufficient to prove security for *non-adaptive* corruption queries. This is a setting where the adversary $\mathcal{A}$ can perform corruption queries only on identities for which no signature query had already been performed. This proposition can be used to simplify security proofs.

Proposition 1 ([17]). MKHS *is secure against adversaries that do not make corruption queries if and only if* MKHS *is secure against adversaries that make non-adaptive corruption queries.*

2.3 Homomorphic Signatures

Despite some minor syntactic modifications, homomorphic signatures can be seen as a special case of multi-key homomorphic signatures for algorithms that run on inputs by a single user only. For the purpose of this work, single-key homomorphic signature schemes are defined by five PPT algorithms $\mathsf{HS} = (\mathsf{HS.Setup}, \mathsf{HS.KeyGen}, \mathsf{HS.Sign}, \mathsf{HS.Eval}, \mathsf{HS.Verify})$ that have the same input-output behavior as the corresponding algorithms in MKHS except:

- There is no identity space ID and the labels are simply $\ell = \tau$.
- The evaluation algorithm HS.Eval takes as input a circuit C, a single public key pk and a set of signatures $\sigma_1, \ldots, \sigma_{\mathsf{n}}$. In particular HS.Eval runs without labels or dataset identifier.
- The verification algorithm HS.Verify accepts inputs from a single user only, *i.e.,* the labeled program $\mathcal{P}$ is of the form $\mathcal{P} = (C, (\tau_1, \ldots, \tau_{\mathsf{n}}))$ and only one public key pk is provided.

The properties of authentication and evaluation correctness are analogous to the ones for MKHS in the case of computations on inputs by a single client. Regarding succinctness, a homomorphic signature scheme HS has *succinct signatures* if the size of any signature σ output by HS.Eval depends only logarithmic in the number n inputs to the labelled program, *i.e., size*$(\sigma) = \mathsf{poly}(\lambda, \log(\mathsf{n}))$.

Finally, we observe that the specialization to the single-key setting of the above security definition corresponds to the strong-adaptive security definition

of HS that is formalized in [10]. In particular, the definitions in [10] allow for a simple treatment of Type-3 forgeries. In [10] it is also shown that HS constructions for circuits that are secure in this stronger model can be generically built, e.g., from [23].

3 The Matrioska Compiler

In this section, we present Matrioska: a generic compiler from a single-key homomorphic signature scheme $\mathsf{HS} = (\mathsf{HS.KeyGen}, \mathsf{HS.Sign}, \mathsf{HS.Eval}, \mathsf{HS.Verify})$ to a (single-hop) multi-key scheme $\mathsf{MKHS} = (\mathsf{MKHS.KeyGen}, \mathsf{MKHS.Sign}, \mathsf{MKHS.Eval}, \mathsf{MKHS.Verify})$. The result is summarized in the following theorem:

Theorem 2. *Let* HS *be a homomorphic signature scheme that is correct and unforgeable. Then, for any given integer number* $\mathsf{T} \geq 1$ *there exists a multi-key homomorphic signature scheme* $\mathsf{MKHS}(\mathsf{HS}, \mathsf{T})$ *that supports computations on signatures generated using at most* T *distinct keys, it is correct and unforgeable. Furthermore, if* HS *supports circuits of maximum size* s *and maximum depth* d *and it has succinctness* l*, then* $\mathsf{MKHS}(\mathsf{HS}, \mathsf{T})$ *on* T *distinct users has succinctness* $\mathsf{T} \cdot l$*, and can support circuits of size* s' *and depth* d' *provided that* $s > (s')^{{c_s}^{\mathsf{T}-1}}$ *and* $d > \max\{d', d_{\mathsf{HSV}}((s')^{{c_s}^{\mathsf{T}-1}}, \lambda)\}$*, where* d_{HSV} *and* c_s *are a function and a non-negative constant that depend from the single-key scheme* HS*.*

More precisely, d_{HSV} expresses the depth of the circuit for the verification algorithm HS.Verify as a function of its input length (which includes the description of the labeled program $\mathscr{P}$); c_s is a constant such that the size of HS.Verify on input a circuit C is $size(C)^{c_s}$. Notice that by efficiency of HS such c_s exists, and d_{HSV} can, in the worst case, be written as $size(C)^{c_d}$ for some other constant c_d.

Theorem 2 can be instantiated in two ways. If HS is a fully-homomorphic signature (whose existence is not yet known), then for any $s' = \mathsf{poly}(\lambda)$ and for any constant number T, we are guaranteed that HS is executed on poly-sized circuits. Otherwise, if HS is an HS for circuits of bounded polynomial depth (and of any, or bounded, polynomial size), as *e.g.,* [23], then for any $s' = \mathsf{poly}(\lambda)$ and for any fixed number of keys T, we can derive a polynomial bound d on the depth. The proof of Theorem 2 is constructive. First we show a method to define MKHS given a HS scheme and a value T. Next, in a sequence of lemmas, we prove all the properties stated in the theorem.

Our construction is rather involved. Therefore, in the next section we first illustrate our ideas for a simple case of a computation that takes inputs from three different users, and then, in Sect. 3.2, we describe the full compiler.

3.1 An Intuition: The Three-Client Case

We provide here a simplified example to explain the core idea of our Matrioska compiler. To ease the exposition we consider the case $t = 3$ (three clients with

identities $\mathsf{id}_1, \mathsf{id}_2$ and id_3) and deliberately remove dataset identifiers. A detailed description for $t = \mathsf{n} = 3$ can be found in the full version of this paper [18].

Let $\mathscr{P} = (C, (\ell_1, \ldots, \ell_{\mathsf{n}}))$ be a labelled program, where C a (n)-input circuit (with $\mathsf{n} = \mathsf{n}_1 + \mathsf{n}_2 + \mathsf{n}_3$) and the labels $\ell_i = (\mathsf{id}_i, \tau_i)$ are ordered, *i.e.,* first n_1 inputs belong to client id_1, the subsequent n_2 to id_2 and the last n_3 inputs to id_3. Let σ_i be the signature on message m_i for the label ℓ_i. For simplicity assume that $C(\mathsf{m}_1, \ldots, \mathsf{m}_{\mathsf{n}}) = \mathsf{y} = 1$.

Step 1. We want extract from C a circuit that contains only inputs by clients id_2 and id_3. To this end, we define E_1 as the partial evaluation of C on the messages $\mathsf{m}_{\mathsf{n}_1+1}, \ldots, \mathsf{m}_{\mathsf{n}}$. Thus, E_1 is an n_1-input circuit with hardwired in it the inputs by clients id_2 and id_3. In our framework E_1 is obtained with two basic operations on the bit string C: (1) setting any gate g with left or right input wire in $[\mathsf{n}] \setminus [\mathsf{n}_1]$ to be a constant gate (*i.e.,* setting the bits $\mathsf{L}(g)$ and $\mathsf{R}(g)$ to 0), and (2) initializing the now constant gate to the value m_i for $i \in [\mathsf{n}] \setminus [\mathsf{n}_1]$. At this point we obtained a circuit with inputs of a single client only, and we can run $\hat{\sigma}_1 \leftarrow \mathsf{HS.Eval}(E_1, \mathsf{pk}_{\mathsf{id}_1}, \sigma_1, \ldots, \sigma_{\mathsf{n}_1})$. By construction $E_1(\mathsf{m}_1, \ldots, \mathsf{m}_{\mathsf{n}_1}) = C(\mathsf{m}_1, \ldots, \mathsf{m}_{\mathsf{n}}) = 1$, therefore $\mathsf{HS.Verify}((E_1, (\tau_1, \ldots, \tau_{\mathsf{n}_1})), \mathsf{pk}_{\mathsf{id}_1}, \hat{\sigma}_1, 1) = 1$.

Step 2. The actual inductive procedure begins now. We wish to verify the correctness of $\hat{\sigma}_1$ using the messages input by client id_2 as variables. Consider the input to the (single-client) verification as the string $S_1 = ((E_1, (\tau_1, \ldots, \tau_{\mathsf{n}_1})), \mathsf{pk}_{\mathsf{id}_1}, \hat{\sigma}_1, 1)$. Recall that to construct the circuit E_1 we used the messages $\mathsf{m}_{\mathsf{n}_1+1}, \ldots \mathsf{m}_{\mathsf{n}}$ (hard-wired in its gate description). To free the inputs by client id_2 we modify S_1 in the following way: (1) identify the gates that contain the messages $\mathsf{m}_{\mathsf{n}_1+1}, \ldots, \mathsf{m}_{\mathsf{n}_1+\mathsf{n}_2}$, (2) turn these gates into input gates by setting the left/right wires to the opportune values w (using $\mathscr{P}$). Let us consider $\mathsf{HS.Verify}$ on the modified string S_1, this is a proper circuit E_2 such that $E_2(\mathsf{m}_{\mathsf{n}_1+1}, \ldots, \mathsf{m}_{\mathsf{n}_1+\mathsf{n}_2}) = \mathsf{HS.Verify}((E_1, (\tau_1, \ldots, \tau_{\mathsf{n}_1})), \mathsf{pk}_{\mathsf{id}_1}, \hat{\sigma}_1, 1) = 1$. Being E_2 a single-client circuit we can run $\hat{\sigma}_2 \leftarrow \mathsf{HS.Eval}(E_2, \mathsf{pk}_{\mathsf{id}_2}, \sigma_{\mathsf{n}_1+1}, \ldots, \sigma_{\mathsf{n}_1+\mathsf{n}_2})$.

Step 3. This is analogous to **Step 2**: we wish to verify the correctness of $\hat{\sigma}_2$ using the messages input by client id_3 as variables and define a circuit that is completely determined by public values, no hard-wired message value. Let $S_2 = ((E_2, (\tau_{\mathsf{n}_1+1}, \ldots, \tau_{\mathsf{n}_1+\mathsf{n}_2})), \mathsf{pk}_{\mathsf{id}_2}, \hat{\sigma}_2, 1)$, we free the inputs by client id_3 as in **Step 2**. We define E_3 as the formal evaluation of $\mathsf{HS.Verify}$ on the modified string S_2. By construction it holds that $E_3(\mathsf{m}_{\mathsf{n}_1+\mathsf{n}_2+1}, \ldots, \mathsf{m}_{\mathsf{n}}) = \mathsf{HS.Verify}((E_2, (\tau_{\mathsf{n}_1+1}, \ldots, \tau_{\mathsf{n}_1+\mathsf{n}_2})), \mathsf{pk}_{\mathsf{id}_2}, \hat{\sigma}_2, 1) = 1$, and we can run $\hat{\sigma}_3 \leftarrow \mathsf{HS.Eval}(E_3, \mathsf{pk}_{\mathsf{id}_3}, \sigma_{\mathsf{n}_1+\mathsf{n}_2+1}, \ldots, \sigma_{\mathsf{n}})$.

The multi-key homomorphic evaluation algorithm outputs $\hat{\sigma} = (\hat{\sigma}_1, \hat{\sigma}_2, \hat{\sigma}_3)$.

The $\mathsf{Matrioska}$ verification procedure needs only reconstruct the final circuit E_3, as this is fully determined by the public values $(\mathscr{P}, \mathsf{pk}_{\mathsf{id}_1}, \mathsf{pk}_{\mathsf{id}_2}, \hat{\sigma}_1, \hat{\sigma}_2, \mathsf{HS.Verify}, 1)$. Let $\mathscr{C}_3 = (E_3, (\tau_{\mathsf{n}_1+\mathsf{n}_2+1}, \ldots, \tau_{\mathsf{n}}))$, the verification concludes by running the single-key verification algorithm: $\mathsf{HS.Verify}(\mathscr{C}_3, \mathsf{pk}_3, \hat{\sigma}_3, 1)$.

3.2 The Matrioska Compiler

In this section we describe our compiler in the general case of computing on signatures generated by t different keys.

Definition 3 (Matrioska). *Let* $\mathsf{HS} = (\mathsf{HS.Setup}, \mathsf{HS.KeyGen}, \mathsf{HS.Sign}, \mathsf{HS.Eval}, \mathsf{HS.Verify})$ *be a single-key homomorphic signature scheme, we define a multi-key homomorphic signature scheme* MKHS *as follows:*

$\mathsf{MKHS.Setup}(1^\lambda, \mathsf{T}, s', d') \to \mathsf{pp}$. *The set-up algorithm takes as input the security parameter* λ*, a positive integer* T *that represents a bound for the maximal number of distinct identities involved in the same homomorphic computation, and bounds* $s', d' = \mathsf{poly}(\lambda)$ *on the size and depth respectively of the circuits used in the* MKHS.Eval *and* MKHS.Verify *algorithms. Setup first uses* T, s', d' *to derive two integers* s *and* d *such that* $s > (s')^{{c_s}^{\mathsf{T}-1}}$ *and* $d > \max\{d', d_{\mathsf{HSV}}((s')^{{c_s}^{\mathsf{T}-1}}, \lambda)\}$*. Next, it runs* $\mathsf{HS.Setup}(1^\lambda, s, d)$ *to obtain a tag space* $\mathcal{T}$ *(which corresponds to the label space of* HS*), a message space* $\mathcal{M}$ *and a set of admissible circuits* $\mathcal{F}$*.*[2] *Labels of the multi-key scheme are defined as pairs* $\ell = (\mathsf{id}, \tau) \in \mathsf{ID} \times \mathcal{T}$*, where the first entry is a client-identity identifier. Labeled programs are of the form* $\mathcal{P} = (C, (\ell_1, ..., \ell_t))$ *with labels as above.*

$\mathsf{MKHS.KeyGen}(\mathsf{pp}) \to (\mathsf{pk}, \mathsf{sk})$. *The multi-key key-generation algorithm runs* HS.KeyGen *to obtain a public-secret key pair. This key-pair will be associated to an identity* $\mathsf{id} \in \mathsf{ID}$*. When we need to distinguish among clients we make the dependency on the identity explicit, e.g.,* $(\mathsf{pk}_{\mathsf{id}}, \mathsf{sk}_{\mathsf{id}})$.

$\mathsf{MKHS.Sign}(\mathsf{sk}, \Delta, \ell, \mathsf{m}) \to \sigma$. *This algorithm takes as input a secret key* sk*, a data set identifier* Δ *(e.g., a string), a label* $\ell = (\mathsf{id}, \tau)$ *for the message* m*. It outputs*

$$\sigma \leftarrow \mathsf{HS.Sign}(\mathsf{sk}_{\mathsf{id}}, \Delta, \tau, \mathsf{m}). \tag{1}$$

Without loss of generality we assume that σ *includes* m.

$\mathsf{MKHS.Eval}(\mathcal{P}, \Delta, \{(\boldsymbol{\sigma}_i, \mathsf{pk}_{\mathsf{id}_i})\}_{i\in[t]}) \to \hat{\boldsymbol{\sigma}}$. *Let* $\mathcal{P} = (C, (\ell_1, \dots, \ell_{\mathsf{n}}))$*, where* $C = (\mathsf{n}, 1, \mathsf{q}, \mathsf{L}, \mathsf{R}, \mathsf{G})$ *and the* $\mathsf{n} \geq t$ *labels are of the form* $\ell_j = (\mathsf{id}_i, \tau_j)$ *for some* $i \in [t]$ *and* $\tau_j \in \mathcal{T}$*, where* $t \leq \mathsf{T}$.

The case $t = 1$ *In this case all the* n *signatures belong to the same user, that is to say, there exists an identity* $\mathsf{id} \in \mathsf{ID}$ *such that for all* $j \in [\mathsf{n}]$ *the labels are of the form* $\ell = (\mathsf{id}, \tau_j)$ *for some* $\tau_j \in \mathcal{T}$*. Thus, it is possible to run the classical evaluation algorithm of* HS *and the output of the multi-key evaluation algorithm for* $t = 1$ *is:*

$$\hat{\boldsymbol{\sigma}} = \hat{\sigma}_{\mathsf{id}} \leftarrow \mathsf{HS.Eval}\big(E_0, \mathsf{pk}_{\mathsf{id}}, (\sigma_1^{\mathsf{id}}, \dots, \sigma_{\mathsf{n}}^{\mathsf{id}})\big). \tag{2}$$

The case $t \geq 2$ *In this case the inputs to the labeled program belong to* t *distinct users. Without loss of generality, we assume that the labels are ordered per client identity, i.e., all the labels between* ℓ_{t_j} *and* $\ell_{\mathsf{t}_{j+1}-1}$ *are of the form* $(\mathsf{id}_j, *)$*. For each* $i \in [t]$ *the signature vector* $\boldsymbol{\sigma}_i$ *is* $\boldsymbol{\sigma}_i = (\sigma_1^i, \dots, \sigma_{\mathsf{n}_i}^i)$ *for opportune values* $\mathsf{n}_i \in [\mathsf{n} - t + 1]$ *satisfying* $\sum_{i=1}^t \mathsf{n}_i = \mathsf{n}$*. Let* $\mathsf{t}_i = (\sum_{j=0}^{i-1} \mathsf{n}_j) + 1$*, where*

[2] If HS works without these a-priori bounds, it is enough to run $\mathsf{HS.Setup}(1^\lambda)$.

we set $\mathsf{n}_0 = 0$*, then* t_i *corresponds to the index of first input of identity* id_i*. The multi-key homomorphic evaluation performs the following* $t+1$ *steps.*

Step 0. *Given* $\mathscr{P} = (C, (\ell_1, \ldots, \ell_{\mathsf{n}}))$ *retrieve the messages corresponding to the labels* $\ell_1, \ldots, \ell_{\mathsf{n}}$*. For notation sake let* m_j *be the message corresponding to label* ℓ_j*. Compute the value* $\mathsf{y} = C(\mathsf{m}_1, \ldots, \mathsf{m}_{\mathsf{n}})$*. Define a single-input single-output circuit* $EQ^{\mathsf{y}}(x)$ *that outputs 1 if and only if* $x = \mathsf{y}$*. Construct* $E_0 = C \triangleright EQ^{\mathsf{y}} = (\mathsf{n}, 1, \mathsf{q}_0, \mathsf{L}_0, \mathsf{R}_0, \mathsf{G}_0)$*. The properties of* EQ^{y} *imply that:*

$$E_0(x_1, \ldots, x_{\mathsf{n}}) = 1 \text{ iff } C(x_1, \ldots, x_{\mathsf{n}}) = \mathsf{y}\ . \tag{3}$$

Note that E_0 *can be constructed directly from* C *and* y*, moreover*

$$E_0(\mathsf{m}_1, \ldots, \mathsf{m}_{\mathsf{n}}) = 1. \tag{4}$$

Step 1. *We build a* n_1*-input circuit* E_1 *that corresponds to a partial evaluation of* E_0 *on the inputs of identities* id_j *with* $j > 1$*. Given* $\mathscr{C}_0 = (E_0, (\ell_1, \ldots, \ell_{\mathsf{n}}))$*, the signatures* $\boldsymbol{\sigma_1} = (\sigma_1^1, \ldots, \sigma_{\mathsf{n}_1}^1)$ *and the messages* $\mathsf{m}_{\mathsf{n}_1+1}, \ldots, \mathsf{m}_{\mathsf{n}}$ *do:*

- *Define the mask circuit* $M_1 = (\mathsf{n}_1, \mathsf{n}, \mathsf{n}, \mathsf{L}_1', \mathsf{R}_1', \mathsf{G}_1')$ *where*

$$\mathsf{L}_1'(j) = \mathsf{R}_1'(j) = \begin{cases} 1 & \text{for } j \in [\mathsf{n}_1] \\ 0 & \text{for } j \in [\mathsf{n}] \setminus [\mathsf{n}_1] \end{cases} \text{ and } \mathsf{G}_1' = \begin{cases} 0 & \text{for } j \in [\mathsf{n}_1] \\ \mathsf{m}_j & \text{for } j \in [\mathsf{n}] \setminus [\mathsf{n}_1] \end{cases}.$$

  By construction $M_1(b_1, \ldots, , b_{\mathsf{n}_1}) = (b_1, \ldots b_{\mathsf{n}_1}, \mathsf{m}_{\mathsf{n}_1+1}, \ldots, \mathsf{m}_{\mathsf{n}})$.
- *Compose* M_1 *with* E_0 *to obtain* $E_1 = M_1 \triangleright E_0 = (\mathsf{n}_1, 1, \mathsf{q}_1, \mathsf{L}_1, \mathsf{R}_1, \mathsf{G}_1)$ *where:* $\mathsf{q}_1 = \mathsf{q}_0 + \mathsf{n}$*;* $\mathsf{G}_1 = (\mathsf{G}_1' || \mathsf{G}_0)$*;* $\mathsf{L}_1(g) = \mathsf{L}_1'(g)$ *for* $g \in [\mathsf{n}]$*,* $\mathsf{L}_1(g) = (\mathsf{L}_0(g - \mathsf{n} + 1) + 1)$ *for* $g \in [\mathsf{n}+1, \mathsf{n}+\mathsf{q}_0]$ *if* $\mathsf{L}_0(g - \mathsf{n} + 1) \neq 0$ *and* 0 *whenever* $\mathsf{L}_0(g - \mathsf{n} + 1) = 0$*. The function* $\mathsf{R}_1(g)$ *is defined analogously. Equation* (4) *implies*

$$E_1(\mathsf{m}_1, \ldots, \mathsf{m}_{\mathsf{n}_1}) = 1. \tag{5}$$

- *Compute* $\hat{\sigma}_1 \leftarrow \mathsf{HS.Eval}(E_1, \mathsf{pk}_{\mathsf{id}_1}, \boldsymbol{\sigma}_1)$*. This is possible since* E_1 *is a circuit involving only inputs of client* id_1*.*

Remark 2. Let $\mathscr{C}_1 = (E_1, (\tau_1, \ldots, \tau_{\mathsf{n}_1}))$*. Equation* (5) *and the correctness of the* HS *scheme imply* $\mathsf{HS.Verify}(\mathscr{C}_1, \Delta, \mathsf{pk}_{\mathsf{id}_1}, \hat{\sigma}_1, 1) = 1$.

Step *i* ***for*** $i \in [2, t]$*. The goal is to construct an* n_i*-input circuit* E_i *using* $\mathscr{C}_{i-1} = (E_{i-1}, (\tau_{\mathsf{t}_i}, \ldots, \tau_{\mathsf{t}_{i+1}-1}))$*,* Δ*,* $\mathsf{pk}_{\mathsf{id}_i}$ *and* $\boldsymbol{\sigma_i} = (\sigma_1^i, \ldots \sigma_{\mathsf{n}_i}^i)$*. This will be possible using the circuits* $\mathsf{HSV}_i = (\mathsf{n}_{\mathsf{HSV}i}, 1, \mathsf{q}_{\mathsf{HSV}_i}, \mathsf{L}_{\mathsf{HSV}_i}, \mathsf{R}_{\mathsf{HSV}_i}, \mathsf{G}_{\mathsf{HSV}_i})$ *for the (single-key) homomorphic signature verification against the value 1.*[3]

Let $S_{i-1} = (\mathscr{C}_{i-1}, \Delta, \mathsf{pk}_{\mathsf{id}_{i-1}}, \boldsymbol{\sigma}_{i-1})$ *be a string of* $\mathsf{n}_{\mathsf{HSV}i} = size(S_{i-1})$ *bits. Set* $g_1 = 1$*. The gates of* E_{i-1} *that embed the* n_i *values input by identity* id_i *are located in the interval* $I_i = [g_i, g_i + \mathsf{n}_i]$*, where* $g_i = 3\lg(N_{i-1}) + 2\mathsf{q}_{i-1}\lg(\mathsf{w}_{i-1}) + g_{i-1} + \mathsf{n}_{i-1}$ *(see [18] for an explanation).*

[3] The readers can consider the circuit HSV_i to be the representation of $\mathsf{HS.Verify}(\mathscr{C}_{i-1}, \cdot, \cdot, 1)$ where $\mathscr{C}_{i-1}$ is a labelled program for a circuit of size at most $O((\mathsf{n}_{\mathsf{HSV}i-1} + \mathsf{q}_{\mathsf{HSV}_{i-1}})\lg(\mathsf{w}_{\mathsf{HSV}_{i-1}}))$.

- *Define the mask circuit* $M_i = (\mathsf{n}_i, \mathsf{n}_{\mathsf{HSV}\,i}, \mathsf{L}'_i, \mathsf{R}'_i, \mathsf{G}'_i)$ *where*

$$\mathsf{L}'_i(g) = \mathsf{R}'_i(g) = \begin{cases} 0 & \text{if } g \in [\mathsf{n}_{\mathsf{HSV}\,i}] \setminus I_i \\ 1 & \text{if } g \in I_i \end{cases} \quad \text{and} \quad \mathsf{G}'_i(g) = \begin{cases} S_{i-1}(g) & \text{if } g \in [\mathsf{n}_{\mathsf{HSV}\,3}] \setminus I_i \\ 0 & \text{if } g \in I_i \end{cases}$$

  Note that for gates g *in the interval* I_i, $\mathsf{L}'_i(g) = 1$ *and* $\mathsf{G}'_i(g) = 0$ *which means that* M_i *outputs its* n_i *input bits exactly the interval* I_i*, while outside* I_i *the output of* M_i *is constant. In particular:* $M_i(\mathsf{m}_{\mathsf{t}_i}, \ldots, \mathsf{m}_{\mathsf{t}_i+\mathsf{n}_i}) = S_{i-1}$.
- *Compose* M_i *with* HSV_i *to obtain* $E_i = M_i \triangleright \mathsf{HSV}_i = (\mathsf{n}_i, 1, \mathsf{q}_i, \mathsf{L}_i, \mathsf{R}_i, \mathsf{G}_i)$ *where:* $\mathsf{q}_i = \mathsf{n}_{\mathsf{HSV}\,i} + \mathsf{q}_{\mathsf{HSV}_i}$*;* $\mathsf{G}_i = (\mathsf{G}'_i || \mathsf{G}_{\mathsf{HSV}_i})$*;* $\mathsf{L}_i(g) = \mathsf{L}'_i(g)$ *for* $g \in [\mathsf{n}_{\mathsf{HSV}\,i}]$, $\mathsf{L}_i(g) = \mathsf{L}_{\mathsf{HSV}_i}(g - \mathsf{n}_{\mathsf{HSV}\,i} + 1) + \mathsf{n}_i$ *for* $g \in [\mathsf{n}_{\mathsf{HSV}\,i} + 1, \mathsf{q}_i]$ *if* $\mathsf{L}_{\mathsf{HSV}_i}(g - \mathsf{n}_{\mathsf{HSV}\,i} + 1) \neq 0$*, and* 0 *otherwise; and* R_i *is defined analogously. . Circuit composition ensures that*[4] $E_i(\mathsf{m}_{\mathsf{t}_i}, \ldots, \mathsf{m}_{\mathsf{t}_i+\mathsf{n}_i}) = \mathsf{HS.Verify}(\mathscr{C}_{i-1}, \Delta, \mathsf{pk}_{\mathsf{id}_{i-1}}, \hat{\sigma}_{i-1}, 1)$*. In particular, applying Remark 2 inductively we get:*

$$E_i(\mathsf{m}_{\mathsf{t}_i}, \ldots, \mathsf{m}_{\mathsf{t}_i+\mathsf{n}_i}) = 1 \tag{6}$$

  Note that E_i *can be constructed directly from* $\mathscr{C}_0$ *given the values* $\mathsf{m}_{\mathsf{t}_i}, \ldots, \mathsf{m}_{\mathsf{n}}$ *and the public data* Δ, $\mathsf{pk}_{\mathsf{id}_j}, \hat{\sigma}_j$ *for* $j \in [i-1]$*. In more details, for* $i \in [2, t]$ *consider the set of bit strings:* $\mathsf{head}_i = (\mathsf{n}_i, 1, \mathsf{q}_i, \mathsf{L}_i, \mathsf{R}_i)$ *and* $\mathsf{tail}_i = (\tau_{\mathsf{t}_i}, \ldots, \tau_{\mathsf{t}_i+\mathsf{n}_i}, \Delta, \mathsf{pk}_{\mathsf{id}_{i-1}}, \hat{\sigma}_{i-1}, \mathsf{G}_{\mathsf{HSV}_i})$*. For every* $i \in [2, t]$ head_i *and* tail_i *are completely determined by the tags for identity* id_{i-1}*, the public key* $\mathsf{pk}_{\mathsf{id}_{i-1}}$ *and the evaluated signature* $\hat{\sigma}_{i-1}$*. It is immediate to see that* head_i *and* tail_i *are respectively the head and the tail of the circuit description of* E_i*. The heart of the string* E_i *is where "all the magic" happens:*

$$\mathsf{body}_i = (\mathsf{head}_{i-1}, \ldots, \mathsf{head}_2, \underbrace{0, \ldots, 0}_{(\mathsf{t}_{i+1}-1)=\sum_{j=1}^{i} \mathsf{n}_j}, \mathsf{m}_{\mathsf{t}_i}, \ldots, \mathsf{m}_{\mathsf{n}}, \mathsf{G}_0, \mathsf{tail}_2, \ldots, \mathsf{tail}_i) \tag{7}$$

  In particular, for $i = t$ *we have:*

$$\begin{aligned} E_t &= \Big(\mathsf{head}_t \qquad\qquad \mathsf{body}_t \qquad\qquad \mathsf{tail}_t \Big) \\ &= \Big(\mathsf{head}_t, (\mathsf{head}_{t-1}, \ldots, \mathsf{head}_2, \underbrace{0, \ldots, 0}_{\mathsf{n}}, \mathsf{G}_0, \mathsf{tail}_2, \ldots, \mathsf{tail}_{t-1}), \mathsf{tail}_t \Big) \end{aligned} \tag{8}$$

  Equation (8) shows that the circuit E_t *is completely determined by the labeled program* $\mathscr{C}_0$ *(to get the tags and the gate description* G_0*), the dataset identifier* Δ*, the public keys* $\mathsf{pk}_{\mathsf{id}_i}$ *and the signatures* $\hat{\sigma}_i$ *for* $i \in [t]$*.*
- *Compute* $\hat{\sigma}_i \leftarrow \mathsf{HS.Eval}(E_i, \mathsf{pk}_{\mathsf{id}_i}, \boldsymbol{\sigma}_i)$.

Remark 3. This is possible since E_i *is a* n_i*-input circuit with inputs from the user* id_i *only. Equation (6) and the correctness of the* HS *scheme imply that*

$$\mathsf{HS.Verify}(\mathscr{C}_i, \Delta, \mathsf{pk}_i, 1, \hat{\sigma}_i) = 1. \tag{9}$$

[4] With abuse of notation one can think that $E_i(\mathsf{m}_{\mathsf{t}_i}, \ldots, \mathsf{m}_{\mathsf{t}_i+\mathsf{n}_i}) = M_i(\mathsf{m}_{\mathsf{t}_i}, \ldots, \mathsf{m}_{\mathsf{t}_i+\mathsf{n}_i}) \triangleright \mathsf{HSV}_i = \mathsf{HSV}_i(M_i(\mathsf{m}_{\mathsf{t}_i}, \ldots, \mathsf{m}_{\mathsf{t}_i+\mathsf{n}_i}))$. Since $M_i(\mathsf{m}_{\mathsf{t}_i}, \ldots, \mathsf{m}_{\mathsf{t}_i+\mathsf{n}_i}) = S_{i-1}$ the claim follows by the definition of HSV_i.

The output of the multi-key evaluation algorithm is the vector of t signatures: $\hat{\boldsymbol{\sigma}} = (\hat{\sigma}_1, \dots, \hat{\sigma}_t)$.

$\mathsf{MKHS.Verify}(\mathcal{P}, \Delta, \{\mathsf{pk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}}, \mathsf{y}, \hat{\boldsymbol{\sigma}}) \rightarrow \{0,1\}$. *The verification algorithm parses the labeled program as* $\mathcal{P} = (C, (\ell_1 \dots, \ell_{\mathsf{n}}))$ *and checks the number* $1 \le t \le \mathsf{T}$ *of distinct identities present among the* n *labels.*

The case $t = 1$ *In this case all the inputs to the labeled program* $\mathcal{P}$ *come from the same user and* $\hat{\boldsymbol{\sigma}} = \hat{\sigma}_{\mathsf{id}}$. *In other words, all the labels are of the form* $\ell_j = (\mathsf{id}, \tau_j)$ *for an* $\mathsf{id} \in \mathsf{ID}$ *and some* $\tau_j \in \mathcal{T}$. *Set* $\mathscr{C}_0 = (C, (\tau_1, \dots, \tau_{\mathsf{n}}))$, *notice that we removed the identity from the labels. The multi-key verification algorithm returns the output of*

$$\mathsf{HS.Verify}(\mathscr{C}_0, \Delta, \mathsf{pk}_{\mathsf{id}}, 1, \hat{\sigma}_{\mathsf{id}}). \tag{10}$$

The case $t \ge 2$ *In this case the labeled program* $\mathcal{P}$ *contains labels with* $t \ge 2$ *distinct identities and* $\hat{\boldsymbol{\sigma}} = (\hat{\sigma}_1, \dots, \hat{\sigma}_t)$. *Without loss of generality, we assume that the labels are ordered per client identity and* $\mathsf{n}_i \in [\mathsf{n} - t + 1]$ *is the number of labels with identity* id_i.

Define $E_0 = (\mathsf{n}, 1, \mathsf{q}_0, \mathsf{L}_0\mathsf{R}_0, \mathsf{G}_0)$ *as the circuit* $E_0 = C \triangleright EQ^{\mathsf{y}}$, *where* $EQ^{\mathsf{y}}(x)$ *is the a single-input single-output circuit that outputs 1 if and only if* $x = \mathsf{y}$. *Thus,* $E_0(x_1, \dots, x_{\mathsf{n}}) = 1$ *whenever* $C(x_1, \dots, x_{\mathsf{n}}) = \mathsf{y}$. *As noted in the Step 0 of the evaluation algorithm,* E_0 *is completely determined by* $\mathcal{P}$ *and* y.

To verify the signature $\hat{\boldsymbol{\sigma}}$, *the multi-key verification algorithm inductively creates the following strings for* $i \in [2, t]$:

$$\mathsf{head}_i = (\mathsf{n}_i, 1, \mathsf{q}_i = \mathsf{n}_{\mathsf{HSV}\,i} + \mathsf{q}_{\mathsf{HSV}_i}, \mathsf{L}_i = (\underbrace{0, \dots, 0}_{(\sum_{j=1}^{i-1} \mathsf{n}_j)-bits}, \overbrace{1, \dots, 1}^{\mathsf{n}_i - bits}, \underbrace{0, \dots, 0}_{(\mathsf{n}-\sum_{j=1}^{i} \mathsf{n}_j)-bits}), \mathsf{R}_i = \mathsf{L}_i)$$

$$\mathsf{tail}_i = (\tau_{\mathsf{t}_{i-1}}, \dots, \tau_{\mathsf{t}_{i-1}+\mathsf{n}_{i-1}}, \Delta, \mathsf{pk}_{\mathsf{id}_{i-1}}, \hat{\sigma}_{i-1}, \mathsf{G}_{\mathsf{HSV}_i})$$

where, the circuit HSV_i *is the same as the one explained in* $\mathsf{MKHS.Eval}$, *i.e., the* HSV_i *is the (single-key) homomorphic signature verification against the value 1. At this point the verifier can combine all the pieces to (re)-construct the description of the circuit* E_t:

$$E_t = (\mathsf{head}_t, \dots, \mathsf{head}_2, \underbrace{0, \dots, 0}_{\mathsf{n}}, \mathsf{G}_0, \mathsf{tail}_2, \dots, \mathsf{tail}_t). \tag{11}$$

Let $\mathscr{C}_t = (E_t, (\tau_{\mathsf{t}_t}, \dots, \tau_{\mathsf{n}}))$, *where we removed* id_t *from the labels. The verification returns:*

$$\mathsf{HS.Verify}(\mathscr{C}_t, \Delta, \mathsf{pk}_{\mathsf{id}_t}, \hat{\sigma}_t, 1). \tag{12}$$

Remark 4. Note that the E_t constructed by the verifier via Eq. (11) coincides with the one created by the evaluator via Eq. (8).

3.3 Correctness and Succinctness of Matrioska

In what follows we show that the Matrioska scheme satisfies the properties stated in Theorem 2.

Succinctness. By construction, for a computation involving messages from t users, our signatures consist of t signatures of the single-input scheme. It is straightforward to see that if HS signatures have length bounded by some polynomial l, the size of Matrioska's signatures is $\leq t \cdot l$, which is, asymptotically, the same level of succinctness as the MK-HS construction by Fiore *et al.* [17].

Correctness. The following two lemmas reduce the authentication and evaluation correctness of Matrioska multi-key homomorphic signatures to the authentication and evaluation correctness, respectively, of the underlying single-key HS scheme.

Lemma 1. *Let* HS *be a single-key homomorphic signature scheme with authentication correctness, then the multi-key homomorphic signature scheme* $\mathsf{MKHS}(\mathsf{HS}, \mathsf{T})$ *obtained from the* Matrioska *compiler of Definition 3 achieves authentication correctness.*

The proof is quite straightforward and uses the labeled identity program $\mathcal{I}_\ell = (C_{\mathsf{id}}, \ell)$. For details check [18].

Lemma 2. *Let* HS *be a single-key homomorphic signature scheme with evaluation correctness, then the multi-key homomorphic signature scheme* $\mathsf{MKHS}(\mathsf{HS}, \mathsf{T})$ *obtained from the* Matrioska *compiler of Definition 3 achieves evaluation correctness.*

The evaluation correctness of Matrioska essentially follows from the evaluation correctness of HS and the way we (inductively) define the circuits E_i. Moreover, notice that our MK-HS scheme is single-hop, therefore we have to prove evaluation correctness with respect to computing on freshly generated signatures (given that authentication correctness is granted by the previous lemma). For a detailed proof check [18].

Circuit Growth. In what follows we analyze the size growth of the circuits E_i computed by the Matrioska compiler, and use this to prove the bounds in Theorem 2.

Lemma 3. *Let* HS *be a correct single-key homomorphic signature scheme that supports computations on circuits of (maximum) depth* d *and size* s*; then the multi-key homomorphic signature scheme* $\mathsf{MKHS}(\mathsf{HS}, \mathsf{T})$ *obtained from the* Matrioska *compiler of Definition 3 supports homomorphic computations on circuits of size* s' *and depth* d' *provided that* $s > (s')^{c_s{}^{\mathsf{T}-1}}$ *and* $d > \max\{d', d_{\mathsf{HSV}}((s')^{c_s{}^{\mathsf{T}-1}}, \lambda)\}$*, where* d_{HSV} *and* c_s *are a function and a non-negative constant that depend on the single-key scheme* HS*.*

Intuitively, for $t = 1$, MKHS is running the plain algorithms of HS. and thus MKHS supports circuits of size $s' < s$ and depth $d' < \max\{d, d_{\mathsf{HSV}}(s)\}$. For $t > 1$ the Matrioska compiler runs $\mathsf{HS.Eval}$ and $\mathsf{HS.Verify}$ on every E_i including E_t. Since $\{E_i\}_{i\in[t]}$ is a sequence of circuits of increasing size and depth we need to make sure that the circuit given as input to MKHS will grow into an E_t that is supported by HS. The details can be found in [18].

3.4 Security of Matrioska

In this section we argue that Matrioska MKHS schemes are unforgeable provided that so is the underlying HS scheme. For the proof we rely on Proposition 1 from [17], which allows for a simpler treating of corruption queries. Due to space limit, the detailed proof appears in the full version of this paper [18] while below we give a proof sketch with the main intuition.

Lemma 4. *Let* HS *be a secure single-key homomorphic signature scheme. Then the multi-key homomorphic signature scheme* MKHS(HS, T) *obtained from the* Matrioska *compiler of Definition 3 is secure. In particular, for any PPT adversary* $\mathcal{A}$ *making signing queries on at most* $Q_{\mathsf{id}} = \mathsf{poly}(\lambda)$ *distinct identities, there is a PPT algorithm* $\mathcal{B}$ *such that:* $Adv_{\mathcal{A}}^{\mathsf{MKHS}} \leq Q_{\mathsf{id}} \cdot Adv_{\mathcal{B}}^{\mathsf{HS}}$.

Proof Sketch. The idea is that a forger against our MKHS scheme must create a forgery for the HS scheme for at least one of the users, say $\mathsf{id}_{i^\star}$, involved in the computation. Thus the reduction $\mathcal{B}$, on input a public key pk, makes a guess for $j^* = i^\star$, programs $\mathsf{pk}_{\mathsf{id}_{j^*}} = \mathsf{pk}$ and generates all the other keys. This allows $\mathcal{B}$ to perfectly simulate all the signing queries (perfectly hiding j^* to $\mathcal{A}$).

When $\mathcal{A}$ returns $(\mathcal{P}^*, \Delta^*, \{\mathsf{pk}^*_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{P}^*}, \mathsf{y}^*, \sigma^*)$, with $\sigma^* = (\hat{\sigma}^*_1, \ldots, \hat{\sigma}^*_t)$, the crucial part of the proof is showing the existence of an index $i^\star$ such that $\hat{\sigma}^*_{i^\star}$ is a forgery for HS. Specifically:

- $\sigma^*$is of type-1 (Δ^* is new). Then $i^\star = t$ and $\hat{\sigma}^*_t$ is a type-1 forgery against HS.
- $\sigma^*$is of type-2. This means: $E_0(\mathsf{m}_1, \ldots, \mathsf{m}_\mathsf{n}) = 0$ while $\mathsf{HS.Verify}(\mathscr{C}_t, \mathsf{pk}_{\mathsf{id}_t}, 1, \hat{\sigma}^*_t) = 1$. Then we show that there must exist a "forking index" $i^\star \in [t]$ such that $E_{i-1}(\mathsf{m}_{\mathsf{t}_{i-1}}, \ldots, \mathsf{m}_{\mathsf{t}_{i-1}+\mathsf{n}_{i-1}}) = 0$ but $\mathsf{HS.Verify}(\mathscr{C}_i, \mathsf{pk}_{\mathsf{id}_i}, \hat{\sigma}^*_i, 1) = 1$, that is, $\hat{\sigma}^*_{i^\star}$ is a type-2 forgery against HS for the labeled program $\mathscr{C}_i$.
- $\sigma^*$is of type-3. If $t = 1$, then $i^\star = 1$ and $\hat{\sigma}^*_1$ is a type-3 forgery against HS. If $t > 1$, let $i \in [t]$ be the first index such that $\exists\, j \in [\mathsf{n}] : \ell_j = (\mathsf{id}_i, \tau_j) \notin L_{\Delta^*}$, *i.e.,* the first identity for which a type-3 forgery condition holds. Then, either $\hat{\sigma}^*_i$ is a type-3 forgery for HS for identity id_i (and thus $i^\star = i$); or there is $i^\star > i$ such that $\hat{\sigma}^*_{i^\star}$ is a type-2 forgery against identity $\mathsf{id}_{i^\star}$. The latter can be argued by showing the existence of a "forking index" as in the previous case. In a nutshell, a type-3 forgery against MKHS comes either from a type-3 forgery at some index i, or, the i-th signature is incorrect and thus there must be a type-2 forgery at a later index to cheat on the fact that verification at index i is correct.

Therefore, if $j^* = i^\star$ (which happens with non-negligible probability $1/Q_{\mathsf{id}}$), $\mathcal{B}$ can convert $\mathcal{A}$'s forgery into one for its challenger.

4 Conclusions and Future Work

In this paper, we presented Matrioska the first generic compiler based on falsifiable assumptions that establishes a formal connection between single-key HS and

multi-key HS schemes. Matrioska introduces an original mechanism to gain multi-key features by levering the homomorphic property of a single-key HS scheme. The resulting signatures are succinct in the sense that their length depends solely on the number of signers involved in the homomorphic computation, and not on the total number of signatures input. Unfortunately, constructions obtained with Matrioska are of limited efficiency, as they require the single-key HS scheme to support circuits of size exponentially large in the maximum number of distinct signers involved in the computation. Achieving full signature succinctness remains an interesting goal for further developments, as well as investigating if Matrioska's approach could be used to enhance other cryptographic primitives with multi-key features.

Acknowledgments. This work was partially supported by the COST Action IC1306 through a STSM grant to Elena Pagnin. Dario Fiore was partially supported by the Spanish Ministry of Economy under project references TIN2015-70713-R (DEDETIS), RTC-2016-4930-7 (DataMantium), and by the Madrid Regional Government under project N-Greens (ref. S2013/ICE-2731).

References

1. Attrapadung, N., Libert, B.: Homomorphic network coding signatures in the standard model. In: Catalano, D., Fazio, N., Gennaro, R., Nicolosi, A. (eds.) PKC 2011. LNCS, vol. 6571, pp. 17–34. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19379-8_2
2. Attrapadung, N., Libert, B., Peters, T.: Computing on authenticated data: new privacy definitions and constructions. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 367–385. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34961-4_23
3. Bellare, M., Hoang, V.T., Rogaway, P.: Foundations of garbled circuits. In: Yu, T., Danezis, G., Gligor, V.D. (eds.) ACM CCS 2012, pp. 784–796. ACM Press (2012)
4. Boneh, D., Freeman, D., Katz, J., Waters, B.: Signing a linear subspace: signature schemes for network coding. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 68–87. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00468-1_5
5. Boneh, D., Freeman, D.M.: Homomorphic signatures for polynomial functions. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 149–168. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20465-4_10
6. Boneh, D., Freeman, D.M.: Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In: Catalano, D., Fazio, N., Gennaro, R., Nicolosi, A. (eds.) PKC 2011. LNCS, vol. 6571, pp. 1–16. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19379-8_1
7. Catalano, D., Fiore, D.: Practical homomorphic message authenticators for arithmetic circuits. J. Cryptol. **31**(1), 23–59 (2018)
8. Catalano, D., Fiore, D., Gennaro, R., Vamvourellis, K.: Algebraic (trapdoor) one-way functions and their applications. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 680–699. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36594-2_38

9. Catalano, D., Fiore, D., Gennaro, R., Vamvourellis, K.: Algebraic (trapdoor) one-way functions: constructions and applications. Theor. Comput. Sci. **592**, 143–165 (2015)
10. Catalano, D., Fiore, D., Nizzardo, L.: On the security notions for homomorphic signatures. In: Preneel, B., Vercauteren, F. (eds.) ACNS 2018. LNCS, vol. 10892, pp. 183–201. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-93387-0_10
11. Catalano, D., Fiore, D., Nizzardo, L.: Programmable hash functions go private: constructions and applications to (homomorphic) signatures with shorter public keys. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9216, pp. 254–274. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48000-7_13
12. Catalano, D., Fiore, D., Warinschi, B.: Adaptive pseudo-free groups and applications. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 207–223. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20465-4_13
13. Catalano, D., Fiore, D., Warinschi, B.: Efficient network coding signatures in the standard model. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) PKC 2012. LNCS, vol. 7293, pp. 680–696. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30057-8_40
14. Catalano, D., Fiore, D., Warinschi, B.: Homomorphic signatures with efficient verification for polynomial functions. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8616, pp. 371–389. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44371-2_21
15. Catalano, D., Marcedone, A., Puglisi, O.: Authenticating computation on groups: new homomorphic primitives and applications. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8874, pp. 193–212. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45608-8_11
16. Desmedt, Y.: Computer security by redefining what a computer is. In: NSPW (1993)
17. Fiore, D., Mitrokotsa, A., Nizzardo, L., Pagnin, E.: Multi-key homomorphic authenticators. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10032, pp. 499–530. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53890-6_17
18. Fiore, D., Pagnin, E.: Matrioska: a compiler for multi-key homomorphic signatures. IACR Cryptology ePrint Archive (2018)
19. Freeman, D.M.: Improved security for linearly homomorphic signatures: a generic framework. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) PKC 2012. LNCS, vol. 7293, pp. 697–714. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30057-8_41
20. Gennaro, R., Katz, J., Krawczyk, H., Rabin, T.: Secure network coding over the integers. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 142–160. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13013-7_9
21. Gennaro, R., Wichs, D.: Fully homomorphic message authenticators. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013. LNCS, vol. 8270, pp. 301–320. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-42045-0_16
22. Gentry, C., Wichs, D.: Separating succinct non-interactive arguments from all falsifiable assumptions. In: Fortnow, L., Vadhan, S.P. (eds.) 43rd ACM STOC, pp. 99–108. ACM Press (2011)
23. Gorbunov, S., Vaikuntanathan, V., Wichs, D.: Leveled fully homomorphic signatures from standard lattices. In: Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing, pp. 469–477. ACM (2015)

24. Johnson, R., Molnar, D., Song, D., Wagner, D.: Homomorphic signature schemes. In: Preneel, B. (ed.) CT-RSA 2002. LNCS, vol. 2271, pp. 244–262. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45760-7_17
25. Lai, R.W., Tai, R.K., Wong, H.W., Chow, S.S.: Multi-key homomorphic signatures unforgeable under insider corruption. IACR Cryptology ePrint Archive 2016/834 (2016)
26. Libert, B., Peters, T., Joye, M., Yung, M.: Linearly homomorphic structure-preserving signatures and their applications. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8043, pp. 289–307. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40084-1_17