Tracing a Linear Subspace: Application to Linearly-Homomorphic Group Signatures

Chloé Hébant[1], David Pointcheval[2], and Robert Schädlich[2]

[1] Cosmian, Paris, France
[2] DIENS, École normale supérieure, PSL University, CNRS, Inria, Paris, France

Abstract. When multiple users have power or rights, there is always the risk of corruption or abuse. Whereas there is no solution to avoid those malicious behaviors, from the users themselves or from external adversaries, one can strongly deter them with tracing capabilities that will later help to revoke the rights or negatively impact the reputation. On the other hand, privacy is an important issue in many applications, which seems in contradiction with traceability.
In this paper, we first extend usual tracing techniques based on codes so that not just one contributor can be traced but the full collusion. In a second step, we embed suitable codes into a set $\mathcal{V}$ of vectors in such a way that, given a vector $\mathbf{U} \in \mathsf{span}(\mathcal{V})$, the underlying code can be used to efficiently find a minimal subset $\mathcal{X} \subseteq \mathcal{V}$ such that $\mathbf{U} \in \mathsf{span}(\mathcal{X})$.
To meet privacy requirements, we then make the vectors of $\mathsf{span}(\mathcal{V})$ anonymous while keeping the efficient tracing mechanism. As an interesting application, we formally define the notion of linearly-homomorphic group signatures and propose a construction from our codes: multiple signatures can be combined to sign any linear subspace in an anonymous way, but a tracing authority is able to trace back all the contributors involved in the signatures of that subspace.

1 Introduction

In any multi-user setting, a user can always share its secret key with a non-legitimate one or get corrupted, which delegates all its rights. One way to escape from such a situation is to make it useless: in threshold cryptography, such keys are useless unless enough keys are obtained. Another approach consists in deterring traitors to share their keys by activity tracing. This idea has been introduced by Chor *et al.* in [CFN94], initially to recover the origin of a pirate decoder box decrypting broadcast messages, such as for PayTV. In this use-case, tracing one traitor at a time makes sense, as the broadcast can continue after having revoked the first traitor. If the pirate decoder is still effective, other traitors can sequentially be traced and revoked. However, things are different if the setting is rather "static", such as when signatures are created jointly by several users. Here, tracing one traitor at a time is meaningless, as the signature cannot be replayed to retrieve all the traitors one after one. We therefore develop techniques that allow to retrieve not only one but all the actual contributors.

1.1 Contributions

Linearly-Homomorphic Group Signatures. In this work, we build multi-user signatures that include tracing capabilities. More specifically, we combine functionality and security guarantees of group signatures [Cv91,BMW03] and linearly-homomorphic signatures [BFKW09]. The former roughly guarantees that, given signatures for vector messages $\mathbf{M}_1, \ldots, \mathbf{M}_n$, anyone can sign elements in the span of these messages. The latter is a multi-user signature scheme which allows members of a (fixed) *group* to sign anonymously on behalf of this group, except towards an authority called the *group manager* that is able to revoke anonymity. Our new primitive puts linearly-homomorphic signatures into a multi-user setting with broad functionality by allowing the aggregation of arbitrary signatures, whether freshly created by different group members or previously aggregated. In addition, this comes along with the strong security guarantees of a group signature. From a privacy point of view, signatures can only be associated with a group, but not with individual members. Group members act thus anonymously, even

towards other members of the group. However, to avoid malicious behavior, the group manager is able to recover the actual creators of a signature. We emphasize that, in contrast to classic group signatures, a linearly-homomorphic group signature can have multiple contributors after aggregation, which generally requires the group manager to identify a set of signers rather than a single one. As a corrupted group member can always mix their own signatures with that of honest signers, it is crucial for the group manager to recover *all* contributors to a signature. Any strict subset of the contributors is vacuous, as there is no guarantee that it contains the malicious ones.

Linear-Subspace Tracing. Our main technical contribution is the construction of an object that we call a *linear-subspace tracing (LST)* scheme. Let $\mathbb{U}$ be a vector space and c a positive integer, as a bound on the size of the collusion. Informally, a LST scheme solves the problem of finding sets $\mathcal{V} \subset \mathbb{U}$ such that the following task can be solved *efficiently* and *uniquely*:

> *On input a vector* $\mathbf{U} \in \bigcup_{\mathcal{X} \subseteq \mathcal{V}, |\mathcal{X}| \leq c} \mathsf{span}(\mathcal{X})$*, recover a minimal (a.k.a. the smallest) subset* $\mathcal{X}_0 \subseteq \mathcal{V}$ *such that* $\mathbf{U} \in \mathsf{span}(\mathcal{X}_0)$*.*

As each subset of $\mathcal{V}$ spans a linear subspace, solving this task can be viewed as "tracing" the smallest subspace that contains the given vector $\mathbf{U}$. Of course, there exist trivial constructions. For example, given $\mathbb{U} = \mathbb{Z}_p^\ell$ for a prime p and a positive integer ℓ, one can choose $\mathcal{V} = \{\mathbf{e}_i : i \in [\ell]\}$ where $\mathbf{e}_i$ denotes the i-th standard unit vector. It even allows large collusions, as one can take $c = \ell$, where ℓ is the cardinality of the set $\mathcal{V}$ and also the maximal size c of the collusion. However, the situation gets more intricate when we try to make the construction more efficient (i.e., for a dimension ℓ smaller than the cardinality n of the set) but for possibly bounded collusions (of maximal size $c \leq n$). Note that in the example above the required dimension of $\mathbb{U}$ grows linearly with the cardinality $\mathcal{V}$, as $n = \ell$. In this work, we consider vector spaces where the discrete logarithm problem in the additive group is assumed to be hard. We present LST schemes for vector spaces of dimension $\ell = \Omega(c^2 \cdot \log(n/\varepsilon))$, where ε denotes the maximum acceptable probability that the tracing will fail.

To meet privacy requirements, we also construct an *anonymous* LST scheme, which informally means that recovering the set $\mathcal{X}_0$ is computationally hard except one holds a special tracing key. We are able to prove anonymity without increasing the lower bound on ℓ. However, we need the stronger assumption that the Decisional Diffie-Hellman problem (instead of only the discrete logarithm) is hard in the additive group of the vector space.

Fully IPP Codes. A formal model to address the traitor tracing problem are codes with the *Identifiable Parent Property (IPP)* introduced by Hollmann *et al.* [HvLT98]. A word $\mathbf{u} = (u_k)$ is a *descendant* of a coalition $\mathcal{X}$ of codewords if every letter u_k is present in at least one codeword of $\mathcal{X}$ at the same position k. Conversely, the elements of $\mathcal{X}$ are called parents. Intuitively, a code $\mathcal{C}$ satisfies the Identifiable Parent Property if for any descendant of $\mathcal{C}$, *at least one* parent codeword can be identified with certainty. In contrast, this work pursues the stronger goal of identifying *all* parent codewords of a word $\mathbf{u}$. We formalize this aspect by defining a variant of IPP codes that we call *fully* IPP (FIPP), as it allows to identify a full coalition instead of just a single parent codeword. For consistency, we then must require that there exists a unique minimal (w.r.t. $\subseteq$) set $\mathcal{X}_0$ among all sets $\mathcal{X} \subseteq \mathcal{C}$ with the property that $\mathbf{u}$ is a descendant of $\mathcal{X}$. Otherwise, it would be impossible to determine which of several minimal subsets was used to derive $\mathbf{u}$.

In the literature, IPP codes are only considered with respect to the above-mentioned notion of descendants. However, the general concept of IPP (or FIPP) can also be studied using other descendancy relations. We primarily use FIPP codes as a building block for our LST scheme. Since this use case differs significantly from the original purpose of tracing traitors in broadcast encryption, a modified definition of descendants proves to be more suitable. We therefore

note that our construction of FIPP codes is not a contribution to the theory of classical IPP codes. We use the terminology merely to give the reader a better intuition. For readers familiar with fingerprinting codes [BS95,Tar03], we further remark that—like the standard definition of descendants—our new version can be seen as a strengthening of the Marking assumption. As the concepts of IPP and fingerprinting codes are very similar, our construction could also be interpreted as a fingerprinting code with a modified Marking assumption.

1.2 Technical Overview

Constructing FIPP Codes. We start with the definition of descendants used throughout this work. Let $\mathcal{Q} = \{a_1, \ldots, a_n, \bot, \top\}$ be an alphabet with two distinguished letters: the *neutral* letter $\top$ and the letter $\bot$ which represents a *collision*. Roughly, a word $\mathbf{u} = (u_k)$ is defined to be a *descendant* of a coalition $\mathcal{X}$ of codewords if there exists a subset $\mathcal{X}_0 \subseteq \mathcal{X}$ such that the following condition is satisfied for all positions k: if all letters that occur at the k-th position of a codeword in $\mathcal{X}_0$ equal either $\top$ or some $a \in \mathcal{Q}$, then $u_k = a$ too. Otherwise, $u_k = \bot$. Intuitively, descendants preserve information at those positions where all parents coincide or equal the neutral letter $\top$, but lose all information elsewhere (since there is a collision of different letters). The condition that there exists a subset $\mathcal{X}_0$ of $\mathcal{X}$ is necessary to cover the case that not all codewords of $\mathbf{X}$ are actively used to derive the descendant $\mathbf{u}$. However, for ease of exposition, we implicitly assume that $\mathcal{X}_0 = \mathcal{X}$ here.

We next describe the construction of our FIPP codes $\mathcal{C} = \{\mathbf{w}_1, \ldots, \mathbf{w}_n\}$ with respect to this definition of descendants. Recall that $\mathcal{Q}$ is of the form $\{a_1, \ldots, a_n, \bot, \top\}$. In our construction, we impose the restriction that the i-th codeword $\mathbf{w}_i$ includes only the letters a_i and $\top$. In this case, the descendancy definition implies that any word containing the letter a_i must have $\mathbf{w}_i$ as a parent codeword since there is no other way to derive a word which contains this letter. Thus, the choice of $\mathbf{w}_1, \ldots, \mathbf{w}_n$ boils down to a balls-into-bins problem, where we must arrange the letters of the codewords in such a way that the number of collisions is small. Our approach is very simple: we divide all codewords into multiple blocks of equal size. Then for each $i \in [n]$, $\mathbf{w}_i$ is chosen such that each of its blocks contains the letter a_i at exactly one position, and $\top$ elsewhere.

Let $\mathbf{u}$ be a descendant of a coalition $\mathcal{X} \subseteq \mathcal{C}$. The tracing algorithm simply outputs the set $\{\mathbf{w}_i \in \mathcal{C} : \mathbf{u} \text{ contains } a_i\}$. As argued above, this method never "accuses" a wrong codeword. Conversely, we need to bound the probability that a parent codeword $\mathbf{w}_i$ is not detected. This happens if there are collisions in *all* positions of $\mathbf{u}$ where $\mathbf{w}_i$ contains the latter a_i. Inside a fixed block, this probability is constant. Since the blocks of $\mathbf{w}_i$ are chosen independently, the probability that there is such a collision in all blocks of $\mathbf{u}$ decreases exponentially in the number of blocks. Thus, each parent codeword of $\mathbf{u}$ is detected with high probability.

Embedding Codewords into Vectors. We next describe our construction of a LST scheme. We start with the generation of the set $\mathcal{V}$ of vectors. At a high level, we embed a FIPP code $\mathcal{C} = \{\mathbf{w}_1, \ldots, \mathbf{w}_n\}$ into the set $\mathcal{V} = \{\mathbf{V}_1, \ldots, \mathbf{V}_n\}$ such that vectors in the span of an arbitrary subset $\mathcal{X} = \{\mathbf{V}_{i_1}, \ldots, \mathbf{V}_{i_d}\} \subseteq \mathcal{V}$ represent descendants of the corresponding set $\mathcal{Y} = \{\mathbf{w}_{i_1}, \ldots, \mathbf{w}_{i_d}\}$ of codewords. Thus, given a vector $\mathbf{U} \in \bigcup_{\mathcal{X} \subseteq \mathcal{V}, |\mathcal{X}| \leq c} \mathsf{span}(\mathcal{X})$, we can compute the smallest subset $\mathcal{X}_0 \subseteq \mathcal{V}$ such that $\mathbf{U} \in \mathsf{span}(\mathcal{X}_0)$ by recovering the word $\mathbf{u}$ embedded in the vector $\mathbf{U}$, followed by an execution of the code's tracing algorithm which reveals all parents of $\mathbf{u}$.

More specifically, we embed codewords of length ℓ into vectors of $\mathbb{G}^{2\ell}$ where $\mathbb{G}$ is a cyclic group of prime order p with a generator G. For each $i \in [n]$, we sample a random scalar $s_i \xleftarrow{\$} \mathbb{Z}_p$. We then construct $\mathbf{V}_i$ by replacing each occurrence of the letter a_i in the codeword $\mathbf{w}_i$ with the tuple $(G, s_i \cdot G)$ and each occurrence of $\top$ with (G, G). Let k be a position where all codewords in $\mathcal{Y}$ agree with either $\top$ or the letter a_{i_j} for some $j \in [d]$. Note that the construction of our

FIPP code ensures that such positions exist for all choices of j with high probability. A simple computation shows that the coordinates $(2k-1, 2k)$ of any vector $\mathbf{U}$ in the span of $\mathcal{X}$ are of the form $(H, s_i \cdot H)$, for some $H \in \mathbb{G}$. If we identify each element of $\{(H, s_i \cdot H) : H \in \mathbb{G}^*\}$ with the letter a_i for $i \in [n]$, then vectors in the span of $\mathcal{X}$ correspond to descendants of $\mathcal{Y}$ as desired. Note that it is easy to recover the descendant $\mathbf{u}$ of $\mathcal{Y}$ corresponding to $\mathbf{U}$ even if the discrete logarithm problem in $\mathbb{G}$ is hard. Indeed, this can be done by testing for each tuple (H, H') of $\mathbf{U}$ if $H' = s \cdot H$ for some $s \in \{s_1, \ldots, s_n\}$. Once $\mathbf{u}$ is recovered, one runs the code's tracing algorithm to obtain its parents.

Anonymization. The basic idea to obtain anonymous versions $\mathbf{V}'_1, \ldots, \mathbf{V}'_n$ of $\mathbf{V}_1, \ldots, \mathbf{V}_n$ is to encrypt them component-wise using the ElGamal encryption scheme [ElG84]. It is well-known that this encryption scheme is partially homomorphic. In our context, this means that for all coefficients $\omega_1, \ldots, \omega_n$, $\sum_{i=1}^n \omega_i \cdot \mathbf{V}'_i$ is a component-wise encryption of $\sum_{i=1}^n \omega_i \cdot \mathbf{V}_i$. Thus, the tracing of an encrypted vector can be done by first decrypting it and passing the resulting vector to the original tracing algorithm. Since the security of ElGamal is based on the DDH assumption, it is straightforward to exploit the random self-reducibility of the DDH problem. If we reuse the same randomness in the encryption of all components of a vector $\mathbf{V}_i$, then one component of all ciphertexts is equal and, thus, must be included only once in the anonymous version of $\mathbf{V}_i$. This leads us to a more efficient transformation where anonymous vectors have only one additional coordinate.

Linearly-Homomorphic Group Signatures. We finally explain how our LST scheme can be used to build linearly-homomorphic group signatures. A well-known blueprint for the construction of group signatures works as follows (see [BMW03]): each group member has a secret signing key that includes a key pair of a classical signature scheme certified by the group manager. To sign a message m in the name of the group, a group member signs m using its private key and encrypts this signature together with certificate and identity information under a public encryption key held by the group manager. The final group signature consists of this ciphertext accompanied by a non-interactive zero-knowledge proof that it contains what it is supposed to contain.

To make this framework linearly-homomorphic, one needs to define a suitable aggregation operation for signatures. In particular, it raises the question of how to aggregate the ciphertexts which encrypt the identities. Naively, one could simply append all the ciphertexts when aggregating a signature. One disadvantage of this method is that the size of signatures grows linearly with the number of involved signers. But even worse, this construction leaks the information of how many signers participated in the creation of the signature, thus breaking anonymity. Therefore, one needs a mechanism which allows to aggregate the ciphertexts in such a way that

1. the aggregated ciphertext does not leak the number of signers (nor any other information about the identities of the signers), and
2. the originally encrypted identities can be recovered after decryption of the aggregated ciphertext.

To solve the first issue, it is straightforward to use a (partially) homomorphic encryption scheme like ElGamal, and to aggregate signatures by applying the homomorphic operation of the encryption scheme. The second requirement, however, is much more challenging.

Tracing contributors to a signature. As a starting point, one could assign a codeword of a classical traceable code (e.g. IPP codes or fingerprinting codes) to each group member. One then forces group members to sign messages together with an encryption of a vector which embeds the signer's codeword in the exponents. This must be done in such a way that the homomorphic operation of the encryption scheme respects the descendancy relation (resp. the

Marking assumption in the context of fingerprinting codes). Then, to trace a signature, one decrypts the vector, recovers the challenge word from the exponents and runs the code's tracing algorithm.

This idea, however, runs into a major problem. For the unforgeability of the underlying linearly-homomorphic signature scheme, the discrete logarithm problem must be hard in the additive group of the vector space. Therefore, we cannot hope to recover the challenge word from the exponents to run the tracing algorithm. Also, running the tracing algorithm directly in the exponent seems difficult.

As a solution we use our above-described code construction which comes with the property that embeddings of codewords into vectors can still be traced, even if the discrete logarithm is hard. (Recall that we call this object a LST scheme.) Roughly, we assign to each group member a vector of a set $\mathcal{V} = \{\mathbf{V}_1, \ldots, \mathbf{V}_n\}$ generated using a LST scheme. We force group members to sign each message together with their respective vector. When multiple signatures are aggregated, one obtains a new vector $\mathbf{U}$ which lies in the span of those vectors $\mathbf{V}_i$ whose associated group members contributed to the derived signature. Using the tracing algorithm of the LST scheme, it is possible to recover exactly the subset of $\mathcal{V}$ which corresponds to the contributors of the signature. To achieve privacy for the group members, we use an anonymous LST scheme where the tracing key is only known to the group manager.

1.3 Organization

The rest of the article is organized as follows. The next section recalls assumptions and definitions that we will use in the paper. Section 3 formally introduces FIPP codes and presents a construction with respect to our new definition of descendant. Section 4 defines and provides a construction of a LST scheme which is made anonymous in Section 5. Finally, our work can be used to create a linearly-homomorphic group signature scheme for which a formal model is provided in Section 6 and the construction in Section 7.

2 Preliminaries

For integers i and j, we write $[i; j]$ to denote the integer interval $\{k \in \mathbb{Z} : i \leq k \leq j\}$. By default, we set $[j] = [1, j]$. For any $q \geq 2$, we let $\mathbb{Z}_q$ denote the ring of integers with addition and multiplication modulo q. Furthermore, for an integer n and a group $(\mathbb{G}, +)$ of prime order p, we interpret $\mathbb{G}^n$ as a vector space over $\mathbb{Z}_p$ in the usual way. We denote the zero element of this vector space by $\mathbf{0}^{(n)}$. Given a vector $\boldsymbol{\omega} = (\omega_i)_{i=1}^n \in \mathbb{Z}_p^n$ and a group element $G \in \mathbb{G}$, we write $\boldsymbol{\omega} \cdot G$ to denote the vector $(\omega_i \cdot G)_{i=1}^n$. Also, we refer to the set of generators of a cyclic group $\mathbb{G}$ by $\mathbb{G}^*$. Given a set $\mathcal{X}$ and an integer $0 \leq c \leq |\mathcal{X}|$, we denote by $\mathcal{P}_c(\mathcal{X})$ the set of all subsets of size at most c, i.e. $\mathcal{P}_c(\mathcal{X}) = \{\mathcal{X}' \subseteq \mathcal{X} : |\mathcal{X}'| \leq c\}$.

2.1 Hardness Assumptions

We recall the assumptions needed for our constructions.

Definition 1 (Discrete Logarithm (DL) Assumption). *The* DL *assumption in a group* $(\mathbb{G}, +)$*, of prime order* p *with generator* G*, states that given* $H = x \cdot G$*, no algorithm can efficiently recover* x*.*

Definition 2 (Decisional Diffie-Hellman (DDH) Assumption). *The* DDH *assumption in a group* $(\mathbb{G}, +)$*, of prime orders* p *with generator* G*, states that no algorithm can efficiently distinguish the two distributions*

$$D_0 = \left\{(x \cdot G, y \cdot G, xy \cdot G) : x, y \xleftarrow{\$} \mathbb{Z}_p\right\}$$
$$D_1 = \left\{(x \cdot G, y \cdot G, z \cdot G) : x, y, z \xleftarrow{\$} \mathbb{Z}_p\right\}$$

We prove some of our results in the *algebraic group model* (AGM) – a computational model in which all adversaries are modeled as algebraic. Let $\mathbb{G}$ be a group of prime order p. Roughly, an algorithm A is called *algebraic* if for all group elements $G \in \mathbb{G}$ output by A, it additionally provides the representation of G relative to all previously received group elements. For a clean definition of the AGM see [FKL18].

Real-world implementations of bilinear structures sometimes allow the construction of group elements without knowing their discrete logarithm. Therefore, it seems important to prove results in a model that takes this property of concrete instantiations into account, e.g. a variant of the generic group model (GGM) where the adversary has access to an oracle that generates random group elements. We remark that besides the classical GGM, this (and similar) modifications are also covered by the AGM. Intuitively, an adversary in the "GGM-R" (GGM with additional oracle for random group elements) can be used to construct an algebraic adversary since all generated random elements and all group operations computed by the GGM-R adversary are known and, thus, the GGM-R simulator is able to extract itself the representations of all elements submitted by the adversary.

2.2 Linearly-Homomorphic Signatures

Linearly-homomorphic signatures were originally introduced by Boneh *et al.* in [BFKW09]. Our definition is similar to that in [LPJY13].

Definition 3 (Linearly-Homomorphic Signature Scheme (LH-Sig)). *A LH-Sig scheme with tag space $\mathcal{T}$ and message space $\mathbb{G}^n$, for a cyclic group $(\mathbb{G}, +)$ of prime order p and a positive integer n, is a collection of five polynomial-time algorithms defined as follows.*

$\mathsf{Setup}(1^\lambda)$*: On input the security parameter λ, this algorithm returns the public parameters* pp.
$\mathsf{KeyGen}(\mathsf{pp})$*: On input the public parameters* pp*, this algorithm returns a key pair* $(\mathsf{sk}, \mathsf{pk})$*. We will assume that* pk *implicitly contains* pp *and* sk *implicitly contains* pk.
$\mathsf{Sign}(\mathsf{sk}, \tau, \mathbf{M})$*: On input a secret key* sk*, a tag $\tau \in \mathcal{T}$ and a message $\mathbf{M} \in \mathbb{G}^n$, this algorithm returns a signature Σ of $\mathbf{M}$ under the tag τ.*
$\mathsf{DeriveSign}(\mathsf{pk}, \tau, (\omega_j, \Sigma_j)_{j=1}^d)$*: On input a public key* pk*, a tag $\tau \in \mathcal{T}$ and d tuples of weights $\omega_j \in \mathbb{Z}_p$ and signatures Σ_j, this algorithm returns a signature Σ on the vector $\mathbf{M} = \sum_{j=1}^d \omega_j \cdot \mathbf{M}_j$ under the tag τ, where Σ_j is a signature on the message $\mathbf{M}_j \in \mathbb{G}^n$ under τ.*
$\mathsf{Verify}(\mathsf{pk}, \tau, \mathbf{M}, \Sigma)$*: On input a public key* pk*, a tag $\tau \in \mathcal{T}$, a message $\mathbf{M} \in \mathbb{G}^n$ and a signature Σ, this algorithm returns 1 if $\tau \in \mathcal{T}$ and Σ is valid relative to* pk *and τ, and 0 otherwise.*

Correctness. The basic consistency requirement of every signature scheme is that honestly generated signatures must be accepted as valid. In the case of a LH-Sig scheme we distinguish between (a) "initial" and (b) "derived" signatures. Let $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\mathsf{Setup}(1^\lambda))$ be any key pair and τ be any tag in $\mathcal{T}$. Then,

(a) for every message $\mathbf{M} \in \mathbb{G}^n$ and $\Sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, \tau, \mathbf{M})$, the scheme satisfies $\mathsf{Verify}(\mathsf{pk}, \tau, \mathbf{M}, \Sigma) = 1$, and
(b) for any list $(\omega_j, \mathbf{M}_j, \Sigma_j)_{j=1}^d$ such that $\mathsf{Verify}(\mathsf{pk}, \tau, \mathbf{M}_j, \Sigma_j) = 1$ for each $j \in [d]$, if $\Sigma \leftarrow \mathsf{DeriveSign}(\mathsf{pk}, \tau, (\omega_j, \Sigma_j)_{j=1}^d)$, then $\mathsf{Verify}(\mathsf{pk}, \tau, \sum_{j=1}^d \omega_j \cdot \mathbf{M}_j, \Sigma) = 1$.

Security. We recall the unforgeability notion of [LPJY13], using our notations.

Definition 4 (Unforgeability of LH-Sig). *For a PPT adversary A and a LH-Sig scheme $\Sigma = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{DeriveSign}, \mathsf{Verify})$ with message space $\mathbb{G}^n$ and tag space $\mathcal{T}$, we define the experiment $\mathbf{Exp}^{\mathrm{unf}}_{\Sigma,A}(1^\lambda)$ as shown in Fig. 1.*

The oracles $\mathcal{O}\mathsf{Sign}$, $\mathcal{O}\mathsf{DeriveSign}$ and $\mathcal{O}\mathsf{Reveal}$ can be called in any order and any number of times. For a tag $\tau \in \mathcal{T}$, $\mathcal{S}_\tau$ denotes the set of messages $\mathbf{M} \in \mathbb{G}^n$ such that the tuple $(\tau, \mathbf{M})$ is in

the set $\mathcal{S}$ maintained by the challenger. The challenger also maintains a table $\mathcal{H}$ that contains tuples of the form $(h, (\tau, \mathbf{M}, \Sigma))$, where h is a handle, $\tau \in \mathcal{T}$, $\mathbf{M} \in \mathbb{G}^n$ and Σ is a signature on $\mathbf{M}$ under τ. We define a lookup operation $\mathsf{Lookup}_{\mathcal{H}}$ that, on input a set of handles $\{h_1, \ldots, h_d\}$, retrieves the tuples $\{(h_j, (\tau_j, \mathbf{M}_j, \Sigma_j))\}_{j=1}^d$ from $\mathcal{H}$ and returns $\{(\tau_j, \mathbf{M}_j, \Sigma_j)\}_{j=1}^d$. If there exists a $j \in [d]$ such that there does not exist a tuple of the form $(h_j, (\cdot, \cdot, \cdot))$ in $\mathcal{H}$, then the $\mathsf{Lookup}_{\mathcal{H}}$ algorithm returns an error message that causes the oracle to abort immediately with return value $\perp$.

We say that a LH-Sig scheme is unforgeable if for any PPT adversary A, there exists a negligible function negl *such that*

$$\mathbf{Adv}^{\mathrm{unf}}_{\Sigma,A}(\lambda) = \Pr\left[\mathbf{Exp}^{\mathrm{unf}}_{\Sigma,A}(1^\lambda) = 1\right] \leq \mathsf{negl}(\lambda).$$

Since $\mathbb{G}^n$ forms a vector space over $\mathbb{Z}_p$, it is consistent to write $\mathsf{span}(\mathcal{A})$ for the subspace spanned by a subset $\mathcal{A}$ of $\mathbb{G}^n$. As usual, we set $\mathsf{span}(\emptyset) = \{\mathbf{0}^{(n)}\}$.

$\underline{\mathsf{Initialize}(1^\lambda)}$:
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{Setup}(1^\lambda))$
$\mathcal{H} \leftarrow \emptyset$; $\mathcal{S} \leftarrow \emptyset$
Return pk

$\underline{\mathcal{O}\mathsf{Sign}(\tau, \mathbf{M})}$:
$\Sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, \tau, \mathbf{M})$
Pick new handle h
$\mathcal{H} \leftarrow \mathcal{H} \cup \{(h, (\tau, \mathbf{M}, \Sigma))\}$
Return h

$\underline{\mathcal{O}\mathsf{Reveal}(h)}$:
$(\tau, \mathbf{M}, \Sigma) \leftarrow \mathsf{Lookup}_{\mathcal{H}}(h)$
$\mathcal{S} \leftarrow \mathcal{S} \cup \{(\tau, \mathbf{M})\}$
Return Σ

$\underline{\mathcal{O}\mathsf{DeriveSign}((h_j, \omega_j)_{j=1}^d)}$:
$\{(\tau_j, \mathbf{M}_j, \Sigma_j)\}_{j=1}^d \leftarrow \mathsf{Lookup}_{\mathcal{H}}(\{h_j\}_{j=1}^d)$
If $\exists j \in [2; d]$ s.t. $\tau_j \neq \tau_1$, return $\perp$
$\mathbf{M} \leftarrow \sum_{j=1}^d \omega_j \cdot \mathbf{M}_j$
$\Sigma \leftarrow \mathsf{DeriveSign}(\mathsf{pk}, \tau_1, (\omega_j, \Sigma_j)_{j=1}^d)$
Pick new handle h
$\mathcal{H} \leftarrow \mathcal{H} \cup \{(h, (\tau_1, \mathbf{M}, \Sigma))\}$
Return h

$\underline{\mathsf{Finalize}(\tau, \mathbf{M}, \Sigma)}$:
If $\mathsf{Verify}(\mathsf{pk}, \tau, \mathbf{M}, \Sigma) = 0$, return 0
If $\mathbf{M} \in \mathsf{span}(\mathcal{S}_\tau)$, return 0
Return 1

Fig. 1. Security game $\mathbf{Exp}^{\mathrm{unf}}_{\Sigma,A}(1^\lambda)$ for unforgeability

As in [LPJY13], we will also consider a weaker notion of unforgeability. A *one-time* linearly-homomorphic signature (OT-LH-Sig) is a LH-Sig scheme, where the tag space is a singleton $\mathcal{T} = \{\varepsilon\}$. Consequently, we can drop the tags τ given as argument to the algorithms Sign, DeriveSign and Verify.

Privacy. Given signatures on messages $\mathbf{M}_1, \ldots, \mathbf{M}_d \in \mathbb{G}^n$, it may be desirable that derived signatures on a message $\mathbf{M} \in \mathsf{span}(\{\mathbf{M}_1, \ldots, \mathbf{M}_d\})$ do not leak information about $\mathbf{M}_1, \ldots, \mathbf{M}_d$ beyond what is revealed by $\mathbf{M}$. A strong definition that formalizes this property is given by Ahn *et al.* [ABC+11] under the name *context hiding*. We first state our definition and explain the differences to [ABC+11] afterwards.

Definition 5 (Context Hiding). *A LH-Sig scheme* $\Sigma = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{DeriveSign}, \mathsf{Verify})$ *with message space $\mathbb{G}^n$ and tag space $\mathcal{T}$ is called perfectly (resp. statistically, computationally) context-hiding if*

- *for any key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{Setup}(1^\lambda))$ and any tag $\tau \in \mathcal{T}$,*
- *for any tuple of messages $\mathcal{M} = (\mathbf{M}_j)_{j=1}^d \in (\mathbb{G}^n)^d$ and any tuple of coefficients $\boldsymbol{\omega} = (\omega_j)_{j=1}^d \in \mathbb{Z}_p^d$, and*
- *for any tuple of signatures $\mathcal{S} = (\Sigma_j)_{j=1}^d$, where Σ_j is a signature returned by* Sign *on input $(\mathsf{sk}, \tau, \mathbf{M}_j)$ with a nonzero probability,*

the following distribution ensembles are perfectly (resp. statistically, computationally) indistinguishable:

$$\mathcal{D}_0 = \left\{ \left(\mathsf{sk}, (\Sigma_j)_{j=1}^d, \mathsf{Sign}(\mathsf{sk}, \tau, \sum_{j=1}^d \omega_j \cdot \mathbf{M}_j) \right) \right\}_{\mathsf{sk},\tau,\mathcal{M},\boldsymbol{\omega},\mathcal{S}}$$
$$\mathcal{D}_1 = \left\{ \left(\mathsf{sk}, (\Sigma_j)_{j=1}^d, \mathsf{DeriveSign}\big(\mathsf{pk}, \tau, (\omega_j, \Sigma_j)_{j=1}^d\big) \right) \right\}_{\mathsf{sk},\tau,\mathcal{M},\boldsymbol{\omega},\mathcal{S}}$$

The definition states that a signature on a message $\mathbf{M}$ derived from a collection of signatures $(\Sigma_j)_{j=1}^d$ on messages $(\mathbf{M}_j)_{j=1}^d$ is indistinguishable from a fresh signature on $\mathbf{M}$, even if the original signatures are known. Consequently, the derived signature is independent of $(\Sigma_j)_{j=1}^d$ and cannot reveal any information about $(\mathbf{M}_j)_{j=1}^d$ beyond what is revealed by $\mathbf{M}$.

Our definition is stronger than that of [ABC+11] in two respects. First, we do not ask for the signatures $\Sigma_1, \ldots, \Sigma_d$ to be distributed according to the output distribution of Sign. Instead, we require indistinguishability for any (fixed) choice. The second aspect affects only the computational version. For completeness, we recall the original definition by Ahn *et al.* in Appendix A.1. Here, we give the adversary access to the secret key (instead of only the public one). This means context hiding holds even with respect to the signer. As a consequence, the oracles $\mathcal{O}$Sign, $\mathcal{O}$DeriveSign and $\mathcal{O}$Reveal can be simulated by the adversary itself and thus removed from the security game. This greatly simplifies the computational version of the definition.

Remark 6. As observed by Ahn *et al.*, the context hiding property can be used to simplify the security experiment for unforgeability. For a LH-Sig scheme Σ and a PPT adversary A, we define the experiment $\mathbf{Exp}^{\mathrm{unf'}}_{\Sigma,A}(1^\lambda)$ as shown in Figure 2. If a LH-Sig scheme is context hiding and unforgeable with respect to $\mathbf{Exp}^{\mathrm{unf'}}_{\Sigma,A}(1^\lambda)$, then it is also unforgeable with respect to $\mathbf{Exp}^{\mathrm{unf}}_{\Sigma,A}(1^\lambda)$. For a proof, see Lemma A.4 of [ABC+11]. We exploit this observation in Section 6 to simplify the definition of traceability within our model of a linearly-homomorphic group signature (LH-GSig) scheme.

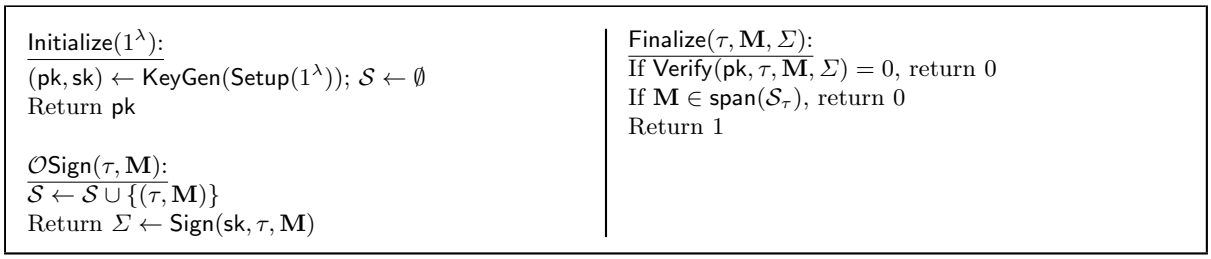

Fig. 2. Simplified security game $\mathbf{Exp}^{\mathrm{unf'}}_{\Sigma,A}(1^\lambda)$ for unforgeability

3 Codes with the Fully Identifiable Parent Property

Let $\mathcal{Q}$ be an alphabet. A set $\mathcal{C} = \{\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(n)}\} \subseteq \mathcal{Q}^\ell$ is called a *code* of *size* n and *length* ℓ. Each $\mathbf{w}^{(i)} \in \mathcal{C}$ is called a *codeword*. Furthermore, we say a subset $\mathcal{X} \subseteq \mathcal{C}$ is a *coalition*. For $k \in [\ell]$, let $\mathcal{Q}_k(\mathcal{X})$ denote the set of letters $a \in \mathcal{Q}$ for which there exists a codeword $\mathbf{w}^{(i)} = (w_1^{(i)}, \ldots, w_\ell^{(i)}) \in \mathcal{X}$ such that $w_k^{(i)} = a$.

We recall some standard terminology that goes back to Chor *et al.* [CFN94]. A word $\mathbf{u} = (u_k)_{k=1}^\ell \in \mathcal{Q}^\ell$ is called a *descendant* of a coalition $\mathcal{X}$ if for any $k \in [\ell]$, $u_k \in \mathcal{Q}_k(\mathcal{X})$. In this case, the elements of $\mathcal{X}$ are called *parent codewords* of $\mathbf{u}$. We denote by $\mathsf{Desc}(\mathcal{X})$ the set of all

descendants of $\mathcal{X}$. Furthermore, for a positive integer c, we write $\mathsf{Desc}_c(\mathcal{C})$ for the set of all descendants of coalitions that contain at most c codewords, i.e.

$$\mathsf{Desc}_c(\mathcal{C}) = \bigcup_{\mathcal{X}\in\mathcal{P}_c(\mathcal{C})} \mathsf{Desc}(\mathcal{X}).$$

For a positive integer c and a word $\mathbf{u} \in \mathsf{Desc}_c(\mathcal{C})$, we write $\mathsf{Par}_c(\mathbf{u})$ for the set of all coalitions $\mathcal{X}$ of size at most c such that $\mathbf{u} \in \mathsf{Desc}(\mathcal{X})$, i.e. $\mathsf{Par}_c(\mathbf{u}) = \{\mathcal{X} \in \mathcal{P}_c(\mathcal{C}) : \mathbf{u} \in \mathsf{Desc}(\mathcal{X})\}$. Then one defines codes with the *identifiable parent property (IPP)* as follows.

Definition 7 (IPP Code). *A code $\mathcal{C}$ is called c-IPP if for any $\mathbf{u} \in \mathsf{Desc}_c(\mathcal{C})$, the intersection of all $\mathcal{X} \in \mathsf{Par}_c(\mathbf{u})$ is nonempty.*

Intuitively, a code $\mathcal{C}$ is IPP if for every descendant $\mathbf{u} \in \mathsf{Desc}_c(\mathcal{C})$ at least one parent codeword can be identified with certainty. In this paper, we pursue the stronger goal to identify not only one parent but a full coalition $\mathcal{X} \subseteq \mathcal{C}$ such that $\mathbf{u} \in \mathsf{Desc}(\mathcal{X})$. In order not to falsely include codewords into $\mathcal{X}$, we must require that there is a smallest coalition with this property. Then we define the notion of a *fully IPP (FIPP)* code as follows.

Definition 8 (FIPP Code). *A code $\mathcal{C}$ is called c-FIPP if for any $\mathbf{u} \in \mathsf{Desc}_c(\mathcal{C})$, there exists a smallest element $\mathcal{X}_0$ in $\mathsf{Par}_c(\mathbf{u})$ with respect to the usual inclusion.*

While IPP codes for the standard definition of descendants have been the subject of intensive studies, little is known about other derivation models. Here, we introduce a novel definition of descendancy.

Definition 9 (Descendant). *Let $\mathcal{Q} = \{a_1, \ldots, a_n, \perp, \top\}$, where $\perp$ and $\top$ are two distinguished letters. We say that a word $\mathbf{u} = (u_k)_{k=1}^{\ell} \in \mathcal{Q}^{\ell}$ is a descendant of a set $\mathcal{X}$ if there exists a nonempty subset $\mathcal{X}_0 \subseteq \mathcal{X}$ that satisfies the following condition for each $k \in [\ell]$.*

1. *If $\mathcal{Q}_k(\mathcal{X}_0) = \{a\}$ for some $a \in \mathcal{Q}$, then $u_k = a$.*
2. *Else, if $\mathcal{Q}_k(\mathcal{X}_0) = \{\top, a\}$ for some $a \in \mathcal{Q} \setminus \{\top\}$, then $u_k = a$.*
3. *Else, $u_k = \perp$.*

In the remainder of this article, we consider IPP and FIPP codes always with respect to Definition 9. We postpone the motivation of this new derivation model to Section 4 where we present a natural application.

Construction. We present an extremely simple construction of a c-FIPP code $\mathcal{C}$ of size n and length ℓ over the alphabet $\mathcal{Q} = \{a_1, \ldots, a_n, \top, \perp\}$.

In the construction, ℓ appears as the product of two positive integers J and K. For convenience, we use tuples $(k, j) \in [K] \times [J]$ to index the letters of codewords. However, they can simply be thought of as a single vector of length ℓ, consisting of K blocks with J coordinates each. For $i \in [n]$, let $\mathcal{S}_i^{(J)} \subseteq \mathcal{Q}^J$ be the set that contains all sequences $(s_1, \ldots, s_J)$ with the property that there exists a $j \in [J]$ such that $s_j = a_i$ and $s_{j'} = \top$ for all $j' \in [J] \setminus \{j\}$. Furthermore, we define $\mathcal{S}_i^{(K,J)} = \bigtimes_{k=1}^{K} \mathcal{S}_i^{(J)}$.

Theorem 10. *Let n, c and $J > c$ be positive integers. If $\mathbf{w}^{(i)} \xleftarrow{\$} \mathcal{S}_i^{(K,J)}$ for each $i \in [n]$, then the code $\mathcal{C} = \{\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(n)}\}$ is c-FIPP with probability at least $1 - \mathsf{negl}(K)$ for some negligible function negl.*

Proof. Let $\mathcal{Q}' = \mathcal{Q} \setminus \{\top, \perp\}$. For $\mathbf{u} = (u_{k,j})_{k\in[K],j\in[J]} \in \mathsf{Desc}_c(\mathcal{C})$, we set $\mathcal{U} = \{u_{k,j} : k \in [K], j \in [J]\} \cap \mathcal{Q}'$ and $\mathcal{X}_0 = \{\mathbf{w}^{(i)} : a_i \in \mathcal{U}\}$. First, observe that $\mathcal{X}_0$ is smaller than or equal to every element of $\mathsf{Par}_c(\mathbf{u})$. This can be seen as follows. Every letter in $\mathcal{U}$ occurs only in one codeword,

and $\mathcal{X}_0$ contains exactly this collection of codewords. Furthermore, descendants can only contain letters of $\mathcal{Q}'$ if they already appeared in at least one parent codeword. Hence, $\mathcal{X}_0$ is a subset of each $\mathcal{X} \in \mathsf{Par}_c(\mathbf{u})$. This property implies in particular that $|\mathcal{X}_0| \leq c$.

It remains to show that $\mathbf{u} \in \mathsf{Desc}(\mathcal{X}_0)$. Since $\mathbf{u} \in \mathsf{Desc}_c(\mathcal{C})$, there exists a set $\mathcal{X}' \subseteq \mathcal{C}$ in $\mathsf{Par}_c(\mathbf{u})$. Let $\mathcal{X}'_0$ denote a possible choice for the specific subset of $\mathcal{X}'$ whose existence is required in our definition of descendants (Definition 9). Note that the elements in $\mathcal{X}' \setminus \mathcal{X}'_0$ are irrelevant for the descendancy of $\mathbf{u}$ which implies that $\mathbf{u} \in \mathsf{Desc}(\mathcal{X}'_0)$. Then it suffices to show that $\mathcal{X}_0 = \mathcal{X}'_0$ with high probability. (In this case it follows in particular that the above choice of $\mathcal{X}'_0$ is unique.) First, since $\mathbf{u} \in \mathsf{Desc}(\mathcal{X}'_0)$, we have that $\mathcal{X}'_0 \in \mathsf{Par}_c(\mathbf{u})$ and, thus, $\mathcal{X}_0 \subseteq \mathcal{X}'_0$. For the other inclusion, we define $\mathcal{I} = \{i \in [n] : \mathbf{w}^{(i)} \in \mathcal{X}'_0\}$. Furthermore, for $i \in [n]$, we parse $\mathbf{w}^{(i)} = (w^{(i)}_{k,j})_{k\in[K],j\in[J]}$ and let $\mathbf{d}^{(i)} = (d^{(i)}_1, \ldots, d^{(i)}_K) \in [J]^K$ such that

$$\left(w^{(i)}_{1,d^{(i)}_1}, \ldots, w^{(i)}_{K,d^{(i)}_K}\right) = \left(a_i, \ldots, a_i\right),$$

i.e. $\mathbf{d}^{(i)}$ contains the positions of the letter a_i in each of the K blocks of $\mathbf{w}^{(i)}$. Let E denote the event that there exists an $i_0 \in \mathcal{I}$ with the property that for all $k \in [K]$, there exists an $i_k \in \mathcal{I}\backslash\{i_0\}$ such that $d^{(i_k)}_k = d^{(i_0)}_k$. Note that E corresponds exactly to the event that $\mathcal{X}_0 \subsetneq \mathcal{X}'_0$. For a fixed choice of $i_0 \in \mathcal{I}$, the probability that for all $k \in [K]$, there exists such an $i_k \in \mathcal{I} \setminus \{i_0\}$ with the property that $d^{(i_k)}_k = d^{(i_0)}_k$, is bounded by $(c/J)^K$. Thus, $\Pr[E] \leq c \cdot (c/J)^K$.

Finally, applying the union bound over all possible coalitions $\mathcal{X}'_0 \subseteq \mathcal{C}$ of size at most c implies the result for all $\mathbf{u} \in \mathsf{Desc}_c(\mathcal{C})$. Hence, we conclude that $\mathcal{C}$ is c-FIPP with probability at least $1 - c \cdot n^c \cdot (c/J)^K = 1 - \mathsf{negl}(K)$. □

Remark 11 (Efficiency). For a practical usability of a FIPP code, one needs efficient generation and tracing algorithms. The code generation in our construction is trivial as one only samples $\mathbf{w}^{(i)} \xleftarrow{\$} \mathcal{S}_i^{(K,J)}$ for each $i \in [n]$. Furthermore, we observe that the proof of Theorem 10 does not only show the existence of a smallest set $\mathcal{X}_0$ but also shows how to construct this set efficiently. Indeed, on input a descendant $\mathbf{u} = (u_{k,j})_{k\in[K],j\in[J]} \in \mathsf{Desc}_c(\mathcal{C})$, the tracing algorithm simply returns the set $\{u_{k,j} : k \in [K], j \in [J]\} \setminus \{\top, \bot\}$.

Another aspect of efficiency considers the length of the code. Let ε denote the acceptable error probability. If one chooses, say, $J = 2c$, then the proof of Theorem 10 yields that $K = \mathcal{O}(c \cdot \log(n/\varepsilon))$, which in turn implies that $\ell = \mathcal{O}(c^2 \cdot \log(n/\varepsilon))$. Furthermore, our alphabet has size $\mathcal{O}(n)$. Thus, our codewords have representations on $\mathcal{O}(c^2 \cdot \log(n/\varepsilon) \cdot \log n)$ bits. It is interesting to compare this bound with the "naive" FIPP code $\mathcal{C} = \{\mathbf{w}^{(i)} = (\delta_{i,j})_{j=1}^n\}_{i=1}^n$, where $\delta_{i,j} = \bot$ if $i = j$, and $\delta_{i,j} = \top$ otherwise. This code uses the binary alphabet $\mathcal{Q} = \{\top, \bot\}$. Codewords in the naive solution thus require $\mathcal{O}(n)$ bits which grows asymptotically faster (in n) than our bound $\mathcal{O}(c^2 \cdot \log(n/\varepsilon) \cdot \log n)$, with reasonable trade-off on c and ε.

4 An Efficient Tracing Algorithm for Linear Subspaces

Let $\mathbb{U}$ be a vector space over a field $\mathbb{K}$ and let $\mathcal{V} = \{\mathbf{V}_1, \ldots, \mathbf{V}_n\} \subset \mathbb{U}$. Given a vector $\boldsymbol{\omega} = (\omega_i)_{i=1}^n \in \mathbb{K}^n$, we write $N_{\boldsymbol{\omega}} = |\{i \in [n] : \omega_i \neq 0\}|$. Then for a positive integer c, we denote $\mathsf{span}_c(\mathcal{V}) = \bigcup_{\mathcal{X}\in\mathcal{P}_c(\mathcal{V})} \mathsf{span}(\mathcal{X})$. Also, we call $\mathcal{V}$ *c-linearly independent* if the following implication is satisfied for every $\boldsymbol{\omega} = (\omega_i)_{i=1}^n \in \mathbb{K}^n$:

$$\sum_{i=1}^{n} \omega_i \cdot \mathbf{V}_i = 0 \implies N_{\boldsymbol{\omega}} = 0 \vee N_{\boldsymbol{\omega}} > c$$

Intuitively, c-linear independence states that for each vector $\mathbf{U}$ in $\mathsf{span}_c(\mathcal{V})$, there exists a unique choice of coefficients $\omega_1, \ldots, \omega_n$ such that $\sum_{i=1}^n \omega_i \cdot \mathbf{V}_i = \mathbf{U}$ and there are at most c nonzero coefficients. However, the definition does not exclude that there may exist coefficients $\omega'_1, \ldots, \omega'_n$ that also satisfy $\sum_{i=1}^n \omega'_i \cdot \mathbf{V}_i = \mathbf{U}$, but with more than c coefficients being nonzero.

Definition 12 (Linear-Subspace Tracing (LST) Scheme). *Let $\mathbb{U}$ be a vector space and c a positive integer. A LST scheme (for linear subspaces of $\mathbb{U}$) is a tuple of two polynomial-time algorithms defined as follows.*

$\mathsf{Gen}(1^\lambda, 1^n)$*: On input the security parameter λ and a positive integer n, this algorithm returns a tracing key tk and a c-linearly independent set $\mathcal{V} \subseteq \mathbb{U}$ of size n.*

$\mathsf{Trace}(\mathsf{tk}, \mathbf{U})$*: On input a tracing key tk and a vector $\mathbf{U} \in \mathsf{span}_c(\mathcal{V})$, this algorithm returns a set $\mathcal{I} \subseteq [n]$.*

For a PPT adversary A and a LST scheme $\mathsf{T} = (\mathsf{Gen}, \mathsf{Trace})$, the experiment $\mathbf{Exp}^{c\text{-cor}}_{\mathsf{T},A}(1^\lambda, 1^n)$ is defined as shown in Fig. 3. We say that a LST scheme T is c-correct if for any PPT adversary A and any polynomial $n = n(\lambda)$, there exists a negligible function negl such that

$$\mathbf{Adv}^{c\text{-cor}}_{\mathsf{T},A}(\lambda, n) = \Pr\left[\mathbf{Exp}^{c\text{-cor}}_{\mathsf{T},A}(1^\lambda, 1^n) = 1\right] \leq \mathsf{negl}(\lambda).$$

$(\mathsf{tk}, \mathcal{V} = \{\mathbf{V}_1, \ldots, \mathbf{V}_n\}) \leftarrow \mathsf{Gen}(1^\lambda, 1^n)$
$\mathcal{X} \xleftarrow{\$} \mathcal{P}_c(\mathcal{V})$; $\mathbf{U} \leftarrow A(\mathsf{tk}, \mathcal{X})$; $\mathcal{I} \leftarrow \mathsf{Trace}(\mathsf{tk}, \mathbf{U})$
If $\mathbf{U} \notin \mathsf{span}(\mathcal{X})$, return 0
If $\{\mathbf{V}_i : i \in \mathcal{I}\} \neq \bigcap_{\mathcal{X}' \subseteq \mathcal{X}, \mathbf{U} \in \mathsf{span}(\mathcal{X}')} \mathcal{X}'$, return 1
Return 0

Fig. 3. Security game $\mathbf{Exp}^{c\text{-cor}}_{\mathsf{T},A}(1^\lambda, 1^n)$ for c-correctness

As mentioned above, c-linear independence of the set $\mathcal{V}$ ensures that vectors in $\mathsf{span}_c(\mathcal{V})$ have a unique representation as a linear combination with at most c coefficients being nonzero. The correctness condition states that the tracing algorithm returns (the indices of) these nonzero coefficients with overwhelming probability.

Construction. Let $\mathbb{G}$ be a group of prime order p in which the DL problem is hard. In the following, we construct a LST scheme $\mathsf{T} = (\mathsf{Gen}, \mathsf{Trace})$ for subspaces of $\mathbb{G}^\ell$, where $\ell = \Theta(c^2 \cdot \log(n/\varepsilon))$ and ε is the maximum acceptable probability that the tracing will fail. We view the construction as an implementation of the FIPP code presented in Section 3.

Intuitively, the Gen algorithm first chooses a random alphabet $\mathcal{Q}$, along with a tracing key tk that serves as a description of $\mathcal{Q}$. Then it generates a FIPP code $\mathcal{C} = \{\mathbf{w}^{(i)} : i \in [n]\}$ of length ℓ' and chooses a (random) representative $\mathbf{V}^{(i)} \in \mathbb{G}^\ell$ for each codeword $\mathbf{w}^{(i)}$ and $\ell = 2 \cdot \ell'$. (We will detail below what "represent" means exactly in our context.) Finally, the Gen algorithm returns tk and $\mathcal{V} = \{\mathbf{V}^{(i)} : i \in [n]\}$. We then show that if a vector $\mathbf{U}$ is in the span of some set $\{\mathbf{V}^{(i_1)}, \ldots, \mathbf{V}^{(i_{c'})}\} \in \mathcal{P}_c(\mathcal{V})$, then the word $\mathbf{u} \in \mathcal{Q}^{\ell'}$ which is represented by $\mathbf{U}$ is in $\mathsf{Desc}(\{\mathbf{w}^{(i_1)}, \ldots, \mathbf{w}^{(i_{c'})}\})$ with overwhelming probability. Thus, it is enough to recover $\mathbf{u}$ from $\mathbf{U}$ and to utilize the tracing mechanism of the FIPP code explained in Remark 11.

An overview of the implementation of our LST scheme $\mathsf{T} = (\mathsf{Gen}, \mathsf{Trace})$ is depicted in Fig. 4. We explain the details of the construction below. We place special emphasis on the connections to the FIPP code from Section 3.

Generation. At first, the Gen algorithm samples a vector $\mathbf{s} = (s_i)_{i=1}^n \xleftarrow{\$} (\mathbb{Z}_p^*)^n$. This vector specifies the alphabet $\mathcal{Q} = \mathcal{Q}_\mathbf{s}$ and serves as the tracing key tk. Formally, $\mathcal{Q}$ is a partition of $\mathbb{G} \times \mathbb{G}$ and the letters are the disjoint subsets of $\mathcal{Q}$, which can be seen as classes of equivalence, defined as follows: for $i \in [n]$, we set $a_i = \{(G, s_i \cdot G) : G \in \mathbb{G}^*\}$, $\top = \{(0,0)\}$, and $\bot = (\mathbb{G} \times \mathbb{G}) \setminus (\top \cup \bigcup_{i=1}^n a_i)$.

In the second step, the generation algorithm generates the codewords. According to Remark 11, one samples $\mathbf{w}^{(i)} \xleftarrow{\$} \mathcal{S}_i^{(K,J)}$ for each $i \in [n]$, where $\mathbf{w}^{(i)} \in \mathcal{Q}^{\ell'}$ and $\ell' = K \cdot J$

$\underline{\mathsf{Gen}(1^\lambda, 1^n)}$:
$\mathbf{s} = (s_i)_{i=1}^n \xleftarrow{\$} (\mathbb{Z}_p^*)^n$
For $i \in [n]$:
  $\mathbf{d}_i = (d_{i,k})_{k=1}^K \xleftarrow{\$} [J]^K$; $G_i \xleftarrow{\$} \mathbb{G}^*$
  $\mathbf{V}^{(i)} \leftarrow (V_{k,j}^{(i)})_{k\in[K],j\in[J]}$ where
  $V_{k,j}^{(i)} = \begin{cases} (G_i, s_i \cdot G_i) & \text{if } j = d_{i,k} \\ (0,0) & \text{otherwise} \end{cases}$
Return $(\mathsf{tk} \leftarrow \mathbf{s}, \mathcal{V} \leftarrow \{\mathbf{V}^{(i)}\}_{i=1}^n)$

$\underline{\mathsf{Trace}(\mathsf{tk} = (s_i)_{i=1}^n, \mathbf{U})}$:
Parse $\mathbf{U} = (U_{k,j})_{k\in[K],j\in[J]}$
$\mathcal{I} \leftarrow \emptyset$
For $k \in [K]$, $j \in [J]$ and $i \in [n]$:
  Parse $U_{k,j} = (G, H)$
  If $s_i \cdot G = H$, then $\mathcal{I} \leftarrow \mathcal{I} \cup \{i\}$
Return $\mathcal{I}$

Fig. 4. Our LST scheme

for suitable polynomials $K(\lambda)$ and $J(\lambda)$. Strictly speaking, these codewords are a sequence of classes of $\mathcal{Q}$. However, since the size of these classes can be exponential, we choose a representative $(G_i, s_i \cdot G_i) \xleftarrow{\$} a_i$ for each letter $a_i \in \mathcal{Q}$. Furthermore, since $\top$ is a singleton, there is only one possible choice for a representative. Then we construct $\mathbf{V}^{(i)}$ by replacing each occurrence of the letter a_i in $\mathbf{w}^{(i)}$ with the selected representative $(G_i, s_i \cdot G_i)$ and each letter $\top$ with the tuple $(0, 0)$.

More straightforward, the construction of the vectors $\mathbf{V}^{(i)}$ can be described as follows. For each $i \in [n]$, one samples a random tuple $\mathbf{d}_i = (d_{i,1}, \ldots, d_{i,K}) \xleftarrow{\$} [J]^K$ and a group generator $G_i \xleftarrow{\$} \mathbb{G}^*$, and sets

$$\mathbf{V}^{(i)} = \left(V_{k,j}^{(i)} = \begin{cases} (G_i, s_i \cdot G_i) & \text{if } j = d_{i,k} \\ (0,0) & \text{otherwise} \end{cases} \right)_{k\in[K],j\in[J]}$$

We emphasize that the random choice of both the vector $\mathbf{s}$ and the representatives of the letters is crucial to achieve correctness against arbitrary PPT adversaries.

Tracing. Let $\mathbf{U} = (U_{k,j})_{k\in[K],j\in[J]} \in \mathcal{P}_c(\mathcal{V})$. The crucial part is to recover the word $\mathbf{u} \in \mathcal{Q}^\ell$ which is represented by $\mathbf{U}$. That is, for each coordinate $(k, j) \in [K] \times [J]$, we need to identify the letter $a \in \mathcal{Q}$ such that $U_{k,j} \in a$.

Since the DL problem is assumed to be hard in $\mathbb{G}$, the scalar s cannot be recovered from a tuple $(G, s \cdot G) \in \mathbb{G} \times \mathbb{G}$. However, since we deal only with a polynomial number n, and thus with a polynomial-size alphabet $\mathcal{Q}_\mathbf{s}$, we can test for each $k \in [K]$, $j \in [J]$ and $i \in [n]$ if $U_{k,j} \in a_i$. Once we have recovered $\mathbf{u}$, we simply use the tracing mechanism explained in Remark 11.

Correctness. To prove correctness of the LST scheme T, one needs to show that each vector $\mathbf{U}$ in the span of some set $\mathcal{X} \in \mathcal{P}_c(\mathcal{V})$ represents a descendant of the set containing all codewords that are represented by an element of $\mathcal{X}$.

To state this result more formally, we introduce the following notations. First, recall that a vector $\mathbf{s} \in (\mathbb{Z}_p^*)^n$ defines an equivalence relation $\mathcal{Q}_\mathbf{s}$ on $\mathbb{G} \times \mathbb{G}$. So it is consistent to write $[(G, H)]_\mathbf{s}$ (or simply $[(G, H)]$ if $\mathbf{s}$ is fixed) for the letter $a \in \mathcal{Q}$ which contains $(G, H) \in \mathbb{G} \times \mathbb{G}$. Similarly, for a vector $\mathbf{V} = (V_i)_{i=1}^\ell \in (\mathbb{G} \times \mathbb{G})^\ell$, $[V_1] \cdots [V_n]$ denotes the word $\mathbf{w} \in \mathcal{Q}^\ell$ which is represented by $\mathbf{V}$. For convenience, we also refer to $\mathbf{w}$ by $[\mathbf{V}]$ and, for a set $\mathcal{V} = \{\mathbf{V}_1, \ldots, \mathbf{V}_n\} \subseteq (\mathbb{G} \times \mathbb{G})^\ell$, we denote the set $\{[\mathbf{V}_1], \ldots, [\mathbf{V}_n]\}$ by $[\mathcal{V}]$.

Theorem 13. *Let* $(\mathsf{tk}, \mathcal{V}) \leftarrow \mathsf{Gen}(1^\lambda, 1^n)$ *and let* A *be any algebraic PPT algorithm that on input* tk *and a set* $\mathcal{X} \in \mathcal{P}_c(\mathcal{V})$ *returns a nonzero vector* $\mathbf{U} \in \mathsf{span}(\mathcal{X})$*. Then it holds* $[\mathbf{U}] \in \mathsf{Desc}([\mathcal{X}])$ *with overwhelming probability under the* DL *assumption.*

For the proof, we need the following lemma.

Lemma 14. *Let $(\mathbb{G}, +)$ be a group of prime order p, $s \in \mathbb{Z}_p^*$ and $n > 2$. Given n tuples (G_i, s_i), for $G_i \xleftarrow{\$} \mathbb{G}^*$ and $s_i \xleftarrow{\$} \mathbb{Z}_p^*$, it is computationally hard under the* DL *assumption to output $(\omega_1, \ldots, \omega_n) \in \mathbb{Z}_p^n$ such that $\omega_i \neq 0$ for at least two $i \in [n]$ and*

$$s \cdot \sum_{i=1}^{n} \omega_i \cdot G_i = \sum_{i=1}^{n} \omega_i s_i \cdot G_i.$$

A proof of Lemma 14 is provided in Appendix B.1.

Proof (of Theorem 13). Let $\mathsf{tk} = \mathbf{s} = (s_i)_{i=1}^n$ and $\mathcal{V} = \{\mathbf{V}^{(1)}, \ldots, \mathbf{V}^{(n)}\}$. We assume that the entries of $\mathbf{s}$ are pairwise distinct, which is correct with overwhelming probability. Also, without loss of generality, we assume that $\mathcal{X} = \{\mathbf{V}^{(1)}, \ldots, \mathbf{V}^{(c)}\}$. Note that if $[\mathcal{V}]$ is c-FIPP, then $\mathcal{X}$ is c-linearly independent. (This can be seen in the proof of Theorem 10.) Since $\mathcal{X}$ contains c elements and is c-linearly independent, it follows that there exist unique coefficients $\omega_1, \ldots, \omega_c$ in $\mathbb{Z}_p$ such that $\mathbf{U} = \sum_{j=1}^{c} \omega_j \cdot \mathbf{V}^{(i)}$. These coefficients are provided by the (algebraic) adversary A. We set $\mathcal{X}_0 = \{\mathbf{V}^{(i)} \in \mathcal{X} : \omega_i \neq 0\}$.

Fix any $(k, j) \in [K] \times [J]$. We consider several cases that correspond to those in Definition 9.

1. $\mathcal{Q}_{k,j}([\mathcal{X}_0]) = \{a\}$ *for some* $a \in \mathcal{Q}$. We split this case into two subcases.
   (a) $\mathcal{Q}_{k,j}([\mathcal{X}_0]) = \{\top\}$. In this case, we have $V_{k,j}^{(i)} = (0, 0)$ for all $i \in [c]$ satisfying $\omega_i \neq 0$, which implies that also $U_{k,j} = \sum_{i=1}^{c} \omega_i \cdot V_{k,j}^{(i)} = (0, 0)$. Thus, $[U_{k,j}] = \top$.
   (b) $\mathcal{Q}_{k,j}([\mathcal{X}_0]) = \{a_i\}$ *for some* $i \in [c]$. Since we assume that $s_1, \ldots, s_n$ are pairwise distinct, it follows that $\mathcal{X}_0 = \{\mathbf{V}^{(i)}\}$ and $V_{k,j}^{(i)} = (G_i, s_i \cdot G_i)$ for some $G_i \in \mathbb{G}^*$. Due to the DL assumption, the probability that $\omega_i \cdot G_i = 0$ is negligible. Thus, $[U_{k,j}] = [\omega_i \cdot (G_i, s_i \cdot G_i)] = a_i$ with overwhelming probability.

   The case $\mathcal{Q}_{k,j}([\mathcal{X}_0]) = \bot$ cannot occur since elements of $\bot$ do not appear as components in the vectors in $\mathcal{V}$.
2. $\mathcal{Q}_{k,j}([\mathcal{X}_0]) = \{\top, a_i\}$ *for some* $i \in [c]$. This case can be seen as the combination of the cases 1. (a) and 1. (b). That is, we have $\omega_{i'} = 0$ or $V_{k,j}^{(i')} = (0, 0)$ for all $i' \in [c] \setminus \{i\}$, and $V_{k,j}^{(i)} = (G_i, s_i \cdot G_i)$ for some $G_i \in \mathbb{G}^*$. Again by the DL assumption, it follows that

$$U_{k,j} = \sum_{i'=1}^{c} \omega_{i'} \cdot V_{k,j}^{(i')} = (\omega_i \cdot G_i, \omega_i s_i \cdot G_i) \in a_i.$$

3. *None of the previous cases applies.* Let $\mathcal{J} = \{i \in [c] : V_{k,j}^{(i)} \neq (0, 0)\}$. If there exists at most one $i \in \mathcal{J}$ such that $\omega_i \neq 0$, then we can argue as in one of the previous cases. Otherwise, the argumentation is as follows. There exist generators $(G_i)_{i \in \mathcal{J}}$ such that $V_{k,j}^{(i)} = (G_i, s_i \cdot G_i)$ for each $i \in \mathcal{J}$. Fix some $s \in \mathbb{Z}_p$. By Lemma 14, it follows that a vector $\boldsymbol{\omega} = (\omega_i)_{i \in \mathcal{J}}$ that has at least two nonzero entries and satisfies

$$U_{k,j} = \sum_{i \in \mathcal{J}} \omega_i \cdot V_{k,j}^{(i)} = \left(\sum_{i \in \mathcal{J}} \omega_i \cdot G_i, s \cdot \sum_{i \in \mathcal{J}} \omega_i \cdot G_i \right)$$

   can be used to solve discrete logarithms in $\mathbb{G}$. Applying a union bound over all values of $s \in \{0, s_1, \ldots, s_n\}$ implies that $U_{k,j} \in \top \cup \bigcup_{i=1}^{n} a_i$ with negligible probability, i.e. $[U_{k,j}] = \bot$ with overwhelming probability.

Finally, the statement of the theorem follows by a union bound over all $(k, j) \in [K] \times [J]$. □

Remark 15 (Efficiency Improvement and Dynamic Generation). Let $i \in [n]$. Note that the vector $\mathbf{V}^{(i)} \in \mathbb{G}^{J \cdot K}$ in Fig. 4 is computed from $\mathbf{d}_i \in [J]^K$, $G_i \in \mathbb{G}$ and $s_i \in \mathbb{Z}_p$. Since these three

values require far less memory than $\mathbf{V}^{(i)}$, it would be desirable to use these three values as an efficient representation of $\mathbf{V}^{(i)}$. Clearly, $\mathbf{d}_i$ and G_i can be recovered from $\mathbf{V}^{(i)}$ and, thus, must not be hidden. Furthermore, Theorem 13 explicitly covers the case that the vector $\mathsf{tk} = (s_i)_{i=1}^n$ is known to the adversary. Thus, knowledge of the scalar s_i does not affect the correctness of the LST scheme. So $(\mathbf{d}_i, G_i, s_i)$ can indeed be used as an efficient representation of $\mathbf{V}^{(i)}$.

Moreover, we observe that the vectors $\mathbf{V}^{(1)}, \ldots, \mathbf{V}^{(n)}$ are generated independently, and the length of the code depends only on the security parameter but not on n (as long as it is polynomially in the security parameter). Therefore, the total number of vectors must not be fixed in advance, but new ones can be added dynamically.

5 Linear-Subspace Tracing and Anonymity

Equivalent vectors $\mathbf{V}_1, \mathbf{V}_2 \in \mathbb{G}^\ell$ (i.e. $[\mathbf{V}_1] = [\mathbf{V}_2]$) are publicly linkable, as the nonzero positions are publicly available. However, in some situations it may be desirable that users act anonymously, with unlinkable actions, except with respect to the authority holding the tracing key. We thus now explain how to make vectors anonymous.

Definition 16 (Anonymity). *For a LST scheme $\mathsf{T} = (\mathsf{Gen}, \mathsf{Trace})$ and a PPT adversary A, we define the experiment $\mathbf{Exp}^{\mathrm{ano}}_{\mathsf{T},A}(1^\lambda, 1^n)$ as shown in Fig. 5. The "Subspace or Full space" oracle $\mathcal{O}\mathsf{SoF}$ can be called any number of times.*

We say that a LST scheme is anonymous if for any PPT adversary A and any polynomial $n = n(\lambda)$, there exists a negligible function negl such that

$$\mathbf{Adv}^{\mathrm{ano}}_{\mathsf{T},A}(\lambda, n) = \left| \Pr\left[\mathbf{Exp}^{\mathrm{ano}}_{\mathsf{T},A}(1^\lambda, 1^n) = 1\right] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda).$$

$\underline{\mathsf{Initialize}}(1^\lambda, 1^n)$:	$\underline{\mathcal{O}\mathsf{SoF}}(\mathcal{X})$:
$b \xleftarrow{\$} \{0,1\}$; $(\mathsf{tk}, \mathcal{V}) \leftarrow \mathsf{Gen}(1^\lambda, 1^n)$	If $\mathcal{X} \not\subseteq \mathcal{V}$, return $\perp$
Return $\mathcal{V}$	$\mathbf{V}_0 \xleftarrow{\$} \mathsf{span}(\mathcal{X})$; $\mathbf{V}_1 \xleftarrow{\$} \mathsf{span}(\mathcal{V})$
	Return $\mathbf{V}_b$
$\underline{\mathsf{Finalize}}(b')$: Return $(b = b')$	

Fig. 5. Security game $\mathbf{Exp}^{\mathrm{ano}}_{\mathsf{T},A}(1^\lambda, 1^n)$ for anonymity

Let $\mathsf{T}' = (\mathsf{Gen}', \mathsf{Trace}')$ be the LST scheme for linear subspaces of $\mathbb{G}^{\ell'}$ defined in Fig. 4. Recall that $\ell' = 2KJ$ where K and J are specified by the underlying FIPP code. Let $(\mathsf{tk}', \mathcal{V}' = \{\mathbf{V}'_i\}_{i=1}^n) \leftarrow \mathsf{Gen}'(1^\lambda, 1^n)$. Note that even if the tracing algorithm cannot be run directly without the tracing key tk', the scheme is still not anonymous because the vectors in $\mathcal{V}'$ do not hide the embedded vectors $\mathbf{d}_1, \ldots, \mathbf{d}_n$. Hence, to break anonymity it is enough to compare the positions of the nonzero entries of the challenge vector with those of $\mathbf{V}_1, \ldots, \mathbf{V}_n$. Nevertheless, the scheme can easily be made anonymous.

Construction. As above, let $(\mathsf{tk}', \mathcal{V}' = \{\mathbf{V}'_i\}_{i=1}^n) \leftarrow \mathsf{Gen}'(1^\lambda, 1^n)$. The basic idea to obtain anonymous versions of $\mathbf{V}'_1, \ldots, \mathbf{V}'_n$ is to encrypt them component-wise using the ElGamal encryption scheme [ElG84]. That is, we sample a vector $\mathbf{x} = (x_k)_{k=1}^{\ell'} \xleftarrow{\$} \mathbb{Z}_p^{\ell'}$ where, for each $k \in [\ell']$, x_k serves as the secret key for the k-th component. The corresponding vector of public keys is $\mathbf{X} = \mathbf{x} \cdot G$ for some fixed generator $G \in \mathbb{G}^*$. Since each ElGamal ciphertext consists of two group elements, component-wise encryption of $\mathbf{V}'_i$ with $\mathbf{X}$ yields a vector $\mathbf{V}_i$ of length $2\ell'$. It is well-known that the ElGamal encryption scheme is partially homomorphic. In our context,

this means that, for all coefficients $\omega_1, \ldots, \omega_n$, $\sum_{i=1}^{n} \omega_i \cdot \mathbf{V}_i$ is a component-wise encryption of $\sum_{i=1}^{n} \omega_i \cdot \mathbf{V}'_i$. Thus, the tracing of a vector $\mathbf{U}$ can be done by first decrypting it and passing the resulting vector $\mathbf{U}'$ to the algorithm Trace'. The output $\mathcal{I}'$ of the tracing of $\mathbf{U}'$ is exactly the wanted output $\mathcal{I}$ for the tracing of $\mathbf{U}$.

Since the security of ElGamal is based on the DDH assumption, it is straightforward to exploit the random self-reducibility of the DDH problem here. If we reuse the same randomness in the encryption of all components of a vector $\mathbf{V}'_i$, $i \in [n]$, then one component of all ciphertexts is equal and, thus, must be included only once in the anonymous version $\mathbf{V}_i$ of $\mathbf{V}'_i$. This leads us to a more efficient anonymous transformation of T' where the vectors have only length $\ell = \ell' + 1$ instead of $2\ell'$.

The full scheme is depicted in Fig. 6. The correctness of the construction is a direct consequence of the correctness of T' and the fact that the ElGamal encryption scheme is partially homomorphic.

$\underline{\mathsf{Gen}(1^\lambda, 1^n)}$:
$(\mathsf{tk}', \mathcal{V}' = \{\mathbf{V}'_i\}_{i=1}^n) \leftarrow \mathsf{Gen}'(1^\lambda, 1^n)$
Choose any $G \in \mathbb{G}^*$
$\mathbf{x} \xleftarrow{\$} \mathbb{Z}_p^{\ell'}$; $\mathbf{X} \leftarrow \mathbf{x} \cdot G$
For $i \in [n]$:
  $r_i \xleftarrow{\$} \mathbb{Z}_p$; $\mathbf{V}_i \leftarrow r_i \cdot G \parallel r_i \cdot \mathbf{X} + \mathbf{V}'_i$
Return $(\mathsf{tk} \leftarrow (\mathsf{tk}', \mathbf{x}), \mathcal{V} \leftarrow \{\mathbf{V}_i\}_{i=1}^n)$

$\underline{\mathsf{Trace}(\mathsf{tk}, \mathbf{U} = (U_k)_{k=0}^{\ell'})}$:
$\mathbf{U}' \leftarrow (U_1, \ldots, U_{\ell'})$
Return $\mathcal{I} \leftarrow \mathsf{Trace}'(\mathsf{tk}', \mathbf{U}' - \mathbf{x} \cdot U_0)$

Fig. 6. Our anonymous LST scheme T

Theorem 17. *The LST scheme* T *is anonymous under the* DDH *assumption in* $\mathbb{G}$*. More precisely, it holds that* $\mathbf{Adv}^{\mathrm{ano}}_{\mathsf{T},A}(\lambda, n) \leq \mathbf{Adv}^{\mathrm{ddh}}_{\mathbb{G}}(\lambda)$ *for any PPT adversary* A*, where* $\mathbf{Adv}^{\mathrm{ddh}}_{\mathbb{G}}(\lambda)$ *denotes the best advantage a PPT algorithm can get in solving the* DDH *problem in* $\mathbb{G}$*.*

Proof. The proof is done via a sequence of hybrid games. The advantage of an adversary A in game $\mathcal{G}_i$ is denoted by

$$\mathbf{Adv}(\mathcal{G}_i) = \left| \Pr[\mathcal{G}_i = 1] - \frac{1}{2} \right|.$$

Hybrid Game $\mathcal{G}_0$*.* This corresponds to the real security game $\mathbf{Exp}^{\mathrm{ano}}_{\mathsf{T},A}(1^\lambda, 1^n)$. We recall the construction of the set $\mathcal{V}$. First, the Gen algorithm samples random vectors $\mathbf{s} = (s_i)_{i=1}^n \xleftarrow{\$} (\mathbb{Z}_p^*)^n$ and $\mathbf{x} \xleftarrow{\$} \mathbb{Z}_p^{\ell'}$ and computes $\mathbf{X} = \mathbf{x} \cdot G$ for some fixed generator $G \in \mathbb{G}^*$. Then, for each $i \in [n]$, the algorithm samples $\mathbf{d}_i = (d_{i,1}, \ldots, d_{i,K}) \xleftarrow{\$} [J]^K$, $G_i \xleftarrow{\$} \mathbb{G}^*$ and $r_i \xleftarrow{\$} \mathbb{Z}_p$, and computes $\mathbf{V}_i = (r_i \cdot G \parallel r_i \cdot \mathbf{X} + \mathbf{V}'_i)$ where

$$\mathbf{V}'_i = \left(V'_{i,k,j} = \begin{cases} (G_i, s_i \cdot G_i) & \text{if } j = d_{i,k} \\ (0,0) & \text{otherwise} \end{cases} \right)_{k \in [K], j \in [J]}.$$

Hybrid Game $\mathcal{G}_1$*.* We slightly modify the Gen algorithm. Instead of sampling $\mathbf{x} \xleftarrow{\$} \mathbb{Z}_p^n$ and computing $\mathbf{X} = \mathbf{x} \cdot G$, it directly samples $\mathbf{X} \xleftarrow{\$} \mathbb{G}^{\ell'}$ now. Furthermore, the algorithm samples a vector $\mathbf{t} = (t_i)_{i=1}^n \xleftarrow{\$} (\mathbb{Z}_p^*)^n$ and sets $\mathbf{V}_i = (r_i \cdot G \parallel r_i \cdot \mathbf{X} + \mathbf{v}'_i \cdot G)$ where the vector $\mathbf{v}'_i \in \mathbb{Z}_p^{\ell'}$ is defined as follows

$$\mathbf{v}'_i = \left(v'_{i,k,j} = \begin{cases} (t_i, s_i t_i) & \text{if } j = d_{i,k} \\ (0,0) & \text{otherwise} \end{cases} \right)_{k \in [K], j \in [J]}.$$

Since the distribution of $\mathcal{V}$ does not change, it follows $\mathbf{Adv}(\mathcal{G}_1) = \mathbf{Adv}(\mathcal{G}_0)$.

Hybrid Game $\mathcal{G}_2$. We embed a Diffie-Hellman tuple (X, Y, Z) in basis G. The challenger samples $\boldsymbol{\mu}, \boldsymbol{\nu} \xleftarrow{\$} \mathbb{Z}_p^{\ell'}$ and sets $\mathbf{X} = \boldsymbol{\mu} \cdot X + \boldsymbol{\nu} \cdot G$. Clearly, this does not change the distribution of $\mathbf{X}$. Furthermore, we modify the implementation of $\mathcal{O}\mathsf{SoF}$. First, if $b = 1$, we simply replace $\mathcal{X}$ with $\mathcal{V}$ and continue as in the case $b = 0$. On input a set $\mathcal{X} = \{\mathbf{V}_{i_1}, \ldots, \mathbf{V}_{i_d}\} \subseteq \mathcal{V}$, the oracle samples random scalars $\omega_1, \ldots, \omega_d \xleftarrow{\$} \mathbb{Z}_p$ and computes $\mathbf{U}_1 = (\omega_1 r_{i_1} \cdot Y \parallel r_{i_1} \cdot \mathbf{Z} + \mathbf{v}'_{i_1} \cdot Y)$ where $\mathbf{Z} = \omega_1 \boldsymbol{\mu} \cdot Z + \omega_1 \boldsymbol{\nu} \cdot Y$. Subsequently, it returns $\mathbf{U} = \mathbf{U}_1 + \sum_{j=2}^{d} \omega_j \cdot \mathbf{V}_{i_j}$. Note that if $Y = y \cdot G$ for some (unknown) $y \in \mathbb{Z}_p$, then $\mathbf{U}_1 = \omega_1 y \cdot \mathbf{V}_{i_1}$, i.e. $\mathbf{U}_1$ is a random multiple of $\mathbf{V}_{i_1}$. Thus, the oracle still returns a uniformly random element of $\mathsf{span}(\mathcal{X})$ which implies that the games $\mathcal{G}_3$ and $\mathcal{G}_2$ are again perfectly indistinguishable and $\mathbf{Adv}(\mathcal{G}_2) = \mathbf{Adv}(\mathcal{G}_1)$.

Hybrid Game $\mathcal{G}_3$. We replace the above Diffie-Hellman tuple with a random tuple $(X, Y, Z) \xleftarrow{\$} \mathbb{G}^3$. Thus, we have $\mathbf{Adv}(\mathcal{G}_3) \geq \mathbf{Adv}(\mathcal{G}_2) - \mathbf{Adv}^{\mathrm{ddh}}_{\mathbb{G}}(\lambda)$. Moreover, we observe that $\mathbf{Z}$ (and thus $\mathbf{U}$) are uniformly random vectors in $\mathbb{G}^\ell$ now. Therefore, it follows that $\mathbf{Adv}(\mathcal{G}_3) = 0$.

Using a hybrid argument, we conclude that $\mathbf{Adv}^{\mathrm{ano}}_{\mathsf{T},A}(\lambda, n) \leq \mathbf{Adv}^{\mathrm{ddh}}_{\mathbb{G}}(\lambda)$. □

Remark 18 (Randomizability). Let $\mathsf{T} = (\mathsf{Gen}, \mathsf{Trace})$ be a LST scheme and let $(\mathsf{tk}, \mathcal{V} = \{\mathbf{V}_i\}_{i=1}^n) \leftarrow \mathsf{Gen}(1^\lambda, 1^n)$. Given a vector $\mathbf{U} \in \mathsf{span}(\mathcal{X})$ for some subset $\mathcal{X} \subseteq \mathcal{V}$, it is easy to find another vector $\mathbf{U}'$ in $\mathsf{span}(\mathcal{X})$ unlinkable to $\mathbf{U}$. Indeed, one can simply sample a scalar $\omega \xleftarrow{\$} \mathbb{Z}_p$ and set $\mathbf{U}' = \omega \cdot \mathbf{U}$. Then the unlinkability follows from the DL assumption in $\mathbb{G}$. This randomization is even possible without knowledge of the vectors in $\mathcal{X}$.

However, this method creates several unlinkable representations of the same subspace $\mathsf{span}(\mathcal{X})$ rather than of the same vector $\mathbf{U}$. To achieve the latter, one can add a dummy vector $\mathbf{V}_0$ by running $(\mathsf{tk}, \mathcal{V} = \{\mathbf{V}_i\}_{i=0}^n) \leftarrow \mathsf{Gen}(1^\lambda, 1^{n+1})$ instead of $\mathsf{Gen}(1^\lambda, 1^n)$. Then one sets $\mathcal{V}' = \{\mathbf{V}_i\}_{i=1}^n$ and defines an equivalence relation on $\mathsf{span}(\mathcal{V}) \times \mathsf{span}(\mathcal{V})$ where vectors are equivalent if and only if their orthogonal projection onto the subspace $\mathsf{span}(\mathcal{V}')$ is equal. The randomization of a vector $\mathbf{U}$ inside its equivalence class can be done by sampling a scalar $\omega \xleftarrow{\$} \mathbb{Z}_p$ and computing $\mathbf{U}' = \mathbf{U} + \omega \cdot \mathbf{V}_0$. If the vector $\mathbf{V}_0$ is public, then the randomization can be done by everyone. Furthermore, if T is anonymous, then, for any $\mathbf{U}_1, \mathbf{U}_2 \in \mathsf{span}(\mathcal{V})$, the distributions $\{(\mathbf{U}_1, \mathbf{U}_2, \mathbf{U}'_1, \mathbf{U}'_2)\}$ and $\{(\mathbf{U}_1, \mathbf{U}_2, \mathbf{U}'_2, \mathbf{U}'_1)\}$ are computationally indistinguishable, where $\mathbf{U}'_1 = \mathbf{U}_1 + \omega_1 \cdot \mathbf{V}_0$ and $\mathbf{U}'_2 = \mathbf{U}_2 + \omega_2 \cdot \mathbf{V}_0$ for $\omega_1, \omega_2 \xleftarrow{\$} \mathbb{Z}_p$. If T is $(c+1)$-correct, then the "randomizable" variant (where $\mathbf{V}_0$ is public) is still c-correct. Indeed, one can simply run Trace and remove the index 0 from its output set.

Note that in our concrete construction (Fig. 6), a randomizable, c-correct LST scheme can even be obtained from a c-correct LST scheme (instead of a $(c+1)$-correct scheme). Using the notation of Fig 6, if one chooses $\mathbf{V}_0 = (r_0 \cdot G \parallel r_0 \cdot \mathbf{X} + \mathbf{V}'_0)$ for $\mathbf{V}'_0 = \mathbf{0}^{(\ell')}$, then the ElGamal encryption preserves all security guarantees, but one avoids interferences of $\mathbf{V}'_0$ with $\mathbf{V}'_1, \ldots, \mathbf{V}'_n$ inside the underlying scheme T'. Thus, c-correctness suffices.

Remark 19 (Strong Correctness). It is worth mentioning that the encrypted LST scheme in Fig. 6 also satisfies a stronger correctness notion. The security experiment in Fig. 3 states that the adversary sees only c out of the n vectors in $\mathcal{V} = \{\mathbf{V}_1, \ldots, \mathbf{V}_n\}$. As mentioned above, for the non-encrypted LST scheme described in Fig. 4, this is a necessary restriction since a vector $\mathbf{V}_i \in \mathcal{V}$ does not hide the embedded vector $\mathbf{d}_i$ chosen during its creation. If an adversary A knew more than c vectors, then it could specifically create "collisions", i.e. it could combine vectors $\mathbf{V}_{i_0}$ and $\mathbf{V}_{i_1}$ for which the corresponding vectors $\mathbf{d}_{i_1}$ and $\mathbf{d}_{i_1}$ coincide in some positions.

However, in the encrypted scheme it follows from the semantic security of the ElGamal encryption scheme that the vectors $\mathbf{d}_1, \ldots, \mathbf{d}_n$ cannot be efficiently recovered from $\mathbf{V}_1, \ldots, \mathbf{V}_n$. Thus, even if the adversary knows the entire set $\mathcal{V}$, it is not able to determine which of the vectors must be combined to create many collisions. (Intuitively, the tracing fails if for one vector, the adversary can create a collision in each of its K blocks.) Therefore, we do not need

to bound the number of vectors an adversary is allowed to see, but only the number of vectors it is allowed to combine in the challenge vector $\mathbf{U}$. This stronger security game is shown in Fig. 7.

$(\mathsf{tk}, \mathcal{V} = \{\mathbf{V}_1, \ldots, \mathbf{V}_n\}) \leftarrow \mathsf{Gen}(1^\lambda, 1^n)$
$\mathbf{U} \leftarrow A(\mathsf{tk}, \mathcal{V})$; $\mathcal{I} \leftarrow \mathsf{Trace}(\mathsf{tk}, \mathbf{U})$
If $\mathbf{U} \notin \mathsf{span}_c(\mathcal{V})$, return 0
If $\{\mathbf{V}_i : i \in \mathcal{I}\} \neq \bigcap_{\mathcal{X} \subseteq \mathcal{V}, \mathbf{U} \in \mathsf{span}(\mathcal{X})} \mathcal{X}$, return 1
Return 0

Fig. 7. Security game $\mathbf{Exp}^{c\text{-scor}}_{\mathsf{T},A}(1^\lambda, 1^n)$ for strong c-correctness

6 A Model for Linearly-Homomorphic Group Signatures

Group signatures were originally proposed by [Cv91], with a formal security model in [BMW03]. We build on their definition by combining it with linearly-homomorphic signatures. More precisely, we define a linearly-homomorphic group signature scheme as follows.

Definition 20 (Linearly-Homomorphic Group Signature (LH-GSig)). *A LH-GSig scheme with tag space $\mathcal{T}$ and message space $\mathbb{Z}_p^n$, for a positive integer n and a prime number p, consists of the following polynomial-time algorithms.*

$\mathsf{GKg}(1^\lambda, 1^\kappa)$*: This algorithm takes as input a tuple $(1^\lambda, 1^\kappa)$, where λ is the security parameter and κ is the group size, and returns a tuple $(\mathsf{gpk}, \mathsf{gmsk}, \mathbf{gsk})$, where gpk is the group public key, gmsk is the group manager's secret key and $\mathbf{gsk}$ is a κ-vector of keys with $\mathsf{gsk}[i]$ being the secret signing key of group member $i \in [\kappa]$. We will assume that gmsk and all $\mathsf{gsk}[i]$ implicitly contain gpk.*

$\mathsf{GSig}(\mathsf{gsk}[i], \tau, \mathbf{m})$*: On input a secret signing key $\mathsf{gsk}[i]$ for some $i \in [\kappa]$, a tag $\tau \in \mathcal{T}$ and a message $\mathbf{m} \in \mathbb{Z}_p^n$, this algorithm returns a signature Σ of $\mathbf{m}$ under the tag τ.*

$\mathsf{GDrv}(\mathsf{gpk}, \tau, (\omega_j, \Sigma_j)_{j=1}^d)$*: On input the group public key gpk, a tag $\tau \in \mathcal{T}$ and d tuples of weights $\omega_j \in \mathbb{Z}_p$ and signatures Σ_j, this algorithm returns a signature Σ on the vector $\mathbf{m} = \sum_{j=1}^d \omega_j \cdot \mathbf{m}_j$ under the tag τ, where Σ_j is a signature on the message $\mathbf{m}_j \in \mathbb{Z}_p^n$ under τ.*

$\mathsf{GVf}(\mathsf{gpk}, \tau, \mathbf{m}, \Sigma)$*: On input the group public key gpk, a tag $\tau \in \mathcal{T}$, a message $\mathbf{m} \in \mathbb{Z}_p^n$ and a signature Σ, this algorithm returns 1 if $\tau \in \mathcal{T}$ and Σ is valid relative to gpk and τ, and 0 otherwise.*

$\mathsf{Open}(\mathsf{gmsk}, \tau, \mathbf{m}, \Sigma)$*: On input the group manager's secret key gmsk, a tag $\tau \in \mathcal{T}$, a vector $\mathbf{m} \in \mathbb{Z}_p^n$ and a signature Σ, this algorithm returns a subset $\mathcal{A} \subseteq [\kappa]$ containing the signers of Σ if $\mathsf{GVf}(\mathsf{gpk}, \tau, \mathbf{m}, \Sigma) = 1$, and $\perp$ otherwise.*

Correctness. LH-GSig schemes must meet two correctness requirements concerning the verification and the opening of signatures. The former is very similar to the case of LH-Sig schemes (Definition 3). Let $(\mathsf{gpk}, \mathsf{gmsk}, \mathbf{gsk}) \leftarrow \mathsf{GKg}(1^\lambda, 1^\kappa)$ be any keys and τ be any tag in $\mathcal{T}$. Then,

(a) for each $\mathbf{m} \in \mathbb{Z}_p^n$, $i \in [\kappa]$ and $\Sigma \leftarrow \mathsf{GSig}(\mathsf{gsk}[i], \tau, \mathbf{m})$, the scheme satisfies $\mathsf{GVf}(\mathsf{gpk}, \tau, \mathbf{m}, \Sigma) = 1$, and

(b) for any list $(\omega_j, \mathbf{m}_j, \Sigma_j)_{j=1}^d$ such that $\mathsf{GVf}(\mathsf{gpk}, \tau, \mathbf{m}_j, \Sigma_j) = 1$ for each $j \in [d]$, if $\Sigma \leftarrow \mathsf{GDrv}(\mathsf{gpk}, \tau, (\omega_j, \Sigma_j)_{j=1}^d)$, then $\mathsf{GVf}(\mathsf{gpk}, \tau, \sum_{j=1}^d \omega_j \cdot \mathbf{m}_j, \Sigma) = 1$.

The second aspect asks that the opening algorithm correctly recovers the identity of the signers from an honestly generated signature. Note that, since LH-GSig schemes allow the combination of signatures issued by different group members, the opening algorithm returns a subset

of $[\kappa]$ rather than a single element. For a LH-GSig scheme Σ, a positive integer c and a PPT adversary A, we define the experiment $\mathbf{Exp}^{c\text{-cor}}_{\Sigma,A}(1^\lambda, 1^\kappa)$ as shown in Fig. 8. We say that openings of Σ are c-correct if for any PPT adversary A and any integer κ, there exists a negligible function negl such that

$$\mathbf{Adv}^{c\text{-cor}}_{\Sigma,A}(\lambda,\kappa) = \Pr\left[\mathbf{Exp}^{c\text{-cor}}_{\Sigma,A}(1^\lambda,1^\kappa)=1\right] \leq \mathsf{negl}(\lambda).$$

$\underline{\mathsf{Initialize}(1^\lambda, 1^\kappa)}$:
$\mathcal{H} \leftarrow \emptyset$
$(\mathsf{gpk}, \mathsf{gmsk}, \mathbf{gsk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^\kappa)$
Return gpk

$\underline{\mathcal{O}\mathsf{GDrv}((h_j, \omega_j)_{j=1}^d)}$:
$\{(\tau_j, \mathcal{I}_j, \mathbf{m}_j, \Sigma_j)\}_j \leftarrow \mathsf{Lookup}_{\mathcal{H}}(\{h_j\}_{j=1}^d)$
If $\exists j \in [2; d]$ s.t. $\tau_j \neq \tau_1$, return $\perp$
$\mathbf{m} \leftarrow \sum_{j=1}^d \omega_j \cdot \mathbf{m}_j$
$\Sigma \leftarrow \mathsf{GDrv}(\mathsf{pk}, \tau_1, (\omega_j, \Sigma_j)_{j=1}^d)$
Pick new handle h
$\mathcal{H} \leftarrow \mathcal{H} \cup \{(h, (\tau_1, \mathcal{I} = \bigcup_{j=1}^d \mathcal{I}_j, \mathbf{m}, \Sigma))\}$
Return (h, Σ)

$\underline{\mathcal{O}\mathsf{GSig}(i, \tau, \mathbf{m})}$:
$\Sigma \leftarrow \mathsf{GSig}(\mathsf{gsk}[i], \tau, \mathbf{m})$
Pick new handle h
If $\mathbf{m} = \mathbf{0}^{(n)}$, then $\mathcal{I} \leftarrow \emptyset$; else $\mathcal{I} \leftarrow \{i\}$
$\mathcal{H} \leftarrow \mathcal{H} \cup \{(h, (\tau, \mathcal{I}, \mathbf{m}, \Sigma))\}$
Return (h, Σ)

$\underline{\mathsf{Finalize}(h)}$:
$(\tau, \mathcal{I}, \mathbf{m}, \Sigma) \leftarrow \mathsf{Lookup}_{\mathcal{H}}(h)$
If $|\mathcal{I}| \leq c$ and $\mathsf{Open}(\mathsf{gmsk}, \mathbf{m}, \Sigma) \neq \mathcal{I}$,
return 1
Return 0

Fig. 8. Security game $\mathbf{Exp}^{c\text{-cor}}_{\Sigma,A}(1^\lambda, 1^\kappa)$ for c-correctness of openings

Since the unforgeability of LH-Sig schemes does not exclude that signatures of the zero vector can be created without knowledge of the secret key, we prefer not to trace signers of this vector in general. However, if desired, our construction could easily be modified to allow the creation of deliberately traceable signatures on the zero vector.

Security – Anonymity. Intuitively, anonymity requires that an adversary not in possession of the group manager's secret key gmsk cannot efficiently recover the identities of the signers from a signature. As usual, to win the security game an adversary does not need to recover the identity of a signer from a signature, but it only needs to distinguish which of two (collections of) signers of its choice signed a target message of its choice.

Definition 21 (Anonymity). *For a LH-GSig scheme Σ and a PPT adversary A, we define the experiment $\mathbf{Exp}^{\text{ano}}_{\Sigma,A}(1^\lambda, 1^\kappa)$ as shown in Fig. 9. The "Fresh or Derived" oracle $\mathcal{O}$FoD can be called any number of times.*

We say that a LH-GSig scheme Σ is anonymous if for any PPT adversary A and any polynomial $\kappa = \kappa(\lambda)$, there exists a negligible function negl *such that*

$$\mathbf{Adv}^{\text{ano}}_{\Sigma,A}(\lambda,\kappa) = \left|\Pr\left[\mathbf{Exp}^{\text{ano}}_{\Sigma,A}(1^\lambda,1^\kappa)=1\right] - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda).$$

Our definition captures various cases. This means the adversary can win the security game if it is able to distinguish the given signatures in either of the following scenarios:

- The adversary is given two fresh signatures created by running the Sign algorithm on input the signing keys of two different group members.
- The adversary is given a fresh signature created by running Sign and a derived signature created using the DeriveSign algorithm and a set of "parent" signatures
- The adversary is given two signatures derived from different sets of parent signatures.

$\underline{\mathsf{Initialize}(1^\lambda, 1^\kappa)}$:
$b \xleftarrow{\$} \{0,1\}$
$(\mathsf{gpk}, \mathsf{gmsk}, \mathbf{gsk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^\kappa)$
Return $\mathbf{gsk}$

$\underline{\mathsf{Finalize}(b')}$: Return $(b = b')$

$\underline{\mathcal{O}\mathsf{FoD}(i, \tau, (\omega_j, \mathbf{m}_j, i_j)_{j=1}^d)}$:
$\Sigma_0^* \leftarrow \mathsf{GSig}(\mathsf{gsk}[i], \tau, \sum_{j=1}^d \omega_j \cdot \mathbf{m}_j)$
$(\Sigma_j \leftarrow \mathsf{GSig}(\mathsf{gsk}[i_j], \tau, \mathbf{m}_j))_{j=1}^d$
$\Sigma_1^* \leftarrow \mathsf{GDrv}(\tau, (\omega_j, \Sigma_j)_{j=1}^d)$
Return $(\Sigma_b^*, (\Sigma_j)_{j=1}^d)$

Fig. 9. Security game $\mathbf{Exp}^{\text{ano}}_{\Sigma,A}(1^\lambda, 1^\kappa)$ for anonymity

Also, our security game ensures that signatures created by the same group members are unlinkable. This is a consequence of the fact that the adversary is given **gsk** (i.e. all signing keys). Therefore, it can simulate a signing oracle itself.

On the other hand, note that the adversary in $\mathbf{Exp}^{\text{ano}}_{\Sigma,A}(1^\lambda, 1^\kappa)$ does not have access to an opening oracle. As observed by Bellare *et al.* [BMW03], this would require to use an IND-CCA secure encryption scheme. However, such encryption schemes are not malleable which is crucial for our signature scheme to be linearly homomorphic. We therefore adapt the IND-CPA version of anonymity proposed by Boneh *et al.* with the suggested relaxations. For more details see Section 5.1 of [BBS04].

Security – Traceability. In case of misuse, signer anonymity can be revoked by the group manager. For this to be an effective mechanism, we require that no colluding subset of group members (of size at most c) can create signatures that are opened incorrectly or cannot be opened at all. This is even true if the coalition is in possession of the group manager's secret key gmsk.

Definition 22 (Traceability). *For a LH-GSig scheme Σ with tag space $\mathcal{T}$ and message space $\mathbb{Z}_p^n$, a positive integer c and a PPT adversary A we define the experiment $\mathbf{Exp}^{c\text{-tr}}_{\Sigma,A}(1^\lambda, 1^\kappa)$ as shown in Fig. 10.*

The oracles $\mathcal{O}\mathsf{GSig}$ and $\mathcal{O}\mathsf{Corrupt}$ can be called in any order and any number of times. For a tag $\tau \in \mathcal{T}$, $\mathcal{S}_\tau$ denotes the set of all $i \in [\kappa]$ such that there exists a tuple $(\tau, i, \cdot) \in \mathcal{S}$. Similarly, for a tag τ and an $i \in [\kappa]$, we define $\mathcal{S}_{\tau,i}$ to be the set that contains all $\mathbf{m} \in \mathbb{Z}_p^n$ such that $(\tau, i, \mathbf{m}) \in \mathcal{S}$.

We say that a LH-GSig scheme Σ is c-traceable if for any PPT adversary A and any polynomial $\kappa = \kappa(\lambda)$, there exists a negligible function negl such that

$$\mathbf{Adv}^{c\text{-tr}}_{\Sigma,A}(\lambda, \kappa) = \Pr\left[\mathbf{Exp}^{c\text{-tr}}_{\Sigma,A}(1^\lambda, 1^\kappa) = 1\right] \leq \mathsf{negl}(\lambda).$$

$\underline{\mathsf{Initialize}(1^\lambda, 1^\kappa)}$:
$(\mathsf{gpk}, \mathsf{gmsk}, \mathbf{gsk}) \leftarrow \mathsf{GKg}(1^\lambda, 1^\kappa)$
$\mathcal{C} \leftarrow \emptyset$; $\mathcal{S} \leftarrow \emptyset$
Return gmsk

$\underline{\mathcal{O}\mathsf{GSig}(\tau, i, \mathbf{m})}$:
$\mathcal{S} \leftarrow \mathcal{S} \cup \{(\tau, i, \mathbf{m})\}$
Return $\Sigma \leftarrow \mathsf{GSig}(\mathsf{gsk}[i], \tau, \mathbf{m})$

$\underline{\mathcal{O}\mathsf{Corrupt}(i)}$:
$\mathcal{C} \leftarrow \mathcal{C} \cup \{i\}$; return $\mathsf{gsk}[i]$

$\underline{\mathsf{Finalize}(\tau, \mathbf{m}, \Sigma)}$:
Return 0 if
- $\mathsf{Verify}(\mathsf{gpk}, \tau, \mathbf{m}, \Sigma) = 0$, or
- $\mathbf{m} = \mathbf{0}^{(n)}$, or
- $|\mathcal{C} \cup \mathcal{S}_\tau| > c$

Return 1 if
- $\mathsf{Open}(\mathsf{gmsk}, \mathbf{m}, \Sigma) = \emptyset$, or
- $\mathsf{Open}(\mathsf{gmsk}, \mathbf{m}, \Sigma) \nsubseteq \mathcal{C} \cup \mathcal{S}_\tau$, or
- $\mathsf{Open}(\mathsf{gmsk}, \mathbf{m}, \Sigma) = \mathcal{A}$, $\mathcal{A} \cap \mathcal{C} = \emptyset$ and $\mathbf{m} \notin \mathsf{span}(\bigcup_{i \in \mathcal{A}} \mathcal{S}_{\tau,i})$

Return 0

Fig. 10. Security game $\mathbf{Exp}^{c\text{-tr}}_{\Sigma,A}(1^\lambda, 1^\kappa)$ for c-traceability

A reasonable notion of traceability should in particular imply unforgeability. If one removes the adversary's access to $\mathcal{O}\mathsf{Corrupt}$ and provides only the group public key gpk instead of the group manager's secret key gmsk, then $\mathbf{Exp}^{\text{c-tr}}_{\Sigma,A}(1^\lambda, 1^\kappa)$ roughly equals $\mathbf{Exp}^{\text{unf}'}_{\Sigma,A}(1^\lambda)$ as defined in Fig. 2. However, according to Remark 6, to obtain equivalence to the more general security experiment $\mathbf{Exp}^{\text{unf}}_{\Sigma,A}(1^\lambda)$ as defined in Fig. 1, the signatures must in addition be context hiding. But this property is always implied by the anonymity of a LH-GSig scheme. Thus, by a proof similar to that of Lemma A.4 in [ABC+11], one can replace the oracles $\mathcal{O}\mathsf{Sign}$, $\mathcal{O}\mathsf{DeriveSign}$ and $\mathcal{O}\mathsf{Reveal}$ in $\mathbf{Exp}^{\text{unf}}_{\Sigma,A}(1^\lambda)$ with the single oracle $\mathcal{O}\mathsf{GSig}$ as in $\mathbf{Exp}^{\text{c-tr}}_{\Sigma,A}(1^\lambda, 1^\kappa)$. So our traceability notion implies indeed a general variant of unforgeability.

Remark 23. One-time linearly-homomorphic signatures (without tags) have the general problem that everyone can create a signature for any message in the span of previously signed messages. Therefore, the more signatures are available, the less meaningful individual signatures become. To overcome this problem, one introduces tags and allows aggregation only if signatures were created with respect to the same tag. In this way, the number of signatures that can be combined is decreased, and individual signatures gain more significance.

The tracing mechanism in our LH-GSig model faces a very similar issue. Indeed, the fact of being traced becomes less significant the more signatures are available, which is due to basic attacks of the following shape: suppose two honest signers B and C have published signatures Σ_B and Σ_C of respective messages $\mathbf{m}_B$ and $\mathbf{m}_C$. Whenever a malicious signer A wants to sign an evil message $\mathbf{m}_A$, it may instead sign $\mathbf{m}_A - (\mathbf{m}_B + \mathbf{m}_C)$ and combine that signature with Σ_B and Σ_C to obtain a signature for $\mathbf{m}_A$. Every signature produced in this way traces back to A, B and C, and the tracing authority has no chance to identify the malicious signer among these three.

We think that this weakness in the traceability should be seen in the same spirit as the previously mentioned lack of significance in the context of general linearly-homomorphic signatures. Therefore, it seems natural to defeat attacks like the one described above by applying the same countermeasures, i.e. to introduce tags (such as a time stamp). This lowers the number of signatures available during a certain time period and, in particular, it prevents the malicious signer A from reusing the signatures Σ_B and Σ_C forever. In many practical scenarios it seems unlikely that a malicious signer knows signatures of the same (honest) signers in every time interval. However, if the malicious signer mixes its signature in each time interval with signatures from different signers (or it does not mix its signatures at all), then A will soon be identified as the unique common point.

Another approach is to make not only signatures but also derivations traceable. This can be done (at least for the last step of derivations) by employing an additional group signature layer. More specifically, let $\Sigma' = (\mathsf{GKg}', \mathsf{GSig}', \mathsf{GVf}', \mathsf{Open}')$ be a (classical) group signature scheme whose message space consists of all signatures that can be created using the algorithms GSig and GDrv of Σ. All parties that should be able to derive new signatures are provided with signing keys of Σ'. We call the members of this group the *derivers*, similar to the members of the Σ-group that are called *signers*. For consistency, the group of derivers must contain but is not limited to the group of signers. Signatures in our new scheme consist of a Σ-signature Σ accompanied by a Σ'-signature of Σ. A signer can create a signature of a new message $\mathbf{M}$ by first signing $\mathbf{M}$ using its Σ-key, and then signing the resulting signature again using its Σ'-key. On the other hand, derivers cannot create new Σ-signatures, but they can derive them using (the first component of) existing signatures, and subsequently sign them using their Σ'-key. In this way it is not possible to recover the entire derivation history of signatures. Nevertheless, if a signature of an evil message is published, it is possible to identify a user that participated in the last step of the creation. This is enough to defeat the above scenario: Here A could be identified since it would have to sign the Σ-signature derived from Σ_A, Σ_B and Σ_C using its own Σ'-key.

7 Generic Construction of a LH-GSig scheme

7.1 Properties of LH-Sig schemes

We introduce two properties of LH-Sig schemes that we exploit in our LH-GSig construction.

Zero-Signability. Note that signatures of the zero vector are never considered a valid forgery in the security game $\mathbf{Exp}^{\mathrm{unf}}_{\Sigma,A}(1^\lambda)$ (Fig. 1). Thus, the definition of LH-Sig schemes does not preclude signatures of the zero vector to be efficiently computable without knowledge of the signing key, but it does not require it either. If signatures of the zero vector can be computed under any tag, then we call a LH-Sig scheme zero-signable.

Definition 24 (Zero-Signability). *A LH-Sig scheme* $\Sigma = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{DeriveSign}, \mathsf{Verify})$ *with message space* $\mathbb{G}^n$ *and tag space* $\mathcal{T}$ *is called zero-signable if there exists a polynomial-time algorithm* ZSign *of the following shape:*

$\mathsf{ZSign}(\mathsf{pk}, \tau)$*: On input a public key* pk *and a tag* $\tau \in \mathcal{T}$*, this algorithm returns a signature* Σ *on* $\mathbf{0}^{(n)}$ *under* τ *that is valid with respect to* pk*.*

Universality. We consider a LH-Sig scheme $\Sigma = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{DeriveSign}, \mathsf{Verify})$. Let $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\mathsf{Setup}(1^\lambda))$. Abusing notation, $\mathsf{Sign}(\mathsf{sk}, \tau, \mathbf{M})$ denotes the set of all signatures output by the Sign algorithm on input $(\mathsf{sk}, \tau, \mathbf{M})$ with a nonzero probability. A signature Σ on a message $\mathbf{M}$ is called *universal* with respect to sk if $\Sigma \in \bigcap_{\tau \in \mathcal{T}} \mathsf{Sign}(\mathsf{sk}, \tau, \mathbf{M})$.

The unforgeability implies that the Sign algorithm outputs universal signatures only with negligible probability. Nevertheless, knowledge of the secret key sk may enable an efficient computation of such signatures. In this case, we call the LH-Sig scheme universal.

Definition 25 (Universality). *A LH-Sig scheme* $\Sigma = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{DeriveSign}, \mathsf{Verify})$ *with message space* $\mathbb{G}^n$ *and tag space* τ *is called universal if there exists a polynomial-time algorithm* USign *of the following shape:*

$\mathsf{USign}(\mathsf{sk}, \mathbf{M})$*: On input a signing key* sk *and a message* $\mathbf{M} \in \mathbb{G}^n$*, this algorithm returns a signature* Σ *on* $\mathbf{M}$ *that is universal with respect to* sk*.*

Note that each OT-LH-Sig scheme is trivially universal. We present a generic conversion of a OT-LH-Sig scheme into a universal, zero-signable LH-Sig scheme with tag space $\mathbb{Z}_p^*$ in Appendix A.2.

Remark 26. Let $\Sigma' = (\mathsf{Setup}', \mathsf{KeyGen}', \mathsf{Sign}', \mathsf{DeriveSign}', \mathsf{Verify}')$ be a LH-Sig scheme with message space $\mathcal{M}' = \mathbb{G}^n$ and tag space $\mathcal{T}'$. Then Σ' can easily be turned into a LH-Sig scheme $\Sigma = (\mathsf{Setup}', \mathsf{KeyGen}', \mathsf{Sign}, \mathsf{DeriveSign}', \mathsf{Verify})$ with message space $\mathcal{M} = \mathbb{Z}_p^n$ (and tag space $\mathcal{T} = \mathcal{T}'$) by fixing a generator $G \in \mathbb{G}^*$ and replacing scalar messages $\mathbf{m} \in \mathbb{Z}_p^n$ with $\mathbf{m} \cdot G$, i.e. $\mathsf{Sign}(\mathsf{sk}, \tau, \mathbf{m})$ runs $\mathsf{Sign}'(\mathsf{sk}, \tau, \mathbf{m} \cdot G)$ and $\mathsf{Verify}(\mathsf{pk}, \tau, \mathbf{m}, \Sigma)$ runs $\mathsf{Verify}'(\mathsf{pk}, \tau, \mathbf{m} \cdot G, \Sigma)$.

For $k \in [n]$, let $\mathbf{e}_k$ denote the k-th standard unit vector in $\mathbb{Z}_p^n$ and let $\mathbf{E}_k = \mathbf{e}_k \cdot G$. If Σ' is zero-signable, then universal signatures $\Sigma_1, \ldots, \Sigma_n$ of $\mathbf{E}_1, \ldots, \mathbf{E}_n$ suffice to sign any message $\mathbf{m} = (m_i)_{i=1}^n \in \mathbb{Z}_p^n$ under an arbitrary tag $\tau \in \mathcal{T}$. Indeed, a signature Σ of $\mathbf{m}$ can be obtained by setting $m_0 = 1$ and computing $\Sigma_0 \leftarrow \mathsf{ZSign}'(\mathsf{pk}, \tau)$ and $\Sigma \leftarrow \mathsf{DeriveSign}'(\mathsf{pk}, \tau, (m_k, \Sigma_k)_{k=0}^n)$.

7.2 High-level Description

Our c-traceable LH-GSig scheme with message space $\mathbb{Z}_p^n$ and tag space $\mathcal{T}$ is based on the following building blocks:

- an anonymous, $(c+1)$-correct LST scheme $\mathsf{T} = (\mathsf{Gen}, \mathsf{Trace})$ for linear subspaces of $\mathbb{G}^\ell$, and

– a universal, zero-signable and context-hiding LH-Sig scheme $\Sigma' = (\mathsf{Setup}', \mathsf{KeyGen}', \mathsf{Sign}', \mathsf{DeriveSign}', \mathsf{Verify}')$ with message space $\mathbb{G}^{n+\ell}$ and tag space $\mathcal{T}' = \mathcal{T}$, and the additional algorithms ZSign' and USign'.

Let $(\mathsf{tk}, \mathcal{V} = \{\mathbf{V}_i\}_{i=1}^{\kappa}) \leftarrow \mathsf{Gen}(1^\lambda, 1^\kappa)$ and $(\mathsf{sk}', \mathsf{pk}') \leftarrow \mathsf{KeyGen}'(\mathsf{Setup}'(1^\lambda))$. At a high level, we use the idea of Remark 26 that universal signatures of the unit vectors suffice to sign any message in $\mathbb{Z}_p^n$ under any tag $\tau \in \mathcal{T}$. However, to enable tracing, we provide the i-th group member, for $i \in [\kappa]$, with universal signatures $(\sigma_{i,k})_{k=1}^n$ on $(\mathbf{E}_k \parallel \mathbf{V}_i)_{k=1}^n$ instead of just $(\mathbf{E}_k)_{k=1}^n$. To sign a message $\mathbf{m} = (m_k)_{k=1}^n \in \mathbb{Z}_p^n$, the i-th group member proceeds exactly as in Remark 26. That is, it computes $\sigma_\tau \leftarrow \mathsf{ZSign}'(\mathsf{pk}', \tau)$ and $\sigma \leftarrow \mathsf{DeriveSign}'(\mathsf{pk}, \tau, (m_k, \sigma_{i,k})_{k\in[n]\cup\{\tau\}})$ for $m_\tau = 1$. Then it outputs the signature $\Sigma = (\sigma, \mathbf{C} = \sum_{k=1}^n m_k \cdot \mathbf{V}_i)$, i.e. Σ-signatures are tuples that consists of a Σ'-signature and a vector in $\mathbb{G}^\ell$. For the verification one simply checks that $\mathsf{Verify}'(\mathsf{pk}', \tau, (\mathbf{m} \parallel \mathbf{C}), \sigma) = 1$.

This method obviously generalizes to signature derivations. Given a tag τ and d tuples $(\omega_j, \Sigma_j = (\sigma_j, \mathbf{C}_j))_{j=1}^d$ where Σ_j is a signature on a message $\mathbf{m}_j$ under τ, one obtains a message on $\mathbf{m} = \sum_{j=1}^d \mathbf{m}_j$ by deriving a Σ'-signature $\sigma \leftarrow \mathsf{DeriveSign}'(\mathsf{pk}', \tau, (\omega_j, \sigma_j)_{j=0}^d)$ and outputting $\Sigma = (\sigma, \mathbf{C} = \sum_{j=1}^d \omega_j \cdot \mathbf{C}_j)$. Note that $\mathbf{C}$ is in the span of $\{\mathbf{C}_j\}_{j=1}^d$. Hence, the LST scheme T can be used to recover all group members who participated in the creation of Σ.

However, this construction is not anonymous. While the first component of a Σ-signature is a signature of the *context-hiding* signature scheme Σ', the second component can be used to link signatures created by the same group members. Therefore, we add a randomization mechanism as described in Remark 18. More precisely, we run $(\mathsf{tk}, \mathcal{V} = \{\mathbf{V}_0, \ldots, \mathbf{V}_\kappa\}) \leftarrow \mathsf{Gen}(1^\lambda, 1^{\kappa+1})$ and $\sigma_0 \leftarrow \mathsf{USign}'(\mathsf{sk}', (\mathbf{0}^{(n)} \parallel \mathbf{V}_0))$, and include $(\mathbf{V}_0, \sigma_0)$ in the group public key. Given a Σ signature $\Sigma = (\sigma, \mathbf{C})$ on a message $\mathbf{m}$ under a tag τ, one can randomize it by sampling a random scalar $\omega_0 \xleftarrow{\$} \mathbb{Z}_p$ and setting $\tilde{\Sigma} = (\tilde{\sigma}, \tilde{\mathbf{C}})$ for $\tilde{\sigma} \leftarrow \mathsf{DeriveSign}'(\mathsf{pk}', \tau, ((1, \sigma), (\omega_0, \sigma_0)))$ and $\tilde{\mathbf{C}} \leftarrow \mathbf{C} + \omega_0 \cdot \mathbf{V}_0$. By the anonymity of the LST scheme T, it follows that $\mathbf{C}$ and $\tilde{\mathbf{C}}$ cannot be linked. Furthermore, since σ_0 is a signature on the vector $(\mathbf{0}^{(n)} \parallel \mathbf{V}_0)$, $\tilde{\sigma}$ is a Σ'-signature on $\mathbf{m} \parallel \tilde{\mathbf{C}}$. This in turn implies that $\tilde{\Sigma} = (\tilde{\sigma}, \tilde{\mathbf{C}})$ is still a Σ-signature on the original message $\mathbf{m}$.

7.3 Our Scheme

Figure 11 depicts our LH-GSig scheme. The LST scheme $\mathsf{T} = (\mathsf{Gen}, \mathsf{Trace})$ can be instantiated from the construction presented in Fig. 6. For the LH-Sig scheme $\Sigma' = (\mathsf{Setup}', \mathsf{KeyGen}', \mathsf{Sign}', \mathsf{DeriveSign}', \mathsf{Verify}')$, we provide a construction in Appendix A.2. While both components are proven secure in the AGM, the security analysis of our (generic) LH-GSig scheme is done in the standard model.

The correctness of verifications follows from the correctness of the underlying LH-Sig scheme Σ'. The c-correctness of openings is a consequence of the $(c+1)$-correctness of the LST scheme T. Security is stated in the following theorems.

Theorem 27. *Let c be a positive integer. If Σ' is unforgeable and T is $(c+1)$-correct, then Σ is c-traceable.*

Proof. Let A be any PPT adversary in $\mathbf{Exp}_{\Sigma,A}^{c\text{-tr}}(1^\lambda, 1^\kappa)$. The simulation runs the experiment honestly. Eventually, the adversary calls $\mathsf{Finalize}$ on input $(\tau, \mathbf{m}, \Sigma = (\sigma, \mathbf{C}))$.

For $i \in [\kappa]$, we parse the signing key as $\mathsf{gsk}[i] = (\boldsymbol{\sigma}_i = (\sigma_{i,k})_{k=1}^n, \mathbf{V}_i)$. Furthermore, for $i \in \mathcal{S}_\tau$, let $\mathcal{S}_{\tau,i} = (\mathbf{m}_{i,j})_{j\in[d_i]}$ and, for $j \in [d_i]$, let $\Sigma'_{i,j} = (\sigma'_{i,j}, \mathbf{C}_{i,j})$ denote the corresponding signature issued by $\mathcal{O}\mathsf{GSig}$ on input $(i, \tau, \mathbf{m}_{i,j})$. Abusing notation, we refer by $\mathsf{DeriveSign}'(\mathsf{pk}', \tau, (\omega_j, \sigma_j)_j)$ to the set of all signatures output by the $\mathsf{DeriveSign}'$ algorithm on input $(\mathsf{pk}', \tau, (\omega_j, \sigma_j)_j)$ with a nonzero probability. Then the unforgeability of Σ' states that with overwhelming probability

there exist coefficients ω_0, $(\omega_{i,k})_{i\in\mathcal{C},k\in[n]}$ and $(\omega'_{i,j})_{i\in\mathcal{S}_\tau,j\in[d_i]}$ such that

$$\begin{aligned}\big(\mathbf{m}\cdot G \parallel \mathbf{C}\big) = \omega_0\cdot\big(\mathbf{0}^{(n)} \parallel \mathbf{V}_0\big) + \sum_{i\in\mathcal{C}}\sum_{k=1}^{n}\omega_{i,k}\cdot\big(\mathbf{e}_k\cdot G \parallel \mathbf{V}_i\big) + \\ \sum_{i\in\mathcal{S}_\tau}\sum_{j=1}^{d_i}\omega'_{i,j}\cdot\big(\mathbf{m}_{i,j}\cdot G \parallel \mathbf{V}_i\big) \\ \sigma \in \mathsf{DeriveSign}'\Big(\mathsf{pk}', \tau, \big((\omega_0,\sigma_0),(\omega_{i,k},\sigma_{i,k})_{i\in\mathcal{C},k\in[0;n]}, \\ (\omega'_{i,j},\sigma'_{i,j})_{i\in\mathcal{S}_\tau,j\in[d_i]}\big)\Big)\end{aligned}$$

This implies in particular that $\mathbf{C}$ is in the span of $\{\mathbf{V}_i : i\in\mathcal{C}\cup\mathcal{S}_\tau\cup\{0\}\}$. For the experiment to return 1, the condition $|\mathcal{C}\cup\mathcal{S}_\tau|\le c$ must be met. Then it follows from the $(c+1)$-correctness of T that $\mathsf{Open}(\mathsf{gmsk},\tau,\mathbf{m},\Sigma)$ returns the set of all $i\in[\kappa]$ with the property that $\mathsf{gsk}[i]$ was involved in the computation of Σ. Suppose that $\mathbf{m}\neq\mathbf{0}$ and denote $\mathcal{A}=\mathsf{Open}(\mathsf{gmsk},\mathbf{m},\Sigma)$. Then it follows immediately that

- $\mathcal{A}\neq\emptyset$,
- $\mathcal{A}\subseteq\mathcal{C}\cup\mathcal{S}_\tau$ and
- either $\mathcal{A}\cap\mathcal{C}\neq\emptyset$ or $\mathbf{m}\in\mathsf{span}(\bigcup_{i\in\mathcal{A}}\mathcal{S}_{\tau,i})$. □

Theorem 28. *If Σ' is (computationally) context hiding and T is anonymous, then Σ is anonymous. More precisely, it holds that*

$$\mathbf{Adv}^{\mathrm{ano}}_{\Sigma,A}(\lambda,\kappa)\le\mathbf{Adv}^{\mathrm{ano}}_{\mathsf{T}}(\lambda,\kappa+1)+Q\cdot\mathbf{Adv}^{\mathrm{ch}}_{\Sigma'}(\lambda)$$

for any PPT adversary A, where Q is the number of queries to $\mathcal{O}\mathsf{FoD}$ and $\mathbf{Adv}^{\mathrm{ch}}_{\Sigma,A}(\lambda)$ the best advantage a PPT algorithm can get in distinguishing the distributions $\mathcal{D}_0$ and $\mathcal{D}_1$ in the definition of context hiding (Definition 5).

Proof. The proof is done via a sequence of hybrid games. The advantage of an adversary A in game $\mathcal{G}_\alpha$ is denoted by

$$\mathbf{Adv}(\mathcal{G}_\alpha)=\left|\Pr[\mathcal{G}_\alpha=1]-\frac{1}{2}\right|.$$

Hybrid Game $\mathcal{G}_0$. This corresponds to the real security game $\mathbf{Exp}^{\mathrm{ano}}_{\Sigma,A}(1^\lambda,1^\kappa)$. Recall that for $i\in[\kappa]$, the signing key $\mathsf{gsk}[i]$ consists of a vector $\mathbf{V}_i$ used for the tracing, and a tuple of Σ'-signatures $(\sigma_{i,k})_{k=1}^n$, where $\sigma_{i,k}$ is a signature on $(\mathbf{e}_k\cdot G \parallel \mathbf{V}_i)$ for some fixed generator $G\in\mathbb{G}^*$.

Hybrid Game $\mathcal{G}_{1,k}$. We modify the behavior of the oracle $\mathcal{O}\mathsf{FoD}$ during the first k queries. On input $(i,\tau,(\omega_j,\mathbf{m}_j,i_j)_{j=1}^d)$, we replace the first component of the challenge signature Σ_b^* with a fresh signature on $\sum_{j=1}^d\omega_j\cdot(\mathbf{m}_j\cdot G \parallel \mu_j\cdot\mathbf{C}_j)$ where $\mathbf{m}_j=(m_{j,\nu})_{\nu=1}^n$ and $\mu_j=\sum_{\nu=1}^n m_{j,\nu}$. If $b=0$, then the distribution of Σ_b^* does not change. If $b=1$, then it follows by the context hiding property of Σ' that $\mathbf{Adv}(\mathcal{G}_{1,Q})\ge\mathbf{Adv}(\mathcal{G}_0)-Q\cdot\mathbf{Adv}^{\mathrm{ch}}_{\Sigma'}(\lambda)$.

Hybrid Game $\mathcal{G}_2$. We modify the implementation of $\mathcal{O}\mathsf{FoD}$ again by replacing the second component of Σ_b^* with a random string $\mathbf{C}\xleftarrow{\$}\mathbb{G}^\ell$. Then it follows by the anonymity of T that $\mathbf{Adv}(\mathcal{G}_2)\ge\mathbf{Adv}(\mathcal{G}_{1,Q})-\mathbf{Adv}^{\mathrm{ano}}_{\mathsf{T}}(\lambda,\kappa+1)$. Moreover, we observe that the signature Σ_b^* is independent of the bit b now. Thus, we have $\mathbf{Adv}(\mathcal{G}_2)=0$.

Using a hybrid argument, we conclude that $\mathbf{Adv}^{\mathrm{ano}}_{\Sigma,A}(\lambda,\kappa)\le\mathbf{Adv}^{\mathrm{ano}}_{\mathsf{T}}(\lambda,\kappa+1)+Q\cdot\mathbf{Adv}^{\mathrm{ch}}_{\Sigma'}(\lambda)$ which completes the theorem. □

$\underline{\mathsf{GKg}(1^\lambda, 1^\kappa):}$
$(\mathsf{sk}', \mathsf{pk}') \leftarrow \mathsf{KeyGen}'(\mathsf{Setup}'(1^\lambda))$
Choose any $G \in \mathbb{G}^*$
$(\mathsf{tk}, \{\mathbf{V}_i\}_{i=0}^{\kappa}) \leftarrow \mathsf{Gen}(1^\lambda, 1^{\kappa+1})$
$\sigma_0 \leftarrow \mathsf{USign}'(\mathsf{sk}', (\mathbf{0}^{(n)} \parallel \mathbf{V}_0))$
For $i \in [\kappa]$:
  $(\sigma_{i,k} \leftarrow \mathsf{USign}'(\mathsf{sk}', \mathbf{e}_k \cdot G \parallel \mathbf{V}_i))_{k=1}^n$
  $\mathsf{gsk}[i] \leftarrow ((\sigma_{i,k})_{k=1}^n, \mathbf{V}_i)$
$\mathsf{gpk} \leftarrow (\mathsf{pk}', G, \mathbf{V}_0, \sigma_0)$; $\mathsf{gmsk} \leftarrow \mathsf{tk}$
Return $(\mathsf{gpk}, \mathsf{gmsk}, \mathbf{gsk})$

$\underline{\mathsf{GDrv}(\mathsf{gpk}, \tau, (\omega_j, \Sigma_j = (\sigma_j, \mathbf{C}_j))_{j=1}^d):}$
$\omega_0 \xleftarrow{\$} \mathbb{Z}_p$; $\mathbf{C}_0 \leftarrow \mathbf{V}_0$; $\mathbf{C} \leftarrow \sum_{j=0}^d \omega_j \cdot \mathbf{C}_j$
$\sigma \leftarrow \mathsf{DeriveSign}'(\mathsf{pk}', \tau, (\omega_j, \sigma_j)_{j=0}^d)$
Return $\Sigma \leftarrow (\sigma, \mathbf{C})$

$\underline{\mathsf{GSig}(\mathsf{gsk}[i] = ((\sigma_k)_{k=1}^n, \mathbf{V}), \tau, \mathbf{m}):}$
$(\Sigma_k \leftarrow (\sigma_k, \mathbf{V}))_{k=1}^n$
$\sigma_\tau \leftarrow \mathsf{ZSign}'(\mathsf{pk}', \tau)$; $\Sigma_\tau \leftarrow (\sigma_\tau, \mathbf{0}^{(\ell)})$
Parse $\mathbf{m} = (m_k)_{k=1}^n$; $m_\tau \leftarrow 1$
$\Sigma \leftarrow \mathsf{GDrv}(\mathsf{gpk}, \tau, (m_k, \Sigma_k)_{k \in [n] \cup \{\tau\}})$
Return Σ

$\underline{\mathsf{GVf}(\mathsf{gpk}, \tau, \mathbf{m}, \Sigma = (\sigma, \mathbf{C})):}$
Return $b \leftarrow \mathsf{Verify}'(\mathsf{pk}', \tau, \mathbf{m} \cdot G \parallel \mathbf{C}, \sigma)$

$\underline{\mathsf{Open}(\mathsf{gmsk}, \tau, \mathbf{m}, \Sigma = (\sigma, \mathbf{C})):}$
If $\mathsf{GVf}(\mathsf{gpk}, \tau, \mathbf{m}, \Sigma) = 0$, return $\perp$
$\mathcal{A} \leftarrow \mathsf{Trace}(\mathsf{gmsk}, \mathbf{C})$
Return $\mathcal{A} \setminus \{0\}$

Fig. 11. Our LH-GSig scheme

Note that we could slightly relax the condition on Σ' to be universal. For the scheme to work, it is not important that the signatures σ_0 and $(\sigma_{i,k})_{i \in [\kappa], k \in [n]}$ are valid with respect to each tag $\tau \in \mathcal{T}$. In fact, they do not even have to be valid with respect to any tag, as their validity is never checked. In the scheme, they are only passed to the $\mathsf{DeriveSign}'$ algorithm together with another signature σ_τ which is valid with respect to a tag $\tau \in \mathcal{T}$. Therefore, it is sufficient to require that this derivation works correctly and returns a signature also valid with respect to τ.

Also, note that OT-LH-Sig schemes are in particular universal. Hence, each zero-signable and context-hiding OT-LH-Sig can be used to construct a one-time LH-GSig scheme.

Remark 29. Our concrete LST scheme presented in Sections 4 and 5 allows for two efficiency improvements. First, Remark 15 shows that the vectors $\mathbf{V}_1, \ldots, \mathbf{V}_\kappa$ have a small representation which significantly lowers the size of the signing keys. Furthermore, according to Remark 18, our LST scheme allows choosing $\mathbf{V}_0$ in such a way that c-correctness of T suffices to obtain c-correctness of Σ. Thus, the necessary size of ℓ can be (slightly) decreased which lowers the size of both signing keys and signatures.

7.4 Efficiency

The size of signatures in the c-traceable LH-GSig scheme Σ presented in Figure 11 depends mainly on the instantiation of the underlying LH-Sig and LST schemes denoted Σ' and T respectively. More precisely, a Σ-signature consists of a Σ'-signature and a vector $\mathbf{C}$ with entries in $\mathbb{G}_1$ whose dimension depends on T. In Appendix A.2, we present an instantiation for Σ' where signatures consist of a small constant number of elements of $\mathbb{G}_1$. For T, if one uses our LST scheme presented in Sections 4 and 5, then signatures consist of $\Theta(c^2 \cdot \log(\kappa/\varepsilon))$ group elements where κ denotes the size of the group and ε is the maximum acceptable probability that the tracing will fail. On the other hand, a naive solution which uses a LST scheme based on the "naive FIPP code" described in Remark 11 (i.e. each group member puts its identity in a separate coordinate) would yield signatures of size $\mathcal{O}(\kappa)$ group elements. Thus, Σ-signatures in the naive solution grow asymptotically faster (in the size of the group) than in our construction.

Acknowledgments

This work was supported by the France 2030 ANR Project ANR-22-PECY-003 SecureCompute.

References

ABC+11. Jae Hyun Ahn, Dan Boneh, Jan Camenisch, Susan Hohenberger, abhi shelat, and Brent Waters. Computing on authenticated data. Cryptology ePrint Archive, Report 2011/096, 2011. https://eprint.iacr.org/2011/096.

BBS04. Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 41–55. Springer, Heidelberg, August 2004.

BFKW09. Dan Boneh, David Freeman, Jonathan Katz, and Brent Waters. Signing a linear subspace: Signature schemes for network coding. In Stanislaw Jarecki and Gene Tsudik, editors, *PKC 2009*, volume 5443 of *LNCS*, pages 68–87. Springer, Heidelberg, March 2009.

BMW03. Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 614–629. Springer, Heidelberg, May 2003.

BS95. Dan Boneh and James Shaw. Collusion-secure fingerprinting for digital data (extended abstract). In Don Coppersmith, editor, *CRYPTO'95*, volume 963 of *LNCS*, pages 452–465. Springer, Heidelberg, August 1995.

CFN94. Benny Chor, Amos Fiat, and Moni Naor. Tracing traitors. In Yvo Desmedt, editor, *CRYPTO'94*, volume 839 of *LNCS*, pages 257–270. Springer, Heidelberg, August 1994.

Cv91. David Chaum and Eugène van Heyst. Group signatures. In Donald W. Davies, editor, *EUROCRYPT'91*, volume 547 of *LNCS*, pages 257–265. Springer, Heidelberg, April 1991.

ElG84. Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 10–18. Springer, Heidelberg, August 1984.

FKL18. Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, Heidelberg, August 2018.

HPP20. Chloé Hébant, Duong Hieu Phan, and David Pointcheval. Linearly-homomorphic signatures and scalable mix-nets. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 597–627. Springer, Heidelberg, May 2020.

HvLT98. Henk D.L Hollmann, Jack H van Lint, Jean-Paul Linnartz, and Ludo M.G.M Tolhuizen. On codes with the identifiable parent property. *Journal of Combinatorial Theory, Series A*, 82(2):121–133, 1998.

LPJY13. Benoît Libert, Thomas Peters, Marc Joye, and Moti Yung. Linearly homomorphic structure-preserving signatures and their applications. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 289–307. Springer, Heidelberg, August 2013.

Tar03. Gábor Tardos. Optimal probabilistic fingerprint codes. In *35th ACM STOC*, pages 116–125. ACM Press, June 2003.

A Linearly-Homomorphic Signatures

A.1 Original Experiment for Context Hiding

Fig. 12 shows the original security game for context hiding proposed by Ahn *et al.* [ABC+11], using our notations. The oracles $\mathcal{O}\mathsf{Sign}$, $\mathcal{O}\mathsf{DeriveSign}$, $\mathcal{O}\mathsf{Reveal}$ and $\mathcal{O}\mathsf{FoD}$ can be called in any order and any number of times, except for the "Fresh or Derived" oracle $\mathcal{O}\mathsf{FoD}$ which can be called only once. The operation $\mathsf{Lookup}_{\mathcal{H}}$ is defined in the same way as in Definition 4, except that it returns 2-tuples instead of 3-tuples now.

A.2 Generic Conversion from OT-LH-Sig to LH-Sig

Let $\Sigma' = (\mathsf{Setup}', \mathsf{KeyGen}', \mathsf{Sign}', \mathsf{DeriveSign}', \mathsf{Verify}')$ be a OT-LH-Sig scheme with message space $\mathcal{M}' = \mathbb{G}^{n+3}$. We complete it into a universal, zero-signable LH-Sig scheme $\Sigma = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{DeriveSign}, \mathsf{Verify})$ with tag space $\mathcal{T} = \mathbb{Z}_p^*$ and message space $\mathcal{M} = \mathbb{G}^n$ as shown in Fig. 13.

The conversion is similar (but not equal) to that presented in Appendix C.5 of [HPP20]. In the HPP construction, the tags are tuples (τ, H) chosen at random from $\mathbb{Z}_p^* \times \mathbb{G}^*$. To sign a message $\mathbf{M} \in \mathcal{M}$, one extends $\mathbf{M}$ into a vector $\mathbf{M}' \in \mathcal{M}'$ with the three additional components $(H, \tau \cdot H, \tau^2 \cdot H)$ and computes a signature of $\mathbf{M}'$ using Sign'. The unforgeability of the scheme is a consequence of the following observation. For distinct Square Diffie-Hellman tuples $\mathbf{T}_1, \mathbf{T}_2 \in \mathbb{G}^3$, it is computationally hard under the DL assumption in $\mathbb{G}$ to find

$\underline{\mathsf{Initialize}(1^\lambda)}$:
$b \xleftarrow{\$} \{0,1\}$; $\mathcal{H} \leftarrow \emptyset$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{Setup}(1^\lambda))$
Return pk

$\underline{\mathcal{O}\mathsf{Sign}(\tau, \mathbf{M})}$:
$\Sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, \tau, \mathbf{M})$
Pick new handle h
$\mathcal{H} \leftarrow \mathcal{H} \cup \{(h, (\tau, \Sigma))\}$
Return h

$\underline{\mathcal{O}\mathsf{Reveal}(h)}$:
$(\tau, \Sigma) \leftarrow \mathsf{Lookup}_{\mathcal{H}}(h)$
Return Σ

$\underline{\mathcal{O}\mathsf{DeriveSign}((h_j, \omega_j)_{j=1}^d)}$:
$\{(\tau_j, \Sigma_j)\}_{j=1}^d \leftarrow \mathsf{Lookup}_{\mathcal{H}}(\{h_j\}_{j=1}^d)$
If $\exists j, k \in [d]$ s.t. $\tau_j \neq \tau_k$, return $\perp$
$\Sigma \leftarrow \mathsf{DeriveSign}(\mathsf{pk}, \tau_1, (\omega_j, \Sigma_j)_{j=1}^d)$
Pick new handle h
$\mathcal{H} \leftarrow \mathcal{H} \cup \{(h, (\tau_1, \Sigma))\}$
Return h

$\underline{\mathcal{O}\mathsf{FoD}(\tau, (\mathbf{M}_j, \omega_j)_{j=1}^d)}$:
$\Sigma_0^* \leftarrow \mathsf{Sign}(\mathsf{sk}, \tau, \sum_{j=1}^d \omega_j \cdot \mathbf{M}_j)$
$(\Sigma_j \leftarrow \mathsf{Sign}(\mathsf{sk}, \tau, \mathbf{M}_j))_{j=1}^d$
$\Sigma_1^* \leftarrow \mathsf{DeriveSign}(\tau, (\omega_j, \Sigma_j)_{j=1}^d)$
Return $(\Sigma_b^*, (\Sigma_j)_{j=1}^d)$

$\underline{\mathsf{Finalize}(b')}$: Return $(b = b')$

Fig. 12. Security game $\mathbf{Exp}^{\mathrm{ch}}_{\Sigma,A}(1^\lambda)$ for context hiding as proposed by Ahn *et al.*

scalars $\omega_1, \omega_2 \in \mathbb{Z}_p^*$ such that $\omega_1 \cdot \mathbf{T}_1 + \omega_2 \cdot \mathbf{T}_2$ is again a Diffie-Hellman tuple. Thus, only signatures under the same tag can be efficiently combined to derive new signatures.

However, the HPP construction is not zero-signable. To achieve this property, we remove the group element H from the tag and replace it with a random element chosen during the signing process, i.e. H is now part of the signature and the new tags consist only of a scalar in $\mathbb{Z}_p^*$. Furthermore, for $i \in [3]$ and some fixed generator $G \in \mathbb{G}^*$, we add signatures σ_i' of $\mathbf{e}_{n+i}^{(n+3)} \cdot G$ to the public keys, where $\mathbf{e}_{n+i}^{(n+3)}$ denotes the $(n+i)$-th standard unit vector in $\mathbb{Z}_p^{n+3}$. Using the $\mathsf{DeriveSign}'$ algorithm, it is then possible to obtain signatures of any Square Diffie-Hellman tuple and, thus, to sign the zero vector under any tag $\in \mathbb{Z}_p^*$. Intuitively, the modified scheme is still unforgeable since any vector in $\mathsf{span}\{\mathbf{e}_{n+i}^{(n+3)} \cdot G : i \in [3]\} \subseteq \mathcal{M}'$ corresponds to the zero vector in $\mathcal{M}$.

Moreover, we add randomization to the $\mathsf{DeriveSign}$ algorithm so that $\mathbf{\Sigma}$ remains context hiding if $\mathbf{\Sigma}'$ already is. Note that the HPP conversion also has this property since the singing procedure is deterministic (except possibly the Sign' subroutine). However, our new signing algorithm chooses the basis H of the appended Square Diffie-Hellman tuple $(H, \tau \cdot H, \tau^2 \cdot H)$ at random, and we need to simulate the same in the signature derivation.

Also, the scheme $\mathbf{\Sigma}$ is universal. If one chooses $H = 0$ during the signing procedure, then the returned signature is independent of the tag, i.e. it could be output by the signing algorithm on input any tag $\tau \in \mathbb{Z}_p^*$.

Correctness of the scheme is easy to check. The unforgeability is proven in the following theorem.

Theorem 30. *If $\mathbf{\Sigma}'$ is OT-LH-Sig, then $\mathbf{\Sigma}$ is LH-Sig against algebraic adversaries under the* DL *assumption in* $\mathbb{G}$.

For the proof, we need the following lemma.

Lemma 31. *Given random generators $G_1, \ldots, G_d \xleftarrow{\$} \mathbb{G}^*$, it is computationally hard under the* DL *assumption to output scalars $\omega_1, \ldots, \omega_d \in \mathbb{Z}_p$ and $\tau_1, \ldots, \tau_d \in \mathbb{Z}_p^*$ with the following properties:*

- $(\sum_{j=1}^d \omega_j \cdot G_j, \sum_{j=1}^d \omega_j \tau_j \cdot G_j, \sum_{j=1}^d \omega_j \tau_j^2 \cdot G_j)$ *is a Square Diffie-Hellman tuple.*
- *There exist indices $i, j \in [d]$ such that $\omega_i, \omega_j \neq 0$ and $\tau_i \neq \tau_j$.*

A proof of Lemma 31 is provided in Appendix B.2.

$\underline{\mathsf{Setup}(1^\lambda)}$:
$\mathsf{pp}' \leftarrow \mathsf{Setup}'(1^\lambda)$; choose any $G \in \mathbb{G}^*$
Return $\mathsf{pp} \leftarrow (\mathsf{pp}', G)$

$\underline{\mathsf{KeyGen}(1^\lambda)}$:
$(\mathsf{sk}', \mathsf{pk}') \leftarrow \mathsf{KeyGen}'(\mathsf{pp}')$
For $i \in [3]$, $\sigma'_i \leftarrow \mathsf{Sign}'(\mathsf{sk}, \mathbf{e}_{n+i}^{(n+3)} \cdot G)$
$\mathsf{sk} \leftarrow \mathsf{sk}'$; $\mathsf{pk} \leftarrow (\mathsf{pk}', (\sigma'_i)_{i=1}^3)$
Return $(\mathsf{sk}, \mathsf{pk})$

$\underline{\mathsf{Sign}(\mathsf{sk}, \tau, \mathbf{M})}$:
$H \xleftarrow{\$} \mathbb{G}$; $\mathbf{M}' \leftarrow (\mathbf{M} \parallel (H, \tau \cdot H, \tau^2 \cdot H))$
$\sigma \leftarrow \mathsf{Sign}'(\mathsf{sk}, \mathbf{M}')$
Return $\Sigma \leftarrow (\sigma, H)$

$\underline{\mathsf{USign}(\mathsf{sk}, \mathbf{M})}$:
$\sigma \leftarrow \mathsf{Sign}'(\mathsf{sk}, (\mathbf{M} \parallel (0,0,0)))$
Return $\Sigma \leftarrow (\sigma, 0)$

$\underline{\mathsf{ZSign}(\mathsf{pk}, \tau)}$:
$\sigma \leftarrow \mathsf{DeriveSign}'(\mathsf{pk}', (\tau^{i-1}, \sigma'_i)_{i=1}^3)$
Return $\Sigma \leftarrow (\sigma, G)$

$\underline{\mathsf{DeriveSign}(\mathsf{pk}, \tau, (\omega_j, \Sigma_j)_{j=1}^d)}$:
For $j \in [d]$, parse $\Sigma_j = (\sigma_j, H_j)$
$\Sigma_0 = (\sigma_0, H_0) \leftarrow \mathsf{ZSign}(\mathsf{pk}, \tau)$
$\omega_0 \xleftarrow{\$} \mathbb{Z}_p^*$
$\sigma \leftarrow \mathsf{DeriveSign}'(\mathsf{pk}', (\omega_j, \sigma_j)_{j=0}^d)$
$H \leftarrow \sum_{j=0}^d \omega_j \cdot H_j$
Return $\Sigma \leftarrow (\sigma, H)$

$\underline{\mathsf{Verify}(\mathsf{pk}, \tau, \mathbf{M}, \Sigma)}$:
Parse $\Sigma = (\sigma, H)$
$\mathbf{M}' \leftarrow (\mathbf{M} \parallel (H, \tau \cdot H, \tau^2 \cdot H))$
Return $b \leftarrow \mathsf{Verify}'(\sigma, \mathbf{M}')$

Fig. 13. Generic conversion

Proof (of Theorem 30). The simulation runs the experiment $\mathbf{Exp}^{\mathrm{unf}}_{\Sigma,A}(1^\lambda)$ honestly. In addition to tags and messages, our implementation of the oracle $\mathcal{O}\mathsf{Reveal}$ adds also the returned signatures to $\mathcal{S}$, i.e. $\mathcal{S}$ is of the form $(\tau_j, \mathbf{M}_j, \Sigma_j = (\sigma_j, H_j))_{j=1}^d$. We denote by $\mathbf{M}'_j$ the vector $\mathbf{M}_j$ extended with the three additional components $(H_j, A_j = \tau_j \cdot H_j, B_j = \tau_j^2 \cdot H_j)$.

Upon receiving the forgery $(\tau, \mathbf{M}, \Sigma = (\sigma, H))$, the challenger checks that $\mathsf{Verify}(\mathsf{pk}, \tau, \mathbf{M}, \Sigma) = 1$. Then the unforgeability of $\mathbf{\Sigma}'$ implies that there exist coefficients $(\omega_j)_{j=1}^d$ such that $\mathbf{M}' = \sum_{j=1}^d \omega_j \cdot \mathbf{M}'_j$. These coefficients are provided by the (algebraic) adversary. We consider the last three components of $\mathbf{M}'$ which form the Square Diffie-Hellman tuple $(H = \sum_{j=1}^d \omega_j \cdot H_j, \tau \cdot H = \sum_{j=1}^d \omega_j \cdot A_j, \tau^2 \cdot H = \sum_{j=1}^d \omega_j \cdot B_j)$. Then Lemma 31 implies that there exists at most one $\tau' \in \mathcal{T}$ with the property that there exist $j \in [d]$ such that $\tau_j = \tau'$ and $\omega_j \neq 0$. Thus, $\mathbf{M}$ is a linear combination of messages $\mathbf{M}_j$ all signed under the same tag $\tau_j = \tau$, i.e. for all $j \in [d]$, if $\tau_j \neq \tau$, then $\omega_j = 0$. □

Moreover, the scheme preserves the context-hiding property.

Theorem 32. *If $\mathbf{\Sigma}'$ is perfectly (resp. statistically, computationally) context hiding, then so is $\mathbf{\Sigma}$.*

Proof. Recall that signatures $\Sigma = (\sigma, H)$ in $\mathbf{\Sigma}$ consist of a $\mathbf{\Sigma}'$-signature σ and a group element H. H is always a uniformly random element of $\mathbb{G}$, independently of the fact whether Σ was output by Sign or $\mathsf{DeriveSign}$. Thus, an adversary can only hope to distinguish the distribution of σ which implies that $\mathbf{\Sigma}$ preserves the same level of context hiding that $\mathbf{\Sigma}'$ has. □

HPP OT-LH-Sig. For completeness, we recall the HPP $\mathsf{OT\text{-}LH\text{-}Sig}$ scheme which can be used to instantiate the generic conversion in Fig. 13. Let $\mathsf{BLinMapGen}(1^\lambda)$ be a PPT algorithm that takes as input a security parameter λ, and returns a description of an asymmetric bilinear map $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, G, \mathfrak{G}, e)$. Then the HPP $\mathsf{OT\text{-}LH\text{-}Sig}$ scheme for the message space $\mathbb{G}_1^n$ is defined as depicted in Fig 14. Correctness of the scheme is easy to check. Unforgeability is proven in the generic group model [HPP20, Theorem 10]. Moreover, the scheme is perfectly context hiding since for each message $\mathbf{M} \in \mathbb{G}_1^n$, there exists only one valid signature.

Alternatively, one could use the $\mathsf{OT\text{-}LH\text{-}Sig}$ scheme of [LPJY13, Section 3.1] to instantiate Fig. 13. The scheme is proven secure in the standard model instead of only the generic group model. However, it is slightly more complex, which is why we do not recall it here.

$\underline{\mathsf{Setup}(1^\lambda)}$:
Return $\mathsf{pp} \leftarrow \mathsf{BLinMapGen}(1^\lambda)$

$\underline{\mathsf{KeyGen}(\mathsf{pp} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, G, \mathfrak{G}, e)}$:
$\mathbf{s} \xleftarrow{\$} \mathbb{Z}_p^n$; $\mathsf{sk} \leftarrow \mathbf{s}$; $\mathsf{pk} \leftarrow \mathbf{s} \cdot \mathfrak{G}$
Return $(\mathsf{sk}, \mathsf{pk})$

$\underline{\mathsf{Sign}(\mathsf{sk} = (s_i)_{i=1}^n, \mathbf{M} = (M_i)_{i=1}^n)}$:
Return $\sigma \leftarrow \sum_{i=1}^n s_i \cdot M_i$

$\underline{\mathsf{DeriveSign}(\mathsf{pk}, (\omega_j, \sigma_j)_{j=1}^d)}$:
Return $\sigma \leftarrow \sum_{j=1}^d \omega_j \cdot \sigma_j$

$\underline{\mathsf{Verify}(\mathsf{pk} = (\mathfrak{G}_i)_{i=1}^n, \mathbf{M} = (M_i)_{i=1}^n, \sigma)}$:
If $e(\sigma, \mathfrak{G}) = \prod_{i=1}^n e(M_i, \mathfrak{G}_i)$, return 1
Return 0

Fig. 14. HPP OT-LH-Sig scheme

B Deferred Proofs

B.1 Proof of Lemma 14

Lemma 14. *Let $(\mathbb{G}, +)$ be a group of prime order p, $s \in \mathbb{Z}_p^*$ and $n > 2$. Given n tuples (G_i, s_i), for $G_i \xleftarrow{\$} \mathbb{G}^*$ and $s_i \xleftarrow{\$} \mathbb{Z}_p^*$, it is computationally hard under the* DL *assumption to output $(\omega_1, \ldots, \omega_n) \in \mathbb{Z}_p^n$ such that $\omega_i \neq 0$ for at least two $i \in [n]$ and*

$$s \cdot \sum_{i=1}^n \omega_i \cdot G_i = \sum_{i=1}^n \omega_i s_i \cdot G_i.$$

Proof. Up to a guess, which is correct with probability at least $1/n$, we can assume that $\omega_1 \neq 0$. On input a discrete logarithm challenge X in basis G, we set $G_1 = X$ and randomly choose $\beta_i \in \mathbb{Z}_p^*$, for $i \in [2; n]$, to set $G_i = \beta_i \cdot G$. Furthermore, for $i \in [n]$, we randomly choose $s_i \xleftarrow{\$} \mathbb{Z}_p^*$ and output (G_i, s_i). The adversary returns $(\omega_1, \ldots, \omega_n)$ such that $s \cdot \sum_{i=1}^n \omega_i \cdot G_i = \sum_{i=1}^n \omega_i s_i \cdot G_i$. We thus have the following relation with an unknown x:

$$s \left(\omega_1 x + \sum_{i=2}^n \omega_i \beta_i \right) = \omega_1 s_1 x + \sum_{i=2}^n \omega_i s_i \beta_i$$

Denoting $u = \sum_{i=2}^n \omega_i \beta_i$ and $v = \sum_{i=2}^n \omega_i s_i \beta_i$, that can be computed, we obtain that $(s - s_1)\omega_1 x + su - v = 0$. We have $s \neq s_1$ with overwhelming probability and $\omega_1 \neq 0$ by assumption. Therefore, we can compute $x = (v - su)/((s - s_1)\omega_1)$. □

B.2 Proof of Lemma 31

Lemma 31. *Given random generators $G_1, \ldots, G_d \xleftarrow{\$} \mathbb{G}^*$, it is computationally hard under the* DL *assumption to output scalars $\omega_1, \ldots, \omega_d \in \mathbb{Z}_p$ and $\tau_1, \ldots, \tau_d \in \mathbb{Z}_p^*$ with the following properties:*

- *$(\sum_{j=1}^d \omega_j \cdot G_j, \sum_{j=1}^d \omega_j \tau_j \cdot G_j, \sum_{j=1}^d \omega_j \tau_j^2 \cdot G_j)$ is a Square Diffie-Hellman tuple.*
- *There exist indices $i, j \in [d]$ such that $\omega_i, \omega_j \neq 0$ and $\tau_i \neq \tau_j$.*

Proof. Up to a guess, which is correct with probability at least $1/n^2$, we can assume that $\omega_1, \omega_2 \neq 0$ and $\tau_1 \neq \tau_2$. On input a discrete logarithm challenge X in basis G, we sample a random bit $b \xleftarrow{\$} \{0, 1\}$ and choose $G_1, \ldots, G_d$ as follows:

- If $b = 0$, then sample $(\alpha_2, \ldots, \alpha_d) \xleftarrow{\$} (\mathbb{Z}_p^*)^{d-1}$ and set $(G_1 = X, G_2 = \alpha_2 \cdot G, \ldots, G_d = \alpha_d \cdot G)$, i.e. $G_1 = \alpha_1 \cdot G$ for some unknown $\alpha_1 \in \mathbb{Z}_p$.
- If $b = 1$, then sample $(\alpha_1, \alpha_3, \ldots, \alpha_d) \xleftarrow{\$} (\mathbb{Z}_p^*)^{d-1}$ and set $(G_1 = \alpha_1 \cdot G, G_2 = X, G_3 = \alpha_3 \cdot G, \ldots, G_d = \alpha_d \cdot G)$, i.e. $G_2 = \alpha_2 \cdot G$ for some unknown $\alpha_2 \in \mathbb{Z}_p$.

Upon receiving $G_1, \ldots, G_d$, the adversary outputs scalars $\omega_1, \ldots, \omega_d \in \mathbb{Z}_p$ and $\tau_1, \ldots, \tau_d \in \mathbb{Z}_p^*$ such that $(\sum_{j=1}^d \omega_j \cdot G_j = H, \sum_{j=1}^d \omega_j \tau_j \cdot G_j = \tau \cdot H, \sum_{j=1}^d \omega_j \tau_j^2 \cdot G_j = \tau^2 \cdot H)$ for some $H \in \mathbb{G}$ and an unknown $\tau \in \mathbb{Z}_p$. We thus have the following relations:

$$\left(\sum_{j=1}^{d} \alpha_j \omega_j\right) \cdot \tau = \sum_{j=1}^{d} \alpha_j \omega_j \tau_j \qquad \left(\sum_{j=1}^{d} \alpha_j \omega_j \tau_j\right) \cdot \tau = \sum_{j=1}^{d} \alpha_j \omega_j \tau_j^2$$

Let $u = \sum_{j=3}^d \alpha_j \omega_j$, $v = \sum_{j=3}^d \alpha_j \omega_j \tau_j$ and $w = \sum_{j=3}^d \alpha_j \omega_j \tau_j^2$. Combining the above equalities yields that

$$(\alpha_1\omega_1\tau_1 + \alpha_2\omega_2\tau_2 + v)^2 = (\alpha_1\omega_1 + \alpha_2\omega_2 + u)(\alpha_1\omega_1\tau_1^2 + \alpha_2\omega_2\tau_2^2 + w).$$

The latter implies in turn that

$$\begin{aligned}\alpha_1\alpha_2\omega_1\omega_2(\tau_1 - \tau_2)^2 + \alpha_1\omega_1(w - 2v\tau_1 + u\tau_1^2) + \\ \alpha_2\omega_2(w - 2v\tau_2 + u\tau_2^2) + (uw - v^2) = 0.\end{aligned}$$

Let $t = \alpha_2\omega_2(\tau_1 - \tau_2)^2 + w - 2v\tau_1 + u\tau_1^2$. One can efficiently determine whether $t = 0 \mod p$ by computing u, v and w and checking whether the equality $-(w-2v\tau_1+u\tau_1^2)/(\omega_2(\tau_1-\tau_2)^2)\cdot G = G_2$ holds. We consider two cases.

- If $t = 0 \mod p$, but $b = 0$, then the simulation aborts. Otherwise, if $b = 1$, then it returns $\alpha_2 = -(w - 2v\tau_1 + u\tau_1^2)/(\omega_2(\tau_1 - \tau_2)^2) \mod p$. Note that the event "$t = 0 \mod p$" and the bit b are independent. So the simulation returns the correct value with probability 1/2.
- Conversely, if $t \neq 0 \mod p$, but $b = 1$, then the simulation aborts. If $b = 0$, then α_2 is known and one can compute $\alpha_1 = -(\alpha_2\omega_2(w - 2v\tau_2 + u\tau_2^2) + (uw - v^2))/(\omega_1 t) \mod p$.

□