Linearly-Homomorphic Signatures and Scalable Mix-Nets

Chloé Hébant[1,2], Duong Hieu Phan[3], and David Pointcheval[1,2]

[1] DIENS, École normale supérieure, CNRS, PSL University, Paris, France
[2] INRIA, Paris, France
[3] Université de Limoges, France

Abstract Anonymity is a primary ingredient for our digital life. Several tools have been designed to address it such as, for authentication, blind signatures, group signatures or anonymous credentials and, for confidentiality, randomizable encryption or mix-nets. When it comes to complex electronic voting schemes, random shuffling of authenticated ciphertexts with mix-nets is the only known tool. However, it requires huge and complex zero-knowledge proofs to guarantee the actual permutation of the initial ciphertexts in a privacy-preserving way.

In this paper, we propose a new approach for proving correct shuffling: the mix-servers can simply randomize individual ballots, which means the ciphertexts, the signatures, and the verification keys, with an additional global proof of constant size, and the output will be publicly verifiable. The security proof is in the generic bilinear group model. The computational complexity for the each mix-server is linear in the number of ballots. Verification is also linear in the number of ballots, but independent of the number of rounds of mixing. This leads to a new highly scalable technique. Our construction makes use of linearly-homomorphic signatures, with new features, that are of independent interest.

Keywords: Anonymity, random shuffling, linearly-homomorphic signatures

1 Introduction

A shuffle of ciphertexts is a set of ciphertexts of the same plaintexts but in a permuted order such that it is not possible to trace back the senders after decryption. It can be used as a building block to anonymously send messages: if several servers perform a shuffle successively, nobody can trace the messages. More precisely, one honest mix-server suffices to mask the order of the ciphertexts even if all the other ones are dishonest. Moreover increasing the number of mix-servers leads to a safer protocol but also increases its cost. The succession of shuffles constitutes the notion of a mix-net protocol introduced by Chaum [Cha81], with applications to anonymous emails, anonymous routing, but also e-voting.

1.1 State of the Art

Usually, a shuffle of ciphertexts is a permutation applied to randomized ciphertexts. Randomization of the ciphertexts provides the privacy guarantee, but one additionally needs to prove the permutation property. This last step requires huge and complex zero-knowledge proofs. In the main two techniques, Furukawa and Sako [FS01] make proofs of permutation matrices and Neff [Nef01] considers polynomials which remain identical with a permutation of the roots. While the latter approach produces the most efficient schemes, they need to be interactive. Groth and Ishai [GI08] exploited this interactive approach and proposed the first zero-knowledge argument for the correctness of a shuffle with sub-linear communication complexity, but computational complexity is super-linear which was then improved by Bayer and Groth [BG12]. As this is a public random coin interactive Zero-Knowledge protocol, the Fiat-Shamir heuristic [FS87] can be applied to make it non-interactive in the random oracle model. However, with multiple mixing steps, which are required if one wants to guarantee anonymity even if some mix-servers are malicious, the final proof is linear in this number of steps, and the verification cost becomes prohibitive.

The former approach with proof of permutation matrix is more classical, with many candidates. Groth and Lu [GL07] proposed the first non-interactive zero-knowledge (NIZK) proof of shuffle without random oracles, using Groth-Sahai proofs with pairings [GS08], but under non-standard computational assumptions that hold in the generic bilinear group model. Even with that, computations are still very expansive because the overhead proof is linear in Nn, where n is the number of ciphertexts and N the number of mixing rounds. In addition, they needed a Common Reference String (CRS) linear in n. More recently, Fauzi *et al.* [FLSZ17] proposed a new pairing-based NIZK shuffle argument to improve the computation for both the prover and the verifier, and improved the soundness of the protocol. But they still had a CRS linear in the number of ciphertexts, and the soundness holds in the generic bilinear group model.

We propose a totally new approach that handles each ciphertext in an independent way, with just a constant-size overhead in the final proof. The overhead after each shuffle can indeed be updated to keep it constant-size. From our knowledge, this is the most scalable solution. It relies on Groth-Sahai proofs with pairings [GS08] and under a new computational assumption that holds in the generic bilinear group model. As a consequence, assumptions are quite similar to [GL07], but we have a constant-size CRS and a constant-size overhead proof.

Compared to the most efficient schemes to date, namely the Fauzi *et al.*'s scheme [FLSZ17], our scheme is also proven in the generic bilinear group model, but the CRS is shorter: just 8 group elements in contrast to a CRS with a number of group elements linear in the number of ballots. Moreover, in our scheme, the proof is constant-size, independently of the number of mixing rounds, while the proof of Fauzi *et al.*'s scheme grows linearly in the number of rounds. Hence, from 2 rounds, our scheme has a better verifier's computation cost and for 3 rounds the proof sizes are almost the same with the two schemes. With more rounds, our construction gets much better compared to the Fauzi *et al.*'s scheme, and the input ballots already contain signatures by their senders, which makes it quite attractive for electronic voting.

1.2 Our Approach

In our shuffle, each ciphertext C_i (encrypted vote in the ballot, in the context of electronic voting) is signed by its sender and the mix-server randomizes the ciphertexts $\{C_i\}$ and permutes them into the set $\{C'_i\}$ in a provable way. The goal of the proof is to show the existence of a permutation Π from $\{C_i\}$ to $\{C'_i\}$ such that for every i, $C'_{\Pi(i)}$ is a randomization of C_i. Then, the output ciphertexts can be mixed again by another mix-server.

Our approach avoids the proof of an explicit permutation Π on all the ciphertexts (per mixing step) but still guarantees the appropriate properties deeply using the linearly-homomorphic signature schemes:

- each user is associated to a signing/verification key-pair for a linearly-homomorphic signature scheme [BFKW09], and uses it to sign his ciphertext and a way to randomize it. This guarantees that the mix-server will only be able to generate new signatures on randomized ciphertexts, which are unlinkable to the original ciphertexts, due to the new random coins. However, unchanged verification keys would still allow linkability;
- each verification key of the users is thus also certified with a linearly-homomorphic signature scheme, that allows randomization too as well as adaptation of the above signature on the ciphertext, and provides unlinkability.

When talking about linearly-homomorphic signature schemes, we consider signatures that are malleable and that allow to sign any linear combination of the already signed vectors [BFKW09]. In order to be able to use this property on the latter scheme that signs the verification keys of the former scheme, it will additionally require some homomorphic property on the keys.

However, whereas ciphertexts are signed under different keys, which excludes combinations, the verification keys are all signed under the authority's key. Furthermore, a linearly-homomorphic signature scheme not only allows multiplication by a constant, but also linear combinations, which would allow combinations of keys and thus, possibly, of ballots. In order to avoid such combinations, we require a tag-based signature, that allows only linear combinations between signatures using the same tag. As such signatures allow to derive a signature of any message in the sub-vector space spanned by the initially signed messages, when there is no tag, only one sub-vector space can be considered, whereas tags allow to deal with multiple sub-vector space. In the latter case, one thus talks about *Linearly-Homomorphic Signature* ($\mathsf{LH\text{-}Sign}$), whereas the former case is named *One-Time Linearly-Homomorphic Signature* ($\mathsf{OT\text{-}LH\text{-}Sign}$).

In the appendix, we provide a generic conversion from $\mathsf{OT\text{-}LH\text{-}Sign}$ to $\mathsf{LH\text{-}Sign}$, using Square Diffie-Hellman tuples $(g, g^{w_i}, g^{w_i^2})$ for the tags. So, starting from an efficient $\mathsf{OT\text{-}LH\text{-}Sign}$, one can derive all the tools needed for our mix-net application. However, in the body of the paper, we also provide a more efficient $\mathsf{LH\text{-}Sign}$ version, and we thus focus on it in the following.

Unforgeability of the signature schemes will essentially provide the soundness of the proof of correct mixing: only permutations of ballots are possible. Eventually, unlinkability (a.k.a. zero-knowledge property) will be satisfied thanks to the randomizations that are indistinguishable for various users, under some DDH-like assumptions, and the final random permutation of all the ciphertexts. With the above linear homomorphisms of the signatures, we can indeed guarantee that the output C'_j is a randomization of an input C_i, and the verification keys are unlinkable.

More precisely, the signature unforgeability will guarantee that all the ballots in the output ballot-box come from legitimate signers: we will also have to make sure that there is no duplicates, nor new ballots, and the same numbers of ballots in the input ballot-box and output ballot-box for the formal proof of permutation.

This technique of randomizing ciphertexts and verification keys, and adapting signatures, can be seen as an extension of signatures on randomizable ciphertexts [BFPV11] which however did not allow updates of the verification keys. This previous approach excluded anonymity because of the invariant verification keys. Our new approach can find more applications where anonymity and privacy are crucial properties.

1.3 Organization

In the next section, we recall some usual assumptions in pairing-based groups, and we introduce a new *unlinkability assumption* that will be one of the core assumptions of our applications. Note that it holds in the generic bilinear group model. In Section 3, we recall the notion of linearly-homomorphic signatures, with a construction of a one-time linearly-homomorphic signature scheme and its security analysis in the generic bilinear group model. Then we extend it to handle multiple sub-vector spaces. We then apply these constructions to mix-networks in Section 4, followed by a detailed security analysis in Section 5. Eventually, we conclude with some applications in Section 6.

2 Computational Assumptions

In this section, we will first recall some classical computational assumptions and introduce a new one, of independent interest, as it can find many use cases for privacy-preserving protocols.

2.1 Classical Assumptions

All our assumptions will be in the Diffie-Hellman vein, in the pairing setting. We will thus consider an algorithm that, on a security parameter κ, generates $\mathsf{param} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, \mathfrak{g}, e) \leftarrow \mathcal{G}(\kappa)$, an asymmetric pairing setting, with three groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ of prime order p (with 2κ bit-length), g is a generator of $\mathbb{G}_1$ and $\mathfrak{g}$ is a generator of $\mathbb{G}_2$. In addition, the application $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is a non-degenerated bilinear map, hence $e(g, \mathfrak{g})$ is also a generator of $\mathbb{G}_T$. For the sake of clarity, in all the paper, elements of $\mathbb{G}_2$ will be in Fraktur font.

Definition 1 (Discrete Logarithm (DL) Assumption). In a group $\mathbb{G}$ of prime order p, it states that for any generator g, given $y = g^x$, it is computationally hard to recover x.

Definition 2 (Twin Discrete Logarithm (TDL) Assumption). In groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of prime order p, it states that for any generators g and $\mathfrak{g}$ of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively, given $f = g^x$ and $\mathfrak{f} = \mathfrak{g}^x$, it is computationally hard to recover x.

Definition 3 (Decisional Diffie-Hellman (DDH) Assumption). In a group $\mathbb{G}$ of prime order p, it states that for any generator g, the two following distributions are computationally indistinguishable:

$$\mathcal{D}_{\mathsf{dh}}(g) = \{(g, g^x, h, h^x); h \overset{\$}{\leftarrow} \mathbb{G}, x, \overset{\$}{\leftarrow} \mathbb{Z}_p\}$$
$$\mathcal{D}_{\$}^4(g) = \{(g, g^x, h, h^y); h \overset{\$}{\leftarrow} \mathbb{G}, x, y, \overset{\$}{\leftarrow} \mathbb{Z}_p\}.$$

This is well-know, using an hybrid argument, or the random-self-reducibility, that this assumption implies the Decisional Multi Diffie-Hellman (DMDH) Assumption, which claims the indistinguishability, for any constant $n \in \mathbb{N}$, of the distributions:

$$\mathcal{D}_{\mathsf{mdh}}^n(g) = \{(g, (g^{x_i})_i, h, (h^{x_i})_i); h \overset{\$}{\leftarrow} \mathbb{G}, (x_i)_i \overset{\$}{\leftarrow} \mathbb{Z}_p^n\}$$
$$\mathcal{D}_{\$}^{2n+2}(g) = \{(g, (g^{x_i})_i, h, (h^{y_i})_i); h \overset{\$}{\leftarrow} \mathbb{G}, (x_i)_i, (y_i)_i \overset{\$}{\leftarrow} \mathbb{Z}_p^n\}.$$

2.2 Unlinkability Assumption

For anonymity properties, we will use some kind of credential, that can be defined as follows for a scalar u and a basis $g \in \mathbb{G}_1$, with $\mathfrak{g} \in \mathbb{G}_2$, $r, t \in \mathbb{Z}_p$:

$$\mathsf{Cred}(u, g; \mathfrak{g}, r, t) = \left(g, g^t, g^r, g^{tr+u}, \mathfrak{g}, \mathfrak{g}^t, \mathfrak{g}^u\right)$$

Definition 4 (Unlinkability Assumption). In groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of prime order p, for any $g \in \mathbb{G}_1$ and $\mathfrak{g} \in \mathbb{G}_2$, with the definition below, it states that the distributions $\mathcal{D}_{g,\mathfrak{g}}(u, u)$ and $\mathcal{D}_{g,\mathfrak{g}}(u, v)$ are computationally indistinguishable, for any $u, v \in \mathbb{Z}_p$:

$$\mathcal{D}_{g,\mathfrak{g}}(u, v) = \left\{(\mathsf{Cred}(u, g; \mathfrak{g}, r, t), \mathsf{Cred}(v, g; \mathfrak{g}', r', t')); \begin{array}{r} \mathfrak{g}' \overset{\$}{\leftarrow} \mathbb{G}_2, \\ r, t, r', t' \overset{\$}{\leftarrow} \mathbb{Z}_p \end{array}\right\}$$

Intuitively, as we can write the credential as, where × stands for the element-wise product,

$$\mathsf{Cred}(u, g; \mathfrak{g}, r, t) = \left(\begin{pmatrix} g \\ \mathfrak{g} \end{pmatrix}, \begin{pmatrix} g \\ \mathfrak{g} \end{pmatrix}^t, \begin{pmatrix} g \\ g^t \end{pmatrix}^r \times \begin{pmatrix} 1 \\ g^u \end{pmatrix}, \mathfrak{g}^u\right)$$

the third component is an ElGamal ciphertext of the g^u, which hides it, and makes indistinguishable another encryption g^u from an encryption of g^v while, given $(\mathfrak{g}, \mathfrak{g}^u)$ and $(\mathfrak{g}', \mathfrak{g}'^v)$, one cannot guess whether $u = v$, under the DDH assumption in $\mathbb{G}_2$. However the pairing relation allows to check consistency:

$$e(g^{rt+u}, \mathfrak{g}) = e(g^r, \mathfrak{g}^t) \cdot e(g, \mathfrak{g}^u) = e(g^r, \mathfrak{g}^t) \cdot e(g, \mathfrak{g})^u$$
$$e(g^{r't'+v}, \mathfrak{g}') = e(g^{r'}, \mathfrak{g}'^{t'}) \cdot e(g, \mathfrak{g}'^v) = e(g^{r'}, \mathfrak{g}'^{t'}) \cdot e(g, \mathfrak{g}')^v$$

Because of the independent group elements $\mathfrak{g}$ and $\mathfrak{g}' = \mathfrak{g}^s$ in the two credentials, this assumption clearly holds in the generic bilinear group model, as one would either need to compare $u = v$ or equivalently $rt = r't'$, whereas combinations only lead to $e(g, \mathfrak{g})$ to the relevant powers rt, $sr't'$, as well as u and sv, for an unknown s.

Thanks to this unlinkability assumption, and the randomizability of the above credential, proving knowledge of u can lead to anonymous credentials. However, our main application will be for our anonymous shuffles presented in Section 4.

3 Linearly-Homomorphic Signatures

The notion of homomorphic signatures dates back to [JMSW02], with notions in [ABC+12], but the linearly-homomorphic signatures, that allow to sign vector sub-spaces, were introduced in [BFKW09], with several follow-up by Boneh and Freeman [BF11b, BF11a] and formal security definitions in [Fre12]. In another direction, Abe *et al.* [AFG+10] proposed the notion of structure-preserving signature, where keys, messages and signatures all belong in the same group. Later Libert *et al.* [LPJY13] combined both notions and proposed a linearly-homomorphic signature scheme, that is furthermore structure-preserving. Our work is inspired from this construction, but in the asymmetric-pairing setting, and keys do not belong to the same group as the message and signatures. The *structure-preserving* property is then relaxed but fits our needs, as we will use two layers of linearly-homomorphic signature schemes, with swapped groups for the keys and the messages.

3.1 Definition and Security

In this first part, we begin with the formal definition of linearly-homomorphic signature scheme, and the security requirement, the so-called *unforgeability* in case of signatures. Then, we will introduce a new property for linearly-homomorphic signature scheme: the randomizable tag. It will be the key element to obtain the privacy in our mix-net. Our definition is inspired from [LPJY13], but with a possible private key associated to a tag.

Definition 5 (Linearly-Homomorphic Signature Scheme ($\mathsf{LH\text{-}Sign}$)). A linearly-homomorphic signature scheme with messages in $\mathcal{M} \in \mathbb{G}^n$, for a cyclic group $(\mathbb{G}, \times)$ of prime order p, some $n \in \mathsf{poly}(\kappa)$, and some tag set $\mathcal{T}$, consists of the seven algorithms $(\mathsf{Setup}, \mathsf{Keygen}, \mathsf{NewTag}, \mathsf{VerifTag}, \mathsf{Sign}, \mathsf{DerivSign}, \mathsf{Verif})$:

$\mathsf{Setup}(1^\kappa)$**:** Given a security parameter κ, it outputs the global parameter param, which includes the tag space $\mathcal{T}$;

$\mathsf{Keygen}(\mathsf{param}, n)$**:** Given a public parameter param and an integer n, it outputs a key pair $(\mathsf{sk}, \mathsf{vk})$. We will assume that vk implicitly contains param and sk implicitly contains vk;

$\mathsf{NewTag}(\mathsf{sk})$**:** Given a signing key sk, it outputs a tag τ and its associated secret key $\tilde{\tau}$;

$\mathsf{VerifTag}(\mathsf{vk}, \tau)$**:** Given a verification key vk and a tag τ, it outputs 1 if the tag is valid and 0 otherwise;

$\mathsf{Sign}(\mathsf{sk}, \tilde{\tau}, \boldsymbol{M})$**:** Given a signing key, a secret key tag $\tilde{\tau}$ and a vector-message $\boldsymbol{M} = (M_i)_i \in \mathbb{G}^n$, it outputs the signature σ under the tag τ;

$\mathsf{DerivSign}(\mathsf{vk}, \tau, (\omega_i, \boldsymbol{M}_i, \sigma_i)_{i=1}^{\ell})$**:** Given a public key vk, a tag τ and ℓ tuples of weights $\omega_i \in \mathbb{Z}_p$ and signed messages $\boldsymbol{M}_i$ in σ_i, it outputs a signature σ on the vector $\boldsymbol{M} = \prod_{i=1}^{\ell} \boldsymbol{M}_i^{\omega_i}$ under the tag τ;

$\mathsf{Verif}(\mathsf{vk}, \tau, \boldsymbol{M}, \sigma)$**:** Given a verification key vk, a tag τ, a vector-message $\boldsymbol{M}$ and a signature σ, it outputs 1 if $\mathsf{VerifTag}(\mathsf{vk}, \tau) = 1$ and σ is also valid relative to vk and τ, and 0 otherwise.

The tag in $\mathsf{DerivSign}$ allows linear combinations of signatures under the same tag but excludes any operation between signatures under different tags. The latter exclusion will be formalized by the unforgeability. However, the former property is the correctness: for any keys $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Keygen}(\mathsf{param}, n)$, for any tags $(\tau, \tilde{\tau}) \leftarrow \mathsf{NewTag}(\mathsf{sk})$, if $\sigma_i = \mathsf{Sign}(\mathsf{sk}, \tilde{\tau}, \boldsymbol{M}_i)$ are valid signatures for $i = 1, \dots, \ell$ and $\sigma = \mathsf{DerivSign}(\mathsf{vk}, \tau, \{\omega_i, \boldsymbol{M}_i, \sigma_i\}_{i=1}^{\ell})$ from some scalars ω_i, then both

$$\mathsf{VerifTag}(\mathsf{vk}, \tau) = 1 \qquad\qquad \mathsf{Verif}(\mathsf{vk}, \tau, \boldsymbol{M}, \sigma) = 1.$$

Our definition includes, but is more relaxed than, [LPJY13] as we allow a secret key associated to the tag, hence the NewTag algorithm: in such a case, the signer can only sign a message on a tag he generated himself. When there is no secret associated to the tag, actually one can consider that $\tilde{\tau} = \tau$ is enough to generate the signature (in addition to sk). Whereas the $\mathsf{DerivSign}$ algorithm generates a signature under the same tag, we do not enforce to keep the same tag in the unforgeability notion below, this will allow our tag randomizability. However, we expect only signatures on linear combinations of messages already signed under a same tag, as we formalize in the following security notion.

Unforgeability. Whereas linear combinations are possible under the same tag, other combinations (non-linear or under different tags) should not be possible. This is the unforgeability notion (note that we talk about linear combinations component-wise in the exponents, as we consider a multiplicative group $\mathbb{G}$).

Definition 6 (Unforgeability for $\mathsf{LH\text{-}Sign}$). For a $\mathsf{LH\text{-}Sign}$ scheme with messages in $\mathbb{G}^n$, for any adversary $\mathcal{A}$ that, given tags and signatures on messages $(\boldsymbol{M}_i)_i$ under tags $(\tau_i)_i$ both of its choice (for Chosen-Message Attacks), outputs a valid tuple $(\mathsf{vk}, \tau, \boldsymbol{M}, \sigma)$, then there must exist τ' and $(\omega_i)_{i \in I_{\tau'}}$, where $I_{\tau'}$ is the set of messages signed under the tag τ', such that $\boldsymbol{M} \neq \prod_{i \in I_{\tau'}} \boldsymbol{M}_i^{\omega_i}$, but with negligible probability.

Again, because of our relaxed version compared to [LPJY13], we do not exclude the adversary to be able to generate valid signatures under new tags. The linear-homomorphism for signatures, also known as signatures on vector-spaces, requires that the adversary cannot generate a valid signature on a message outside the vector spaces spanned by the already signed messages. Tags are just a way to keep together vectors that define vector spaces. The adversary can rename a vector space with another tag, this is not a security issue. On the opposite, we will exploit this feature for unlinkability with the additional randomizability property on tags (see below).

However, as in [LPJY13], we will also consider a weaker notion of linearly-homomorphic signature: a one-time linearly-homomorphic signature ($\mathsf{OT\text{-}LH\text{-}Sign}$), where the set of tags is a singleton $\mathcal{T} = \{\epsilon\}$. Then we can drop the algorithms NewTag and $\mathsf{VerifTag}$, as well as the τ and $\tilde{\tau}$.

3.2 Our One-Time Linearly-Homomorphic Signature

Libert *et al.* [LPJY13] proposed a construction whose security relies on the Simultaneous Double Pairing assumption, which is implied by the linear assumption in the symmetric case. In our use case we will need two $\mathsf{LH\text{-}Sign}$ schemes. While the first one can simply be one-time and thus possibly in the standard model, the second one needs randomizable tags and we do not know how to build it in the standard model. Thus, we will consider a variant of Libert *et al.* [LPJY13] that can only be proven in the generic bilinear group model [Sho97, BBG05, Boy08].

$\mathsf{Setup}(1^\kappa)$**:** Given a security parameter κ, let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, \mathfrak{g}, e)$ be an asymmetric bilinear setting, where g and $\mathfrak{g}$ are random generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. We set $\mathsf{param} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, \mathfrak{g}, e)$;

$\mathsf{Keygen}(\mathsf{param}, n)$**:** Given the public parameters param, one randomly chooses $\mathsf{sk}_i = s_i \xleftarrow{\$} \mathbb{Z}_p$, for $i = 1, \ldots, n$, which defines the signing key $\mathsf{sk} = (\mathsf{sk}_i)_{i=1}^n$, and the verification key $\mathsf{vk} = (\mathfrak{g}_i)_{i=0}^n$ for $\mathfrak{g}_i = \mathfrak{g}^{s_i}$ and $\mathfrak{g}_0 = \mathfrak{g}$;

$\mathsf{Sign}(\mathsf{sk}, \boldsymbol{M} = (M_i)_i)$**:** Given a signing key $\mathsf{sk} = (s_i)_i$ and a vector-message $\boldsymbol{M} = (M_i)_i \in \mathbb{G}_1^n$, one sets $\sigma = \prod_{i=1}^n M_i^{s_i} \in \mathbb{G}_1$;

$\mathsf{DerivSign}(\mathsf{vk}, (\omega_i, \boldsymbol{M}_i, \sigma_i)_{i=1}^\ell)$**:** Given a verification key and ℓ tuples of weights $\omega_i \in \mathbb{Z}_p$ and signed messages $\boldsymbol{M}_i$ in σ_i, it outputs $\sigma = \prod \sigma_i^{\omega_i}$;

$\mathsf{Verif}(\mathsf{vk}, \boldsymbol{M} = (M_i)_i, \sigma)$**:** Given a verification key vk, a vector-message $\boldsymbol{M}$, and a signature σ, one checks whether the equality $e(\sigma, \mathfrak{g}_0) = \prod_{i=1}^n e(M_i, \mathfrak{g}_i)$ holds or not.

From this description, the derivation of signatures is trivial as the signature of the product of messages is the product of the signatures. But we also have additional properties with the keys:

Property 7 (Message Homomorphism). Given several vector-messages with their signatures, it is possible to generate the signature of any linear combination of the vector-messages, applying the operation on the signatures.

When the messages are the same, one can ask for similar property on the key:

Property 8 (Key Homomorphism). Given a vector-message with signatures under several keys, it is possible to generate the signature of this vector-message under any linear combination of the keys.

$\mathsf{DerivSignKey}(\boldsymbol{M}, (\omega_i, \mathsf{vk}_i, \sigma_i)_{i=1}^{\ell})$: Given a message $\boldsymbol{M}$ and ℓ tuples of weights $\omega_i \in \mathbb{Z}_p$ and signatures σ_i of $\boldsymbol{M}$ under vk_i, it outputs a signature σ of $\boldsymbol{M}$ under the verification key $\mathsf{vk} = \prod_{i=1}^{\ell} \mathsf{vk}_i^{\omega_i}$.

In our case, if a message-signature is valid for a verification key vk, then it is also valid for the verification key $\mathsf{vk}' = \mathsf{vk}^\alpha$, for any α, as $e(\sigma, \mathfrak{g}_0) = \prod_{i=1}^{n} e(M_i, \mathfrak{g}_i)$ implies $e(\sigma, \mathfrak{g}_0^\alpha) = \prod_{i=1}^{n} e(M_i, \mathfrak{g}_i^\alpha)$. However, for two different verification keys vk and vk', and signatures σ and σ' of $\boldsymbol{M}$: $\prod_{i=1}^{n} e(M_i, \mathfrak{g}_i^\alpha \cdot \mathfrak{g}_i'^\beta) = \prod_{i=1}^{n} e(M_i, \mathfrak{g}_i)^\alpha \cdot e(M_i, \mathfrak{g}_i')^\beta = e(\sigma, \mathfrak{g}_0^\alpha) \cdot e(\sigma', \mathfrak{g}_0'^\beta)$, so $\sigma'' = \sigma^\alpha \sigma'^\beta$ is a valid signature of $\boldsymbol{M}$ under $\mathsf{vk}'' = \mathsf{vk}^\alpha \mathsf{vk}'^\beta$ if $\mathfrak{g}_0' = \mathfrak{g}_0$.

Property 9 (Weak Key Homomorphism). Given a vector-message with signatures under several keys (with a specific restriction, as a common $\mathfrak{g}_0$ in our case), it is possible to generate the signature of this vector-message under any linear combination of the keys.

Eventually, one needs to prove the unforgeability:

Theorem 10 (Unforgeability). *Let us consider an adversary $\mathcal{A}$ in the generic bilinear group model. Given valid pairs $(\boldsymbol{M}_j, \sigma_j)_j$ under a verification key vk ($\boldsymbol{M}_i$'s possibly of adversary's choice, for Chosen-Message Attacks), when $\mathcal{A}$ produces a new valid pair $(\boldsymbol{M}, \sigma)$ under the same verification key vk, there exist $(\alpha_j)_j$ such that $\boldsymbol{M} = \prod_j \boldsymbol{M}_j^{\alpha_j}$.*

Proof. The adversary $\mathcal{A}$ is given $(\boldsymbol{M}_j = (M_{j,i})_i, \sigma_j)_j$ which contains group elements in $\mathbb{G}_1$, as well as the verification key $\mathsf{vk} = (\mathfrak{g}_k)_k$ in $\mathbb{G}_2$. Note that in the generic bilinear group model, programmability of the encoding allows to simulate the signatures for chosen messages, which provides the security against Chosen-Message Attacks.

For any combination query, the simulator will consider the input elements as independent variables $X_{j,i}$, V_j, and $\mathfrak{S}_k$ to formally represent the discrete logarithms of $M_{j,i}$ and σ_i in basis g, and $\mathfrak{g}_k$ in basis $\mathfrak{g}_0 = \mathfrak{g}$. As usual, any new element can be seen as a multivariate polynomial in these variables, of degree maximal 2 (when there is a mix between $\mathbb{G}_1$ and $\mathbb{G}_2$ group elements). If two elements correspond to the same polynomial, they are definitely equal, and the simulator will provide the same representation. If two elements correspond to different polynomials, the simulator will provide random independent representations. The view of the adversary remains unchanged unless the actual instantiations would make the representations equal: they would be equal with probability at most $2/p$, when the variables are set to random values. After N combination queries, we have at most $N^2/2$ pairs of different polynomials that might lead to a collision for a random setting with probability less than N^2/p. Excluding such collisions, we can thus consider the polynomial representations only, denoted $\sim$. Then, for the output $(\boldsymbol{M} = (M_k)_k, \sigma)$, one knows $\alpha_{k,j,i}, \beta_{k,j}, \gamma_{i,j}, \delta_j$, such that:

$$M_k \sim \sum_{j,i} \alpha_{k,j,i} X_{j,i} + \sum_j \beta_{k,j} V_j \qquad\qquad \sigma \sim \sum_{j,i} \gamma_{j,i} X_{j,i} + \sum_j \delta_j V_j.$$

As $((M_{j,i})_i, \sigma_j)_j$ and $((M_k)_k, \sigma)$, are valid input and output pairs, we have the following relations between polynomials:

$$V_j = \sum_i X_{j,i}\mathfrak{S}_i \qquad \sum_{j,i} \gamma_{j,i}X_{j,i} + \sum_j \delta_j V_j = \sum_k \left(\sum_{j,i} \alpha_{k,j,i}X_{j,i} + \sum_j \beta_{k,j}V_j \right) \mathfrak{S}_k$$
$$= \sum_{k,j,i} \alpha_{k,j,i}X_{j,i}\mathfrak{S}_k + \sum_{k,j} \beta_{k,j}V_j\mathfrak{S}_k$$

Hence, the two polynomials are equal:

$$\sum_{j,i} \gamma_{j,i}X_{j,i} + \sum_{j,i}(\delta_j - \alpha_{i,j,i})X_{j,i}\mathfrak{S}_i = \sum_{k \neq i,j,i} \alpha_{k,j,i}X_{j,i}\mathfrak{S}_k + \sum_{k,j} \beta_{k,j}V_j\mathfrak{S}_k$$

which leads, for all i, j, to $\gamma_{j,i} = 0$ and $\delta_j = \alpha_{i,j,i}$, and for $k \neq i$, $\alpha_{k,j,i} = 0$ and $\beta_{k,j} = 0$. Hence, $M_k \sim \sum_j \delta_j X_{j,k}$ and $\sigma \sim \sum_j \delta_j V_j$, which means that we have $(\delta_j)_j$ such that $M_k = \prod_j M_{j,k}^{\delta_j}$ and $\sigma = \prod_j \sigma_j^{\delta_j}$. □

3.3 Notations and Constraints

We recall that linear combinations are seen in the exponents. Since we will mainly work on sub-vector spaces of dimension 2 (in a larger vector space), we will denote $\sigma = \mathsf{Sign}(\mathsf{sk}, (\boldsymbol{M}, \boldsymbol{M'}))$, with the verification check $\mathsf{Verif}(\mathsf{vk}, \sigma, (\boldsymbol{M}, \boldsymbol{M'})) = 1$, a signature that allows to derive a valid σ' for any linear combinations of $\boldsymbol{M}$ and $\boldsymbol{M'}$. In general, σ can be the concatenation of $\sigma_1 = \mathsf{Sign}(\mathsf{sk}, \boldsymbol{M})$ and $\sigma_2 = \mathsf{Sign}(\mathsf{sk}, \boldsymbol{M'})$, but some joint random coins may be needed, and some common elements can be merged (the tag), as it will be shown in the full instantiation.

We will also be interested in signing affine spaces: given a signature on $\boldsymbol{M}$ and $\boldsymbol{N}$, one wants to limit signatures on $\boldsymbol{M} \times \boldsymbol{N}^\alpha$ and $1 \times \boldsymbol{N}^\beta$. This is possible by expanding the messages with one more component: for $\overline{\boldsymbol{M}} = (g, \boldsymbol{M})$ and $\overline{\boldsymbol{N}} = (1, \boldsymbol{N})$, linear combinations are of the form $(g^\alpha, \boldsymbol{M}^\alpha\boldsymbol{N}^\beta)$. By imposing the first component to be g, one limits to $\alpha = 1$, and thus to $(g, \boldsymbol{M}\boldsymbol{N}^\beta) = \overline{\boldsymbol{M}} \times \overline{\boldsymbol{N}}^\beta$, while by imposing the first component to be 1, one limits to $\alpha = 0$, and thus to $(1, \boldsymbol{N}^\beta) = \overline{\boldsymbol{N}}^\beta$.

3.4 FSH Linearly-Homomorphic Signature Scheme

In [LPJY13], they proposed a *full-fledged* LH-Sign by adding a public tag during the signature. In our mix-net construction, tags will be related to the identities of the users, and so some kind of randomizability will be required for anonymity, which is not possible with their scheme. Instead, we will consider the scheme proposed in [FHS19], which is a full-fledge LH-Sign version of our previous scheme. We can describe it as follows, using our notations:

$\mathsf{Setup}(1^\kappa)$: Given a security parameter κ, let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, \mathfrak{g}, e)$ be an asymmetric bilinear setting, where g and $\mathfrak{g}$ are random generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. The set of tags is $\mathcal{T} = \mathbb{G}_1 \times \mathbb{G}_2$. We then define $\mathsf{param} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, \mathfrak{g}, e; \mathcal{T})$;

$\mathsf{Keygen}(\mathsf{param}, n)$: Given the public parameters param, one randomly chooses $\mathsf{sk}_i = s_i \xleftarrow{\$} \mathbb{Z}_p$, for $i = 1, \ldots, n$, which defines the signing key $\mathsf{sk} = (\mathsf{sk}_i)_i$, and the verification key $\mathsf{vk} = (\mathfrak{g}_i)_{i=0}^n$ for $\mathfrak{g}_i = \mathfrak{g}^{s_i}$ and $\mathfrak{g}_0 = \mathfrak{g}$;

$\mathsf{NewTag}(\mathsf{sk})$: It chooses a random scalar $R \xleftarrow{\$} \mathbb{Z}_p$ and sets $\tau = (\tau_1 = g^{1/R}, \tau_2 = \mathfrak{g}_0^{1/R})$ and $\tilde{\tau} = R$;

$\mathsf{VerifTag}(\mathsf{vk}, \tau)$: Given a verification key $\mathsf{vk} = (\mathfrak{g}_i)_{i=0}^n$ and a tag $\tau = (\tau_1, \tau_2)$, it checks whether $e(\tau_1, \mathfrak{g}_0) = e(g, \tau_2)$ holds or not;

$\mathsf{Sign}(\mathsf{sk}, \tilde{\tau}, \boldsymbol{M} = (M_i)_i)$: Given a signing key $\mathsf{sk} = (s_i)_i$ and a vector-message $\boldsymbol{M} = (M_i)_i \in \mathbb{G}_1^n$, together with some secret tag $\tilde{\tau}$, one sets $\sigma = (\prod_i M_i^{s_i})^{\tilde{\tau}}$;

$\mathsf{DerivSign}(\mathsf{vk}, \tau, (\omega_i, \boldsymbol{M}_i, \sigma_i)_{i=1}^{\ell})$**:** Given a verification key vk, a tag τ and ℓ tuples of weights $\omega_i \in \mathbb{Z}_p$ and signed messages $\boldsymbol{M}_i$ in σ_i, it outputs $\sigma = \prod \sigma_i^{\omega_i}$;

$\mathsf{Verif}(\mathsf{vk}, \tau, \boldsymbol{M} = (M_i)_i, \sigma)$**:** Given a verification key $\mathsf{vk} = (\mathfrak{g}_i)_i$, a vector-message $\boldsymbol{M} = (M_i)_i$, and a signature σ under the tag $\tau = (\tau_1, \tau_2)$, one checks if the equalities $e(\sigma, \tau_2) = \prod_{i=1}^{n} e(M_i, \mathfrak{g}_i)$ and $e(\tau_1, \mathfrak{g}_0) = e(g, \tau_2)$ hold or not.

When the secret keys for tags are all privately and randomly chosen, independently for each signature, unforgeability has been proven in [FHS19], under Chosen-Message Attacks, in the generic bilinear group model. The intuition is the following: first, under the Knowledge of Exponent Assumption [Dam92, HT98, Gro10], from a new pair (τ_1, τ_2), on the input of either $(g, \mathfrak{g})$ or any other honestly generated pair $(g, \mathfrak{g}_0)$, one can extract the common exponent $1/R$ in the two components. Then, one can see σ as the signature with the secret key $(Rs_i)_i$, with the generator $\mathfrak{g}_0^{1/R}$, instead of $\mathfrak{g}_0$ in the previous construction.

However, if one knows two signatures σ and σ' on $\boldsymbol{M}$ and $\boldsymbol{M}'$ respectively, under the same tag $\tau = (\tau_1, \tau_2)$ with private key $\tilde{\tau}$, and the same key vk, then $\sigma^{\alpha}\sigma'^{\beta}$ is a valid signature of $\boldsymbol{M}^{\alpha}\boldsymbol{M}'^{\beta}$, still under the same tag τ and the same key vk: this is thus a $\mathsf{LH\text{-}Sign}$, where one can control the families of messages that can be combined.

In addition, one can define a tag randomizable property:

Property 11 (Tag Randomizability). Given a valid tuple $(\mathsf{vk}, \tau, \boldsymbol{M}, \sigma)$, one can derive a new valid tuple $(\mathsf{vk}, \tau', \boldsymbol{M}, \sigma')$, for a tag τ' unlinkable to τ.

Our $\mathsf{LH\text{-}Sign}$ has the tag randomizability property, with the algorithm $\mathsf{RandTag}$ defined by:

$\mathsf{RandTag}(\mathsf{vk}, \tau, \boldsymbol{M}, \sigma)$**:** Given a verification key vk, a tag $\tau = (\tau_1, \tau_2)$ and a signature σ on a vector-message $\boldsymbol{M} = (M_i)_i \in \mathbb{G}_1^n$, it chooses $\mu \in \mathbb{Z}_p^*$ and outputs $\tau' = (\tau_1^{1/\mu}, \tau_2^{1/\mu})$ and adapts $\sigma' = \sigma^{\mu}$.

Indeed, from a signature σ on $\boldsymbol{M}$ under the tag $\tau = (\tau_1, \tau_2)$ for the key vk, $\sigma' = \sigma^{\mu}$ is a new signature on $\boldsymbol{M}$ for the same key vk under the tag $\tau' = (\tau_1^{1/\mu}, \tau_2^{1/\mu})$, perfectly unlinkable to τ, as this is a new random Diffie-Hellman tuple in basis $(g, \mathfrak{g}_0)$ with $\tilde{\tau}' = \mu\tilde{\tau}$, for $\mathfrak{g}_0$ in vk.

As already explained above, we will essentially work on sub-vector spaces of dimension 2: we will thus denote $\sigma = (\sigma_1, \sigma_2) = \mathsf{Sign}(\mathsf{sk}, \tilde{\tau}, (\boldsymbol{M}, \boldsymbol{M}'))$, under the tag $\tau = (\tau_1, \tau_2)$, where $\sigma_1 = \mathsf{Sign}(\mathsf{sk}, \tilde{\tau}, \boldsymbol{M})$ and $\sigma_2 = \mathsf{Sign}(\mathsf{sk}, \tilde{\tau}, \boldsymbol{M}')$, for a common private key $R = \tilde{\tau}$ which led to $\tau = (\tau_1, \tau_2)$.

Note that in the following, the use of this $\mathsf{LH\text{-}Sign}$ signature scheme will swap $\mathbb{G}_1$ and $\mathbb{G}_2$, as the messages to be signed will be the verification keys of the previous $\mathsf{OT\text{-}LH\text{-}Sign}$ signature scheme, and thus in $\mathbb{G}_2$. Then the verification keys of this $\mathsf{LH\text{-}Sign}$ scheme will be in $\mathbb{G}_1$.

4 Mix-Networks

A mix-net is a network of mix-servers [Cha81] that allows to shuffle ciphertexts so that all the input ciphertexts are in the output set, but cannot be linked together. Whereas it is easy for a server to apply a random permutation on ciphertexts and randomize them, it is not that easy to provide a proof of correctness that is publicly verifiable, and compact. In this section we present our mix-net where the proof of correctness will be implicit thanks to the properties of the (linearly-homomorphic) signatures and two proofs of Diffie-Hellman tuples.

In a first step, we provide a high-level description of our construction to give the intuitions of our new method. However, this high-level presentation suffers several issues, which are then presented in the second step, while the third step details the solutions, with the full scheme. At this point, the global proof of mixing, after several mix-servers, is linear (and verification thus has a linear cost) in the number of mix-servers. In the fourth and last step, we explain how to obtain a constant-time overhead for the proof to publish, and thus for the verification.

4.1 General Description

We first provide a high-level description of our mix-net in Figure 1. As said above, the goal of this presentation is just for the intuition: there are still many problems, that will be highlighted and addressed in the next sections. We need two signature schemes:

- any $\mathsf{OT\text{-}LH\text{-}Sign}$ scheme (Setup,Keygen,Sign,$\mathsf{DerivSign}$,Verif), with additional $\mathsf{DerivSignKey}$, that will be used to sign ElGamal ciphertexts in $\mathbb{G}_1$: the ciphertexts C_i and the signatures σ_i belong to $\mathbb{G}_1$ and are verified with the user' verification keys $\mathsf{vk}_i = (\mathfrak{g}_k)_k$ in $\mathbb{G}_2$;
- and any $\mathsf{LH\text{-}Sign}$ with randomizable tag scheme (Setup^*,Keygen^*,NewTag^*,$\mathsf{RandTag}^*$,$\mathsf{VerifTag}^*$, Sign^*, $\mathsf{DerivSign}^*$, Verif^*) that will be used to sign users' verification keys vk_i in $\mathbb{G}_2$: the signatures Σ_i also belong to $\mathbb{G}_2$ and are verified with Certification Authority's verification key $\mathsf{VK} = (g_k)_k$ in $\mathbb{G}_1$.

Each user $\mathcal{U}_i$ generates a pair $(\mathsf{sk}_i, \mathsf{vk}_i) \leftarrow \mathsf{Keygen}()$ to sign vectors in $\mathbb{G}_1$. $\mathcal{U}_i$ first encrypts his message M_i under an ElGamal encryption scheme, with encryption key EK and signs it to obtain the signed-encrypted ballot $(C_i, \sigma_{i,1})$ under vk_i. Obviously, some guarantees are needed.

In order to be sure that a ballot is legitimate, all the verification keys must be certified by the system (certification authority CA) that signs vk_i under SK, where $(\mathsf{SK}, \mathsf{VK}) \leftarrow \mathsf{Keygen}^*()$, into Σ_i. Then, anyone can verify the certified keys $(\mathsf{vk}_i, \Sigma_i)_i$ are valid under the system verification key VK. Since we want to avoid combinations between verification keys, we use $\mathsf{LH\text{-}Sign}$ with randomizable tags to sign the verification keys with a tag τ_i per user $\mathcal{U}_i$.

Because of encryption, M_i is protected, but this is not enough as it will be decrypted in the end. One also needs to guarantee unlinkability between the input and output ballots to guarantee the anonymity of the user. As the ballot boxes contain the ciphertexts, as well as the verification keys, the ballots must be transformed in an unlinkable way, then they can be output in a permuted way.

To have C'_i unlinkable to C_i, C'_i must be a randomization of C_i. With an ElGamal encryption, it is possible to randomize a ciphertext by multiplying by an encryption of 1. Thus, anyone can compute an encryption C_0 of 1, and as we use an $\mathsf{OT\text{-}LH\text{-}Sign}$ scheme, from a signature $\sigma_{i,0}$ of C_0

CA = Certificate Authority, $\mathcal{U}_i$ = User$_i$, $\mathcal{S}_j$ = Mix-Server$_j$

Keys

$$\text{CA's keys: } \begin{cases} (\mathsf{SK}, \mathsf{VK}) \leftarrow \mathsf{Keygen}^*() & \text{Authority } \mathsf{LH\text{-}Sign} \text{ signing key} \\ (\mathsf{EK}, \mathsf{DK}) \leftarrow \mathsf{EKeygen}() & \text{Authority homomorphic encryption key} \end{cases}$$

$\mathcal{U}_i$'s keys: $(\mathsf{sk}_i, \mathsf{vk}_i) \leftarrow \mathsf{Keygen}()$ User $\mathsf{OT\text{-}LH\text{-}Sign}$ signing key

CA signs vk_i: $(\tilde{\tau}_i, \tau_i) \leftarrow \mathsf{NewTag}^*(\mathsf{SK})$ $\Sigma_i \leftarrow \mathsf{Sign}^*(\mathsf{SK}, \tilde{\tau}_i, \mathsf{vk}_i)$

Ciphertext for randomization: $C_0 \leftarrow \mathsf{Encrypt}(\mathsf{EK}, 1)$

Initial ballots (for $i = 1, \ldots, n$)

$$\mathcal{U}_i \text{ generates: } \begin{cases} C_i \leftarrow \mathsf{Encrypt}(\mathsf{EK}, M_i) & \text{User's ballot encryption} \\ \sigma_{i,0} \leftarrow \mathsf{Sign}(\mathsf{sk}_i, C_0) & \text{User's signature on randomization} \\ \sigma_{i,1} \leftarrow \mathsf{Sign}(\mathsf{sk}_i, C_i) & \text{User's ballot signature} \end{cases}$$

$\mathcal{BB}\mathsf{ox}^{(0)} = (C_i, \sigma_{i,0}, \sigma_{i,1}, \mathsf{vk}_i, \Sigma_i, \tau_i)_i$

Mix (j-th mix-server, for $i = 1, \ldots, n$)

From $\mathcal{BB}\mathsf{ox}^{(j-1)} = (C_i, \sigma_{i,0}, \sigma_{i,1}, \mathsf{vk}_i, \Sigma_i, \tau_i)_i$, $\mathcal{S}_j$ makes, for all i:

Randomization of the ballot:

$$C'_i = C_i \cdot {C_0}^{\gamma_{j,i}} \qquad \sigma^*_{i,1} = \mathsf{DerivSign}(\mathsf{vk}_i, \{(1, C_0, \sigma_{i,0}), (\gamma_{j,i}, C_i, \sigma_{i,1})\})$$

Randomization of the keys:

$$\begin{cases} \mathsf{vk}'_i = (\mathsf{vk}_i)^{\alpha_j} \qquad \Sigma^*_i = \mathsf{DerivSign}^*(\mathsf{VK}, \tau_i, (\alpha_j, \mathsf{vk}_i, \Sigma_i)) \\ (\mathsf{VK}, \tau'_i, \mathsf{vk}_i, \Sigma'_i) = \mathsf{RandTag}^*(\mathsf{VK}, \tau_i, \mathsf{vk}_i, \Sigma^*_i) \end{cases}$$

Adaptation of the signatures:

$$\begin{aligned} \sigma'_{i,0} &= \mathsf{DerivSignKey}(C_0, (\alpha_j, \mathsf{vk}_i, \sigma_{i,0})) \\ \sigma'_{i,1} &= \mathsf{DerivSignKey}(C'_1, (\alpha_j, \mathsf{vk}_i, \sigma^*_{i,1})) \end{aligned}$$

$\mathcal{BB}\mathsf{ox}^{(j)} = (C'_{\Pi(i)}, \sigma'_{\Pi(i),0}, \sigma'_{\Pi(i),1}, \mathsf{vk}'_{\Pi(i)}, \Sigma'_{\Pi(i)}, \tau'_{\Pi(i)})_i$

Figure 1. High-Level Description (Insecure Scheme)

under the user's key, one can adapt $\sigma_{i,1}$ by using the message homomorphism (Property 7) with $\mathsf{DerivSign}$ to obtain $\sigma^*_{i,1}$. In the same way, vk'_i and τ'_i must be randomizations of respectively vk_i and τ_i. If $\mathsf{vk}'_i = \mathsf{vk}_i^\alpha$, its signature must be derived from Σ_i with $\mathsf{DerivSign}^*$ and τ'_i is obtained with the randomizable tag (Property 11) with $\mathsf{RandTag}^*$. Eventually, as we change the verification key, $\sigma'_{i,0}$ and $\sigma'_{i,1}$ must be adapted, which is possible thanks to the weak key homomorphism (Property 9) with $\mathsf{DerivSignKey}$.

Then one generates a random permutation Π to output a new ballot-box with permuted randomized ballots $(\mathsf{vk}'_{\Pi(i)}, \Sigma'_{\Pi(i)}, C'_{\Pi(i)}, \sigma'_{\Pi(i),0}, \sigma'_{\Pi(i),1})_i$.

4.2 Difficulties

The above high-level scheme gives intuitions of our main approach. However, to get the required security, we still face a few issues that will be explained below and which motivate the full scheme described in the next section.

Expanded Vectors. From the signatures $\sigma_{i,0}$ and $\sigma_{i,1}$ with an $\mathsf{OT\text{-}LH\text{-}Sign}$ scheme, anyone can compute $\sigma = \mathsf{DerivSign}(\mathsf{vk}_i, \{(\alpha, C_0, \sigma_{i,0}), (\beta, C_i, \sigma_{i,1})\})$ for any α, β. As explained in Section 3.3, we can impose $\beta = 1$ and the right format of C'_i.

Non-Trivial Transformation. The weak key homomorphism allows to randomize vk_i into $\mathsf{vk}'_i = \mathsf{vk}_i^\alpha$ but, with our scheme, $\mathsf{Verif}(\mathsf{vk}_i^\alpha, C_i, \sigma_{i,1})$ is valid for any $\alpha \neq 0$ if and only if $\mathsf{Verif}(\mathsf{vk}_i, C_i, \sigma_{i,1})$ is valid. This provides a link between vk'_i and vk_i. To solve this issue, we introduce a randomizer vk_0, as for the ciphertext. This is a special vector also signed by CA to randomize vk_i in a non-trivial way: $\mathsf{vk}'_i = (\mathsf{vk}_i \cdot \mathsf{vk}_0^{\delta_i})^\alpha$. We will thus also have the signature $\Sigma_{i,0}$ of vk_0 and the signature $\Sigma_{i,1}$ (instead of Σ_i) of vk_i, both under the same tag τ_i to allow combinations.

Legitimate Ballots. Whereas all the ballots must be signed, nothing prevents a mix-server to delete a ballot or to add a ballot signed by a legitimate user (that owns a valid key vk_i). If one first checks that the number of ballots is kept unchanged, it is still possible that a ballot was replaced by a new legitimate ballot. Since we will consider honest and corrupted users (and so honest and corrupted ballots), four cases are possible: one replaces an honest or corrupted ballot by another honest or corrupted one. Our scheme will not provide guarantees against the replacement of a corrupted ballot by another corrupted ballot. Nonetheless, by adding a zero-knowledge proof of Diffie-Hellman tuple between the products of the verification keys before and after the mix, we can avoid all the other cases involving honest users.

Active Replication of Ballots. As in the attack of Cortier-Smyth [CS13], if a non-negligible group of users colludes with a (the first) mix-server and copies the ciphertext of an honest user, and sign it, then the final decryption reveals the content of the ciphertext of the honest user, as multiple plaintexts will be identical. This is possible to avoid this attack against privacy with a similar technique as above: each mix-server proves the multiplications of the input ciphertexts and of the output ciphertexts are the same up to a Diffie-Hellman tuple. For plaintexts with enough entropy (which is required to be able to break privacy with the Cortier-Smyth attack) this proof of Diffie-Hellman ratio prevents the attack.

Multiple Servers. After the last round, one gets a proof that the output ballot-box contains a permutation of randomized ciphertexts from the input ballot-box. However, the last mix-server could start from the initial ballot-box instead of the previous one, and then know the permutation. This would break anonymity, as soon as the last mix-server is dishonest. We will ask the mix-servers to sign their contributions to prove the multiple and independent permutations: each mix-server j generates the Diffie-Hellman proofs from $\mathcal{BB}\mathsf{ox}^{(j-1)}$ to $\mathcal{BB}\mathsf{ox}^{(j)}$, and signs them. We will then detail this solution in the next section, which will provide a proof linear in the number of ballots and in the number of mix-servers (because of the multiple signature). Thereafter, with specific multi-signature, one can become independent of the number of mix-servers.

4.3 Our Scheme

With all the previous remarks and explanations, we can now provide the full description of our scheme which is given in Figure 2.

CA = Certificate Authority, $\mathcal{U}_i$ = User$_i$, $\mathcal{S}_j$ = Mix-Server$_j$

$\mathsf{MixSetup}(1^\kappa)$:

Let $\mathsf{param} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, \mathfrak{g}, e) \leftarrow \mathsf{Setup}(1^\kappa)$ and $\mathsf{param}' = \{\mathsf{param}, \mathcal{T} = \mathbb{G}_2 \times \mathbb{G}_1\}$;
Let $\mathsf{NIZK_{DH}\text{-}param} \leftarrow \mathsf{NIZK_{DH}\text{-}Setup}(1^\kappa)$ and $\mathsf{Sparam} \leftarrow \mathsf{SSetup}(1^\kappa)$;
Let $(\mathsf{DK} = d, \mathsf{EK} = h = g^d) \leftarrow \mathsf{EKeygen}(1^\kappa)$ and $\overline{C}_0 = (1, \ell, g, h)$ for $\ell \xleftarrow{\$} \mathbb{G}_1$;
It outputs $\mathsf{Mix\text{-}param} = (\mathsf{param}', \mathsf{NIZK_{DH}\text{-}param}, \mathsf{Sparam}, \mathsf{EK}, \ell)$.

$\mathsf{MixKeygen}(\mathsf{Mix\text{-}param})$:

$$\mathsf{CA}: \begin{cases} \mathsf{SK}= (S_1, S_2, S_3, S_4, S_5) \xleftarrow{\$} \mathbb{Z}_p^5,\ \mathsf{VK} = (g, g^{S_1}, g^{S_2}, g^{S_3}, g^{S_4}, g^{S_5}) \\ \text{and for each user } \mathcal{U}_i,\ \tilde{\tau}_i = R_i \xleftarrow{\$} \mathbb{Z}_p,\ \tau_i = (\tau_{i,1} = \mathfrak{g}^{1/R_i}, \tau_{i,2} = g^{1/R_i}) \end{cases}$$

$\mathsf{vk}_0 = (1, 1, \mathfrak{g}_0 = \mathfrak{g}, 1, 1)$

$\mathcal{S}_j$: $\{ (\mathsf{SK}_j, \mathsf{VK}_j) \leftarrow \mathsf{SKeygen}()$

$$\mathcal{U}_i: \begin{cases} \mathsf{sk}_i= (u_i, v_i, x_i, y_i) \xleftarrow{\$} \mathbb{Z}_p^4,\ \mathsf{vk}_i = (\mathfrak{g}_0 = \mathfrak{g}, \mathfrak{f}_i = \mathfrak{g}_0^{u_i}, \mathfrak{l}_i = \mathfrak{g}_0^{v_i}, \mathfrak{g}_i = \mathfrak{g}_0^{x_i}, \mathfrak{h}_i = \mathfrak{g}_0^{y_i}) \\ \Sigma_i= \left(\Sigma_{i,0} = \mathfrak{g}^{S_3 \tilde{\tau}_i}, \Sigma_{i,1} = (\mathfrak{g}_0^{S_1} \mathfrak{f}_i^{S_2} \mathfrak{l}_i^{S_3} \mathfrak{g}_i^{S_4} \mathfrak{h}_i^{S_5})^{\tilde{\tau}_i}\right) \end{cases}$$

$\mathsf{MixInit}(\mathsf{sk}_i, M_i, \mathsf{vk}_i, \Sigma_i, \tau_i)$:

$\mathcal{U}_i$ chooses $r_i \xleftarrow{\$} \mathbb{Z}_p$ and $\ell_i \xleftarrow{\$} \mathbb{G}_1$ and computes

$$C_i = (a_i = g^{r_i}, b_i = h^{r_i} M_i) \qquad \overline{C}_i = (g, \ell_i, a_i, b_i)$$

$$\sigma_i = (\sigma_{i,0} = \ell^{v_i} g^{x_i} h^{y_i}, \sigma_{i,1} = g^{u_i} \ell_i^{v_i} a_i^{x_i} b_i^{y_i})$$

It outputs $\mathcal{B}_i = (C_i, \ell_i, \sigma_i, \mathsf{vk}_i, \Sigma_i, \tau_i)$.

$$\mathcal{BB}\mathsf{ox}^{(0)} = (\mathcal{B}_i)_{i=1}^N$$

$\mathsf{Mix}(\mathsf{SK}_j, \mathcal{BB}\mathsf{ox}^{(j-1)}, (\mathsf{proof}^{(k)}, \mathsf{sig}^{(k)})_{k=1}^{j-1}, \Pi_j)$:

From $\mathcal{BB}\mathsf{ox}^{(j-1)} = (C_i, \ell_i, \sigma_i, \mathsf{vk}_i, \Sigma_i, \tau_i)_i, (\mathsf{proof}^{(k)}, \mathsf{sig}^{(k)})_{k=1}^{j-1}$,
$\mathcal{S}_j$ chooses $\alpha \xleftarrow{\$} \mathbb{Z}_p$ and for each ballot i, $\gamma_i, \delta_i, \mu_i \xleftarrow{\$} \mathbb{Z}_p$ and computes

$$a_i' = a_i \cdot g^{\gamma_i} \quad b_i' = b_i \cdot h^{\gamma_i} \quad \ell_i' = \ell_i \cdot \ell^{\gamma_i} \quad \sigma_{i,1}' = \sigma_{i,1} \cdot \sigma_{i,0}^{\gamma_i} \cdot \ell_i'^{\delta_i} \quad \sigma_{i,0}' = \sigma_{i,0} \cdot \ell^{\delta_i}$$

$$\mathfrak{g}_0' = \mathfrak{g}_0^\alpha \qquad \mathfrak{f}_i' = \mathfrak{f}_i^\alpha \qquad \mathfrak{l}_i' = (\mathfrak{l}_i \cdot \mathfrak{g}_0^{\delta_i})^\alpha \qquad \mathfrak{g}_i' = \mathfrak{g}_i^\alpha \qquad \mathfrak{h}_i' = \mathfrak{h}_i^\alpha$$

$$\Sigma_{i,1}' = (\Sigma_{i,1} \cdot \Sigma_{i,0}{}^{\delta_i})^{\alpha\mu_i} \qquad \Sigma_{i,0}' = \Sigma_{i,0}^{\alpha\mu_i} \qquad \tau_{i,1}' = \tau_{i,1}^{1/\mu_i} \qquad \tau_{i,2}' = \tau_{i,2}^{1/\mu_i}$$

$$\begin{cases} \mathsf{proof}^{(j)} = \mathsf{NIZK_{DH}\text{-}Proof}((\mathfrak{g}_0, \mathfrak{g}_0', \prod \mathfrak{f}_i, \prod \mathfrak{f}_i') \text{ and } (g, h, \prod a_i' / \prod a_i, \prod b_i' / \prod b_i)) \\ \quad \mathsf{sig}^{(j)} = \mathsf{SSign}(\mathsf{SK}_j, \mathsf{proof}^{(j)}) \end{cases}$$

$\mathcal{S}_j$ outputs $\mathcal{BB}\mathsf{ox}^{(j)} = (C_{\Pi_j(i)}', \ell_{\Pi_j(i)}', \sigma_{\Pi_j(i)}', \mathsf{vk}_{\Pi_j(i)}', \Sigma_{\Pi_j(i)}', \tau_{\Pi_j(i)}')_i, (\mathsf{proof}^{(k)}, \mathsf{sig}^{(k)})_{k=1}^j$

Figure 2. Detailed Shuffling of ElGamal Ciphertexts

Keys. As we will sign expanded ciphertexts of dimension 4 (see below), each user needs a secret-verification key pair $(\mathsf{sk}_i, \mathsf{vk}_i) \leftarrow \mathsf{Keygen}(\mathsf{param}, 4)$ in $\mathbb{Z}_p^4 \times \mathbb{G}_2^5$. With our $\mathsf{OT\text{-}LH\text{-}Sign}$, the first element of vk_i is common for all the users and initialized to $\mathfrak{g}_0 = \mathfrak{g}$. Then, one also needs a signature $\Sigma_i = (\Sigma_{i,0}, \Sigma_{i,1})$ with our $\mathsf{LH\text{-}Sign}$ from the certification authority of the pair $(\mathsf{vk}_0, \mathsf{vk}_i)$ where $\mathsf{vk}_0 = (1, 1, \mathfrak{g}_0, 1, 1)$ is used to make the non-trivial transformation on vk_i during the mixes. This signature is signed by the authority possessing $(\mathsf{SK}, \mathsf{VK}) \leftarrow \mathsf{Keygen}^*(\mathsf{param}', 5)$ in $\mathbb{Z}_p^5 \times \mathbb{G}_1^6$ with a specific tag τ_i per user. Eventually, each mix-server has a pair of (standard) signature scheme $(\mathsf{SK}_j, \mathsf{VK}_j) \leftarrow \mathsf{SKeygen}()$ just to sign with SSign its mixing contribution. The keys VK and $(\mathsf{VK}_j)_j$, as well as $\mathsf{EK} = h = g^d \in \mathbb{G}_1$ and the random $\ell \xleftarrow{\$} \mathbb{G}_1$, are assumed to be known to everybody.

Since we are using ciphertexts with ElGamal, the ciphertext for randomization is $C_0 = (g, h)$, the trivial encryption of $1 = g^0$, with random coin equal to 1.

Initial ballots. Each user encrypts his message M_i under EK to obtain $C_i = (a_i, b_i)$. With the remarks we already made, one needs to expand C_i into $\overline{C}_i = (g, \ell_i, a_i, b_i)$ and C_0 into $\overline{C}_0 = (1, \ell, g, h)$. The addition of the first element is due to the affine space we want in the signature σ_i (see Section 3.3) and the second element is because we randomize the third position of vk_i with $\mathsf{vk}_0 = (1, 1, \mathfrak{g}_0, 1, 1)$ and because the first position of vk_i is used for the verification but not to sign (the last four elements of vk_i are used to sign). Finally, $\sigma_i = (\sigma_{i,0}, \sigma_{i,1})$ is simply the $\mathsf{OT\text{-}LH\text{-}Sign}$ of $(\overline{C}_0, \overline{C}_i)$ under the signing key sk_i.

Mix. To make a mix, the j-th mix-server computes the randomized verification keys $\mathsf{vk}'_i = (\mathsf{vk}_i \cdot \mathsf{vk}_0^{\delta_i})^\alpha$, the randomized ciphertexts $\overline{C}'_i = \overline{C}_i \cdot \overline{C}_0^{\gamma_i}$ and the randomized tags $\tau'_i = \tau_i^{1/\mu_i}$, and updates the signatures σ'_i and Σ'_i, thanks to the properties of the signatures. The random scalar α is common to all the ballots, but $\gamma_i, \delta_i, \mu_i$ are independent random scalars for each ballot. Then, the mix-server chooses a permutation Π and sets the j-th ballot-box $\mathcal{BB}\mathsf{ox}^{(j)}$ with all the randomized and permuted ballots $(C'_{\Pi(i)}, \ell'_{\Pi(i)}, \sigma'_{\Pi(i)}, \mathsf{vk}'_{\Pi(i)}, \Sigma'_{\Pi(i)}, \tau'_{\Pi(i)})_i$. As already explained, the mix-server also needs to make a proof $\mathsf{proof}^{(j)}$ from $\mathcal{BB}\mathsf{ox}^{(j-1)}$ to $\mathcal{BB}\mathsf{ox}^{(j)}$, to guarantee the proper relations between the products of the verification keys and the products of the messages, and signs it in $\mathsf{sig}^{(j)}$. Finally, the output of the mix contains $\mathcal{BB}\mathsf{ox}^{(j)}$ and $(\mathsf{proof}^{(k)}, \mathsf{sig}^{(k)})_{k=1}^{j}$ the set of proofs and mix-server signatures of the previous mixes until the j-th mix.

Proofs. Let us denote $\mathfrak{F} = \prod \mathfrak{f}_i = \mathfrak{g}_0^{\sum u_i}$ and $\mathfrak{F}' = \prod \mathfrak{f}'_i = \mathfrak{g}_0'^{\sum u_i}$ the product of the second element of the user's verification key on all the input ballots and output ballots. If the input and output ballot-boxes contain the same ballots (with the same secret u_i), then $\mathfrak{F}' = \mathfrak{F}^\alpha$, with $\mathfrak{g}'_0 = \mathfrak{g}_0^\alpha$. Hence one adds a proof of Diffie-Hellman tuple for $(\mathfrak{g}_0, \mathfrak{g}'_0, \mathfrak{F}, \mathfrak{F}')$. Together with the verification that there is the same number of ballots in the input and output of the mix, we will show that the same (honest) users are represented in the two ballot-boxes. Since we cannot allow multiple ballots from the same user, we have the guarantee that the same messages from all the honest users are represented in the two ballot-boxes.

The additional proof of Diffie-Hellman tuple for $(g, h, \prod a'_i / \prod a_i, \prod b'_i / \prod b_i)$ will limit the exchange of ballots for corrupted users, as the products of the plaintexts must remain the same: $\prod M'_i = \prod M_i$. Since we already know these products will be the same for honest users, this products must be the same from corrupted users. This will limit the impact of the attack of Cortier-Smyth [CS13].

With these two Diffie-Hellman proofs, the output ballots are a permutation of the input ones. We could use any non-interactive zero-knowledge proofs of Diffie-Hellman tuples $(\mathsf{NIZK_{DH}\text{-}Setup}, \mathsf{NIZK_{DH}\text{-}Proof}, \mathsf{NIZK_{DH}\text{-}Verif})$ and any signature $(\mathsf{SSetup}, \mathsf{SSign}, \mathsf{SVerif})$ to sign the proofs but the next section will provide interesting choices, from the length point of view.

$\mathsf{MixVerif}(\mathcal{BB}\mathsf{ox}^{(0)}, \mathcal{BB}\mathsf{ox}^{(N)}, (\mathsf{proof}^{(k)}, \mathsf{sig}^{(k)})_{k=1}^{N})$:

After N mixes, the input of the verifier is:

$$\mathcal{BB}\mathsf{ox}^{(0)} = (\overline{C}_i, \sigma_{i,1}, \mathsf{vk}_i, \Sigma_{i,1}, \tau_{i,1})_{i=1}^{n}$$

$$\mathcal{BB}\mathsf{ox}^{(N)} = (\overline{C}'_i, \sigma'_{i,1}, \mathsf{vk}'_i, \Sigma'_{i,1}, \tau'_{i,1})_{i=1}^{n'}, (\mathsf{proof}^{(k)}, \mathsf{sig}^{(k)})_{k=1}^{N}$$

It outputs 1 if: $n = n'$, the $(\mathsf{vk}_i)_i$ are all distinct

$\forall k,$

$$\mathsf{NIZK_{DH}\text{-}Verif}(\mathsf{proof}^{(k)})) = 1$$
$$\mathsf{SVerif}(\mathsf{VK}_k, \mathsf{proof}^{(k)}), \mathsf{sig}^{(k)}) = 1$$

and $\forall i,$

$$\mathsf{Verif}(\mathsf{vk}_i, \overline{C}_i, \sigma_{i,1}) = 1 = \mathsf{Verif}^*(\mathsf{VK}, \tau_i, \mathsf{vk}_i, \Sigma_{i,1})$$
$$\mathsf{Verif}(\mathsf{vk}'_i, \overline{C}'_i, \sigma'_{i,1}) = 1 = \mathsf{Verif}^*(\mathsf{VK}, \tau'_i, \mathsf{vk}'_i, \Sigma'_{i,1})$$

Figure 3. Detailed Verification of Shuffling

Verification. The complete verification process, after N mix-servers, is presented in Figure 3. After all the mixes are done, it just requires the input ballot-box $\mathcal{BB}\mathsf{ox}^{(0)}$, the output ballot-box $\mathcal{BB}\mathsf{ox}^{(N)}$, and the signed proofs $(\mathsf{proof}^{(k)}, \mathsf{sig}^{(k)})$, for $k = 1, \ldots, N$ without the elements that were useful for randomization only. The verifier checks the number of input ballots is the same as the number of output ballots, the verification keys (the $\mathfrak{f}_i$'s) in input ballots are all distinct, the signatures $\sigma_{i,1}, \sigma'_{i,1}, \Sigma_{i,1}$ and $\Sigma'_{i,1}$ are valid on individual input and output tuples (equations recalled in Annexe B.1) and all the proofs $\mathsf{proof}^{(k)}$ with the signatures $\mathsf{sig}^{(k)}$ are valid with $\mathsf{NIZK_{DH}\text{-}Verif}$ and SVerif respectively. For that, we suppose that the statement is included in each zero-knowledge proof. Thus, even if the intermediate ballot-boxes are not given to the verifier, it is still possible to perform the verification.

4.4 Constant-Size Proof

From Figure 3, one can note that our mix-net provides a quite compact proof, as it just requires $\mathcal{BB}\mathsf{ox}^{(0)}$ and $\mathcal{BB}\mathsf{ox}^{(N)}$, and the signed proofs $(\mathsf{proof}^{(k)}, \mathsf{sig}^{(k)})$, for $k = 1, \ldots, N$. The size is thus linear in n and N. This is the same for the verification complexity.

Whereas the linear complexity in n cannot be avoided, as the ballot-box must be transferred, the part linear in N could be avoided. Indeed, each proof $\mathsf{proof}^{(j)}$ ensures the relations from the $j-1$-th ballot-box to the j-th ballot-box. The global chain of proofs ensures the relations from the initial ballot-box to the last ballot-box. From the soundness point on view, a compact global proof would be enough. But for privacy, one wants to be sure that multiple mix-servers contributed, to get unlinkability as soon as one server is honest.

To avoid the dependence in N, one can use Groth-Sahai proofs [GS08] (see Appendix A.1 for details) to combine together the proofs into a unique one. However, to be sure that all the mix-servers contributed: each mix-server does as above, but also receives a partial proof $\mathsf{proof'}^{(j-1)}$ from the initial ballot-box to the $j-1$-th ballot-box and, thanks to the homomorphic properties of the Groth-Sahai proof, updates it into $\mathsf{proof'}^{(j)}$, to prove the relation from the initial ballot-box and the j-th ballot-box, as shown in Appendix A.1 for the Diffie-Hellman proof between the products of the keys (the proof is similar for the product of the ciphertexts but with $\mathbb{G}_1$ and $\mathbb{G}_2$ swapped).

At the end of the mixing steps, one has the same elements as above, plus the global proof $\mathsf{proof'}^{(N)}$. All the mix-servers can now verify the proofs and the contributions of all the servers. Only this global proof can be kept, but signed by all the servers: using the multi-signature of Boneh-Drijvers-Neven [BDN18], that is recalled in the Appendix A.2, the size of the signature msig keeps constant, whatever the number of mix-servers. Hence, after multiple mixing steps, the size of the mixing proof (in addition to the input and output ballot-boxes) remains constant.

4.5 Efficiency

We consider VK and $(\mathsf{VK}_j)_j$ are long-term keys known to everybody, as well as EK and ℓ. However, for fair comparison, we do not consider vk_i as long-term keys, and consider them as part of the input of the verifier. But we insist that the $\mathfrak{f}_i$'s in the input ballot-box must be all distinct.

Size of Verifier's Input: The verifier receives:

$$(\overline{C}_i, \sigma_{i,1}, \mathsf{vk}_i, \Sigma_{i,1}, \tau_i)_{i=1}^n \quad (\overline{C}'_i, \sigma'_{i,1}, \mathsf{vk}'_i, \Sigma'_{i,1}, \tau'_i)_{i=1}^n \quad (\mathsf{proof'}^{(N)}, \mathsf{msig'}^{(N)})$$

As the first element $\mathfrak{g}_0$ of vk_i is common to all the users (as well as $\mathfrak{g}'_0$ of vk'_i), the set of all the users' verification keys is represented by $4 \times n + 1$ elements of $\mathbb{G}_2$. Then, all input or output ballots contains $2 \times 5n$ elements from $\mathbb{G}_1$ and $2 \times (6n + 1)$ elements from $\mathbb{G}_2$.

The global proof $\mathsf{proof'}^{(N)}$ is just 4 elements of $\mathbb{G}_1$ and 4 elements of $\mathbb{G}_2$ and msig one element in $\mathbb{G}_2$. Hence, the full verifier's input contains: $10n + 4$ elements of $\mathbb{G}_1$, $12n + 6$ elements of $\mathbb{G}_2$, whatever the number of mix-servers.

Verifier's Computation. Using batch verification [BFI+10], the verifier only needs to make $8n + 7$ pairing evaluations to verify together all the signatures $\sigma_{i,1}$, $\sigma'_{i,1}$, $\Sigma_{i,1}$, $\Sigma'_{i,1}$, τ_i, τ'_i, 6 pairing evaluations to verify $\mathsf{proof}'^{(N)}$ and 2 pairing evaluations to verify msig.

With some specific choices of the bases for the batch verification, as presented in the Appendix B.1, one can improve to $8n + 14$ pairing evaluations for the global verification. This has to be compared to the $4n + 1$ pairing evaluations that have anyway to be performed to verify the signatures in the initial ballot-box.

5 Security Analysis

Let us now formally prove the two security properties: the *soundness* means the output ballot-box contains a permutation of randomizations of the input ballot-box and *privacy* means one cannot link an input ciphertext to an output ciphertext, as soon as one mix-server is honest.

We stress that we are in a particular case where users have private signing keys, and ballots are signed. Unfortunately these keys allow to trace the ballots: with $\mathsf{sk}_i = (u_i, v_i, x_i, y_i)$ and $\mathfrak{g}'_0$, one can recover vk'_i, which contradicts privacy for this ballot. They might also allow to exchange some ballots, which contradicts soundness for these ballots. As a consequence, we do not provide any guarantee to corrupted users, whose keys have been given to the adversary (or even possibly generated by the adversary), but we expect honest users to be protected:

- *soundness for honest users* means that all the plaintexts of the honest users in the input ballot-box are in the output ballot-box;
- *privacy for honest users* means that ballots of honest users are unlinkable from the input ballot-box to the output ballot-box.

5.1 Proof of Soundness

As just explained, we first study the soundness of our protocol, but for honest users only, in the certified key setting, where all the users must prove the knowledge of their private keys before getting their verification keys vk_i certified by the Certification Authority in Σ_i.

Definition 12 (Soundness for Honest Users). A mix-net M is said *sound for honest users* in the certified key setting, if any PPT adversary $\mathcal{A}$ has a negligible success probability in the following security game:

1. the challenger generates the certification keys $(\mathsf{SK}, \mathsf{VK})$ and the encryption keys $(\mathsf{DK}, \mathsf{EK})$;
2. the adversary $\mathcal{A}$ then
   - decides on the corrupted users $\mathcal{I}^*$ and generates itself their keys $(\mathsf{vk}_i)_{i\in\mathcal{I}^*}$;
   - proves its knowledge of the secrete keys to get the certifications Σ_i on vk_i, for $i \in \mathcal{I}^*$;
   - decides on the set $\mathcal{I}$ of the (honest and corrupted) users that will generate a ballot;
   - generates the ballots $(\mathcal{B}_i)_{i\in\mathcal{I}^*}$ for the corrupted users but provides the messages $(M_i)_{i\in\mathcal{I}\setminus\mathcal{I}^*}$ for the honest users;
3. the challenger generates the keys of the honest users $(\mathsf{sk}_i, \mathsf{vk}_i)_{i\in\mathcal{I}\setminus\mathcal{I}^*}$ and their ballots $(\mathcal{B}_i)_{i\in\mathcal{I}\setminus\mathcal{I}^*}$. The initial ballot-box is thus defined by $\mathcal{BB}\mathsf{ox} = (\mathcal{B}_i)_{i\in\mathcal{I}}$;
4. the adversary mixes $\mathcal{BB}\mathsf{ox}$ in a provable way into $(\mathcal{BB}\mathsf{ox}', \mathsf{proof})$.

The adversary wins if $\mathsf{MixVerif}(\mathcal{BB}\mathsf{ox}, \mathcal{BB}\mathsf{ox}', \mathsf{proof}) = 1$ but $\{\mathsf{Decrypt}^*(\mathcal{BB}\mathsf{ox})\} \neq \{\mathsf{Decrypt}^*(\mathcal{BB}\mathsf{ox}')\}$, where $\mathsf{Decrypt}^*$ extracts the plaintexts (using the decryption key DK), but ignores ballots of non-honest users (using the private keys of honest users) and sets of plaintexts can have repetitions.

One can note that this security game does not depend on the mixing steps, but just considers the global mixing, from the input ballot-box $\mathcal{BB}\mathsf{ox}$ to the output ballot-box $\mathcal{BB}\mathsf{ox}'$. The proof proof contains all the elements for proving the honest behavior. In our case, this is just the two Diffie-Hellman proofs.

Theorem 13 (Soundness for Honest Users of Our Mix-Net). *Our mix-net protocol is sound for honest users, in the certified key setting, under the unforgeability against Chosen-Message Attacks of the* $\mathsf{LH\text{-}Sign}$ *and* $\mathsf{OT\text{-}LH\text{-}Sign}$ *signature schemes and the* TDL *assumption.*

Proof. For proving this theorem, we will assume the verification is successful ($\mathsf{MixVerif}(\mathcal{BB}\mathsf{ox}, \mathcal{BB}\mathsf{ox}', \mathsf{proof}) = 1$) and show that for all the honest ballots, in the input and output ballot-boxes, there is a permutation from the input ones to the outputs ones. And we do it in two steps: first, honest keys vk'_i in the output ballot-box are permuted randomizations of the honest keys vk_i in the input ballot-box; then we prove it for the plaintexts.

Permutation of Honest Keys. We first modify the security game by using the unforgeability against Chosen-Message Attacks of the $\mathsf{LH\text{-}Sign}$ signature scheme: we are given VK, and ask the Tag-oracle and the Signing-oracle to obtain Σ_i on all the verification keys vk_i and vk_0. The rest remains unchanged. Note that because of the proof of knowledge of the private keys sk_i before getting vk_i certified, one can also extract them. Actually, one just needs to extract u_i for all the corrupted users. Then one knows all the legitimate u_i's (for honest and corrupted users).

Under the unforgeability of the signature scheme (Setup^*, Keygen^*, NewTag^*, $\mathsf{RandTag}^*$, $\mathsf{VerifTag}^*$, Sign^*, $\mathsf{DerivSign}^*$, Verif^*), for any output ballot with verification key vk'_j there exists a related legitimate verification key vk_i such that $\mathsf{vk}'_j = \mathsf{vk}_i^{\alpha_i} \times \mathsf{vk}_0^{z_i}$, for some scalars z_i, and α_i.

Since in our construction $\mathsf{vk}_i = (\mathfrak{g}_0, \mathfrak{f}_i, \mathfrak{l}_i, \mathfrak{g}_i, \mathfrak{h}_i)$ and $\mathsf{vk}_0 = (1, 1, \mathfrak{g}_0, 1, 1)$, and $\mathsf{vk}'_j = (\mathfrak{g}'_0, \mathfrak{f}'_j, \mathfrak{l}'_j, \mathfrak{g}'_j, \mathfrak{h}'_j)$ and $\mathsf{vk}'_0 = (1, 1, \mathfrak{g}'_0, 1, 1)$ with a common $\mathfrak{g}'_0$ for all the keys, α_i is a common scalar α: $\mathsf{vk}'_j = (\mathsf{vk}_i \times \mathsf{vk}_0^{\delta_i})^\alpha$ and $\mathsf{vk}'_0 = \mathsf{vk}_0^\alpha$. As a consequence, all the keys in the output ballot-box are derived in a similar way from legitimate keys (signed by the Certification Authority): $u'_j = u_i$ remains unchanged. However this does not means they were all in the input ballot-box: the adversary could insert a ballot with a legitimate verification key vk_i, which was not in the initial ballot-box.

The verification process also includes a Diffie-Hellman proof for the tuple $(\mathfrak{g}_0, \mathfrak{g}'_0, \prod_i \mathfrak{f}_i, \prod_j \mathfrak{f}'_j)$. This means that $\sum_i u_i$ are the same on the input ballots and the output ballots. As one additionally checks the numbers of input ballots and output ballots are the same, the adversary can just replace an input ballot by a new one: if $\mathcal{N}$ is the set of new ballots and $\mathcal{D}$ the set of deleted ballots, the sums must compensate: $\sum_{\mathcal{D}} u_i = \sum_{\mathcal{N}} u_i$.

The second game uses the TDL assumption and the simulation-soundness of the proof of knowledge of sk_i (in the certified key setting): Let us be given a tuple $(\mathfrak{g}, \mathfrak{f} = \mathfrak{g}^u, g, f = g^u)$, as input of a TDL challenge in $\mathbb{G}_2$ and $\mathbb{G}_1$: the simulator will guess an honest user i^* that will be deleted, and implicitly sets $u_{i^*} = u$, with $\mathfrak{f}_{i^*}$, which allows it to use $f = g^{u_{i^*}}$ in the signature of $\overline{C}_{i^*}$ on the first component g, while all the other scalars are chosen by the simulator $(v_{i^*}, x_{i^*}, y_{i^*})$, as well as all the other honest user' keys, the authority signing keys, and, for all the corrupted users, the secret element u_i can be extracted at the certification time (using the extractor from the zero-knowledge proof of knowledge) while the zero-knowledge simulator is used for i^*, thanks to the simulation-soundness.

If some honest user is deleted in the output ballot-box, with probability greater than $1/n$, this is i^*: but as shown above, $\sum_{\mathcal{D}} u_i = \sum_{\mathcal{N}} u_i$, and so $u_{i^*} = \sum_{\mathcal{N}} u_i - \sum_{\mathcal{D}\backslash\{i^*\}} u_i$, which breaks the twin discrete logarithm assumption.

Permutation of Honest Ballots. The last game uses the unforgeability of the $\mathsf{OT\text{-}LH\text{-}Sign}$ signature scheme under Chosen-Message Attacks: the simulator receives one verification key vk, that will be assigned at a random honest user i^*, whereas all the other keys are honestly generated. The simulator also generates $(\mathsf{SK}, \mathsf{VK})$ and $(\mathsf{DK}, \mathsf{EK})$, as well as all signatures Σ_i and the honest ballots (with a signing query for σ_{i^*}). Then, the adversary outputs a proven mix of the ballot-box. We have just proven that there exists a bijection Π from $\mathcal{I}$ into $\mathcal{J}$ such that $\mathsf{vk}'_{\Pi(i)} = (\mathsf{vk}_i \times \mathsf{vk}_0^{\delta_i})^\alpha$ for some scalar δ_i, for all the honest users i among the input users in $\mathcal{I}$.

From the signature verification on the output tuples, $C'_{\Pi(i)}$ is signed under $\mathsf{vk}'_{\Pi(i)}$ in $\sigma'_{\Pi(i),1}$, for every i: $e(\sigma'_{\Pi(i),1}, \mathfrak{g}'_0) = e(g, \mathfrak{f}_i^\alpha) \cdot e(\ell'_{\Pi(i)}, \mathfrak{l}_i^\alpha \mathfrak{g}_0^{\alpha\delta_i}) \cdot e(a'_{\Pi(i)}, \mathfrak{g}_i^\alpha) \cdot e(b'_{\Pi(i)}, \mathfrak{h}_i^\alpha)$, and since the same α appears in $\mathfrak{g}'_0 = \mathfrak{g}_0^\alpha$, then for every i, we have

$$\begin{aligned} e(\sigma'_{\Pi(i)}, \mathfrak{g}_0) &= e(g, \mathfrak{f}_i) \cdot e(\ell'_{\Pi(i)}, \mathfrak{l}_i \mathfrak{g}_0^{\delta_i}) \cdot e(a'_{\Pi(i)}, \mathfrak{g}_i) \cdot e(b'_{\Pi(i)}, \mathfrak{h}_i) \\ &= e(g, \mathfrak{f}_i) \cdot e(\ell'_{\Pi(i)}, \mathfrak{l}_i) \cdot e(a'_{\Pi(i)}, \mathfrak{g}_i) \cdot e(b'_{\Pi(i)}, \mathfrak{h}_i) \cdot e({\ell'}^{\delta_i}_{\Pi(i)}, \mathfrak{g}_0) \end{aligned}$$

and so $\sigma'_{\Pi(i)} / {\ell'}^{\delta_i}_{\Pi(i)}$ is a signature of $\overline{C}'_{\Pi(i)} = (g, \ell'_{\Pi(i)}, a'_{\Pi(i)}, b'_{\Pi(i)})$ under vk_i: under the unforgeability assumption of the signature scheme, $C'_{\Pi(i^*)}$ is necessarily a linear combination of the already signed vectors under vk_{i^*}, which are C_{i^*} and C_0, with some coefficients u, v: $a'_{\Pi(i^*)} = a_{i^*}^u g^v$, $b'_{\Pi(i^*)} = b_{i^*}^u h^v$, and $g = g^u 1^v$. Hence, $u = 1$, which means that $C'_{\Pi(i^*)}$ is a randomization of C_{i^*}.

We stress that for this property to hold, each key vk_i must appear at most once in the ballots, otherwise some combinations would be possible. Hence the test that all the $\mathfrak{f}_i$'s are distinct in the input ballot-box. □

We stress that this proposition only guarantees permutation of ciphertexts for honest users. There is indeed no formal guarantee for corrupted users whose signing keys are under the control of a mix-server. The latter could indeed replace the ciphertexts of some corrupted users, by some other ciphertexts under the same identity or even under the identity of another corrupted user. One can note that replacing ciphertexts (and plaintexts) even for corrupted users are not that easy because of the additional Diffie-Hellman proof on the ciphertexts, which implies $\prod_{\mathcal{I}} M_i = \prod_{\mathcal{J}} M'_i$. However, this property is more for the privacy, as we will see below. As a consequence, our result that guarantees a permutation on the honest ballots is optimal. We cannot guarantee anything for the users that share their keys with the mix-servers.

5.2 Proof of Privacy: Unlinkability

After proving the soundness, we have to prove the anonymity (a.k.a. unlinkability), which can also be seen as zero-knowledge property. More precisely, as for the soundness, privacy will only be guaranteed for honest users.

Definition 14 (Privacy for Honest Users). A mix-net M is said to provide *privacy for honest users* in the certified key setting, if any PPT adversary $\mathcal{A}$ has a negligible advantage in guessing b in the following security game:

1. the challenger generates the certification keys $(\mathsf{SK}, \mathsf{VK})$ and the encryption keys $(\mathsf{DK}, \mathsf{EK})$;
2. the adversary $\mathcal{A}$ then
   - decides on the corrupted users $\mathcal{I}^*$ and generates itself their keys $(\mathsf{vk}_i)_{i \in \mathcal{I}^*}$;
   - proves its knowledge of the secrete keys to get the certifications Σ_i on vk_i, for $i \in \mathcal{I}^*$;
   - decides on the corrupted mix-servers $\mathcal{J}^*$ and generates itself their keys $(\mathsf{VK}_j)_{j \in \mathcal{J}^*}$;
   - decides on the set $\mathcal{J}$ of the (honest and corrupted) mix-servers that will make mixes;
   - decides on the set $\mathcal{I}$ of the (honest and corrupted) users that will generate a ballot;
   - generates the ballots $(\mathcal{B}_i)_{i \in \mathcal{I}^*}$ for the corrupted users but provides the messages $(M_i)_{i \in \mathcal{I} \backslash \mathcal{I}^*}$ for the honest users;
3. the challenger generates the keys of the honest mix-servers $(\mathsf{SK}_j, \mathsf{VK}_j)_{j \in \mathcal{J} \backslash \mathcal{J}^*}$ the keys of the honest users $(\mathsf{sk}_i, \mathsf{vk}_i)_{i \in \mathcal{I} \backslash \mathcal{I}^*}$ and their ballots $(\mathcal{B}_i)_{i \in \mathcal{I} \backslash \mathcal{I}^*}$.

The initial ballot-box is thus defined by $\mathcal{BB}\mathsf{ox} = (\mathcal{B}_i)_{i \in \mathcal{I}}$. The challenger randomly chooses a bit $b \stackrel{\$}{\leftarrow} \{0, 1\}$ and then enters into a loop for $j \in \mathcal{J}$ with the attacker:

- let $\mathcal{I}^*_{j-1}$ be the set of indices of the ballots of the corrupted users in the input ballot-box $\mathcal{BB}\mathsf{ox}^{(j-1)}$;

- if $j \in \mathcal{J}^*$, $\mathcal{A}$ builds itself the new ballot-box $\mathcal{BB}\mathsf{ox}^{(j)}$ with the proof $\mathsf{proof}^{(j)}$;
- if $j \notin \mathcal{J}^*$, $\mathcal{A}$ provides two permutations $\Pi_{j,0}$ and $\Pi_{j,1}$ of its choice, with the restriction they must be identical on $\mathcal{I}^*_{j-1}$, then the challenger runs the mixing with $\Pi_{j,b}$, and provides the output $(\mathcal{BB}\mathsf{ox}^{(j)}, \mathsf{proof}^{(j)})$;

In the end, the adversary outputs its guess b' for b. The experiment outputs 1 if $b' = b$ and 0 otherwise.

Contrarily to the soundness security game, the adversary can see the outputs of all the mixing steps to make its decision, hence the index j for the mix-servers. In addition, some can be honest, some can be corrupted. We will assume at least one is honest.

Theorem 15. *Our Mix-Net protocol provides privacy for honest users, in the certified key setting, if (at least) one mix-server is honest, under our unlinkability assumption (see Definition 4), and the* DDH *assumptions in both* $\mathbb{G}_1$ *and* $\mathbb{G}_2$*.*

Proof. This proof will follow a series of games $(\mathbf{G}_i)_i$, where we study the advantage Adv_i of the adversary in guessing b. We start from the real security game and conclude with a game where all the ballots are random, independently from the permutations. Hence, the advantage will be trivially 0.

Game $\mathbf{G}_0$: This is the real game, where the challenger (our simulator) generates SK and VK for the certification authority signature, and randomly chooses $d \stackrel{\$}{\leftarrow} \mathbb{Z}_p$ to generate the encryption public key $\mathsf{EK} = h = g^d$. One also sets $\mathsf{vk}_0 = (1, 1, \mathfrak{g}_0 = \mathfrak{g}^A, 1, 1)$ and $C_0 = \mathsf{Encrypt}_{\mathsf{EK}}(1) = (g, h)$ expanded into $\overline{C}_0 = (1, \ell, C_0)$ with the noise parameter $\ell \stackrel{\$}{\leftarrow} \mathbb{G}_1$. Actually, $A = 1$ in the initial step, when the user encrypts his message M_i, but since the shuffling may happens after several other shuffling iterations, we have the successive exponentiations to multiple α (in A) for vk_0. The attacker $\mathcal{A}$ chooses the set of the initial indices of the corrupted users $\mathcal{I}^*$ and the set of the initial indices of the corrupted mix-servers $\mathcal{J}^*$, provides their verification keys $((\mathsf{vk}_i)_{i \in \mathcal{I}^*}, (\mathsf{VK}_j)_{j \in \mathcal{J}^*})$ together with an extractable zero-knowledge proof of knowledge of sk_i.
From $\mathcal{I}$ and $\mathcal{J}$, one generates the signing keys for the honest mix-servers $j \in \mathcal{J} \backslash \mathcal{J}^*$, and set J to the index of the last honest mix-server. For each $i \in \mathcal{I}$, one chooses $\tau_i = R_i \stackrel{\$}{\leftarrow} \mathbb{Z}_p$ and sets $\tau_i = (\tau_{i,1} = \mathfrak{g}^{1/R_i}, \tau_{i,2} = g^{1/R_i})$. For each honest user $i \in \mathcal{I} \backslash \mathcal{I}^*$, one randomly chooses $u_i, v_i, x_i, y_i, r_i, \rho_i \stackrel{\$}{\leftarrow} \mathbb{Z}_p$ to generate $\mathsf{vk}_i = (\mathfrak{g}_0 = \mathfrak{g}, \mathfrak{f}_i = \mathfrak{g}_0^{u_i}, \mathfrak{l}_i = \mathfrak{g}_0^{v_i}, \mathfrak{g}_i = \mathfrak{g}_0^{x_i}, \mathfrak{h}_i = \mathfrak{g}_0^{y_i})$, and eventually generates all the signatures Σ_i of $(\mathsf{vk}_i, \mathsf{vk}_0)$ under SK with respect to the tag τ_i (using SK and $(\tilde{\tau}_i)_i$).
For the corrupted users, the simulator directly receives the ballots $(\mathcal{B}_i = (\overline{C}_i, \sigma_i, \mathsf{vk}_i, \Sigma_i, \tau_i))_{i \in \mathcal{I}^*}$ while for the honest users, it receives $(M_i)_{i \in \mathcal{I} \backslash \mathcal{I}^*}$ and computes $C_i = \mathsf{Encrypt}_{\mathsf{EK}}(M_i) = (a_i = g^{r_i}, b_i = h^{r_i} M_i)$, $\overline{C}_i = (g, \ell_i = \ell^{\rho_i}, C_i)$ and the signature σ_i of $(\overline{C}_i, \overline{C}_0)$ under sk_i. The input ballot-box is then $\mathcal{BB}\mathsf{ox}^{(0)} = \{(\mathcal{B}_i)_{i \in \mathcal{I}}\}$ including the ballots of the honest and corrupted users. Let $\mathcal{I}^*_0 = \mathcal{I}^*$ be the set of the initial indices of the corrupted users.
The simulator randomly chooses $b \stackrel{\$}{\leftarrow} \{0, 1\}$ and now begins the loop of the mixes: depending if the mix-server j is corrupted or not, the simulator directly receives $(\mathcal{BB}\mathsf{ox}^{(j)}, \mathsf{proof}^{(j)})$ from the adversary or receives $(\Pi_{j,0}, \Pi_{j,1})$. In the latter case, one first checks if $\Pi_{j,0}|_{\mathcal{I}^*_{j-1}} = \Pi_{j,1}|_{\mathcal{I}^*_{j-1}}$ using the honest secret keys to determine $\mathcal{I}^*_{j-1}$. Then, the simulator randomly chooses global $\alpha \stackrel{\$}{\leftarrow} \mathbb{Z}_p$ and individual $\gamma_i, \delta_i, \mu_i \stackrel{\$}{\leftarrow} \mathbb{Z}_p$ for all the users, as an honest mix-

server would do, to compute

$$\begin{aligned}
\mathsf{vk}'_i &= (\mathfrak{g}'_0 = \mathfrak{g}_0^\alpha, \mathfrak{f}'_i = \mathfrak{f}_i^\alpha, \mathfrak{l}'_i = (\mathfrak{l}_i \cdot \mathfrak{g}_0^{\delta_i})^\alpha, \mathfrak{g}'_i = \mathfrak{g}_i^\alpha, \mathfrak{h}'_i = \mathfrak{h}_i^\alpha) = (\mathsf{vk}_i \cdot \mathsf{vk}_0^{\delta_i})^\alpha \\
\mathsf{vk}'_0 &= (1, 1, \mathfrak{g}'_0, 1, 1) = \mathsf{vk}_0^\alpha \\
\overline{C}'_i &= (g, \ell'_i = \ell_i \cdot \ell_0^{\gamma_i}, a'_i = a_i \cdot g_0^{\gamma_i}, b'_i = b_i \cdot h_0^{\gamma_i}) = \overline{C}_i \cdot \overline{C}_0{}^{\gamma_i} \\
\sigma'_i &= (\sigma'_{i,0} = \sigma_{i,0} \cdot {\ell'_0}^{\delta_i}, \sigma'_{i,1} = \sigma_{i,1} \cdot \sigma_{i,0}^{\gamma_i} \cdot {\ell'_i}^{\delta_i}) \\
\Sigma'_i &= (\Sigma'_{i,0} = \Sigma_{i,0}^{\alpha\mu_i}, \Sigma'_{i,1} = (\Sigma_{i,1} \cdot \Sigma_{i,0}^{\delta_i})^{\alpha\mu_i}) \\
\tau'_i &= (\tau'_{i,1} = \tau_{i,1}^{1/\mu_i}, \tau'_{i,2} = \tau_{i,2}^{1/\mu_i})
\end{aligned}$$

and sets $\mathcal{BB}\mathsf{ox}^{(j)} = (\mathcal{B}'_{\Pi_{j,b}(i)})_i$. Eventually, the simulator computes the proof $\mathsf{proof}^{(j)}$ for $(\mathfrak{g}_0, \mathfrak{g}'_0, \prod \mathfrak{f}_i, \prod \mathfrak{f}'_i)$ and $(g, h, \prod a'_i / \prod a_i, \prod b'_i / \prod b_i)$, and signs it using SK_j.
After the full loop on all the mix-servers, the adversary outputs its guess b': $\mathsf{Adv}_{\mathbf{G}_0} = \Pr_{\mathbf{G}_0}[b' = b]$. One important remark is that under the previous soundness result, which has exactly the same setup, the input ballot-box for the last honest mix-server necessarily contains a randomization of the initial honest ballots (the adversary against the soundness is the above adversary together with the honest simulator up to its last honest round, that does not need any secret). Only the behavior of this last honest mix-server will be modified below.

Game $\mathbf{G}_1$: We first switch the Diffie-Hellman proofs for $(\mathfrak{g}_0, \mathfrak{g}'_0, \prod \mathfrak{f}_i, \prod \mathfrak{f}'_i)$ to the zero-knowledge setting: if the input ballot-box for the last honest mix-server is not a randomization of the initial honest ballots, that can be tested using the decryption key, one has built a distinguisher between the settings of the zero-knowledge proofs. In this new setting, one can use the zero-knowledge simulator that does not use α. Under the zero-knowledge property, $\mathsf{Adv}_{\mathbf{G}_0} < \mathsf{Adv}_{\mathbf{G}_1} + \mathsf{negl}()$.

Game $\mathbf{G}_2$: We also switch the proofs for $(g, h, \prod a'_i / \prod a_i, \prod b'_i / \prod b_i)$ to the zero-knowledge setting: as above, the distance remains negligible. In this new setting, one can use the zero-knowledge simulator that does not use $\sum_i \gamma_i$. Under the zero-knowledge property, $\mathsf{Adv}_{\mathbf{G}_1} < \mathsf{Adv}_{\mathbf{G}_2} + \mathsf{negl}()$.

Game $\mathbf{G}_3$: In this game, we do not know anymore the decryption key, and use the indistinguishability of the encryption scheme (which relies on the Decisional Diffie-Hellman assumption): in an hybrid way, we replace the ciphertexts C_i of the honest users by an encryption of 1: $C_i = \mathsf{Encrypt}_{\mathsf{EK}}(1)$. Under the DDH assumption in $\mathbb{G}_1$, $\mathsf{Adv}_{\mathbf{G}_2} < \mathsf{Adv}_{\mathbf{G}_3} + \mathsf{negl}()$.

Game $\mathbf{G}_4$: This corresponds to $C_i = (a_i = g^{r_i}, b_i = h^{r_i})$. But now we can know d, but ℓ is random: under the DDH assumption, we can replace the random value $\ell_i = \ell^{\rho_i}$ by $\ell_i = \ell^{r_i}$. Ultimately, we set $\overline{C}_i = (g, \ell_i = \ell^{r_i}, a_i = g^{r_i}, b_i = h^{r_i})$ for $r_i \overset{\$}{\leftarrow} \mathbb{Z}_p$, for all the honest users, in the initial ballot-box. Under the DDH assumption in $\mathbb{G}_1$, $\mathsf{Adv}_{\mathbf{G}_3} < \mathsf{Adv}_{\mathbf{G}_4} + \mathsf{negl}()$.

Game $\mathbf{G}_5$: In this game, one can first extract the keys of the corrupted users during the certification phase. Then, all the honest mix-servers generate random signing keys sk'_i, random tags τ'_i, and random encryptions C'_i of 1, for all the honest users (the one who do not correspond to the extracted keys), and generate the signatures using the signing keys SK and sk'_i, but still behave honestly for the ballots of the corrupted users. Then, they apply the permutations $\Pi_{j,b}$ on the randomized ballots.

Lemma 16 (Random Ballots for Honest Users). *Under the Unlinkability Assumption (see Definition 4) and* DDH *assumption in* $\mathbb{G}_2$*, the view is computationally indistinguishable:* $\mathsf{Adv}_{\boldsymbol{G}_4} < \mathsf{Adv}_{\boldsymbol{G}_5} + \mathsf{negl}()$*.*

In this last game, the i-th honest user is simulated with initial and output (after each honest mix-server) ciphertexts that are random encryptions of 1, and initial and output signing keys (and thus verification keys vk_i and vk'_i) independently random. As a consequence, permutations $\Pi_{j,b}$ are applied on random ballots, which is perfectly indistinguishable from applying $\Pi_{j,1-b}$

(as we have restricted the two permutations to be identical on ballots of corrupted users): $\mathsf{Adv}_{\mathbf{G}_5} = 0$. Which leads to $\mathsf{Adv}_0 \leq \mathsf{negl}()$. □

Proof of Lemma 16. In the above sequences of games, from $\mathbf{G}_0$ to $\mathbf{G}_4$, we could have checked whether the honest vk_i's in the successive ballot-boxes are permutations of randomized honest initial keys, just using the secret keys of the honest users. So, we can assume in the next hybrid games, from $\mathbf{G}_0(j)$ to $\mathbf{G}_8(j)$, for $j = N, \dots, 1$ that the input ballots in $\mathcal{BB}\mathsf{ox}^{(j-1)}$ contain proper permutations of randomized honest initial keys, as nothing is modified before the generation of this ballot-box. In the following series of hybrid games, for index j, the honest mix-servers up to the $j-1$-th round play as in $\mathbf{G}_4$ and from the $j+1$-th round, they play as in $\mathbf{G}_5$. Only the behavior of the j-th mix-server is modified: starting from an honest behavior. Hence, $\mathbf{G}_0(N) = \mathbf{G}_4$.

Game $\mathbf{G}_0(j)$: In this hybrid game, we assume that the initial ballot-box has been correctly generated (with $\overline{C}_i = (g, \ell_i = \ell^{r_i}, a_i = g^{r_i}, b_i = h^{r_i})$ for $r_i \overset{\$}{\leftarrow} \mathbb{Z}_p$, for all the honest users), and mixing steps up to $\mathcal{BB}\mathsf{ox}^{(j)}$ have been honestly generated (excepted the zero-knowledge proofs that have been simulated). The next rounds are generated at random by honest mix-servers: random signing keys sk'_i and random ciphertexts $\overline{C}'_i = (g, \ell'_i = \ell^{r'_i}, a'_i = g^{r'_i}, b'_i = h^{r'_i})$, with random r'_i, and then correct signatures, using SK and sk'_i. The following sequence of games will modify the randomization of $\mathcal{BB}\mathsf{ox}^{(j-1)}$ into $\mathcal{BB}\mathsf{ox}^{(j)}$ if the j-th mix-server is honest.

Game $\mathbf{G}_1(j)$: We now start modifying the randomization of the ballots by the j-th mix-server, for the corrupted users. As we assumed the signatures Σ_i provided by the certification authority from a proof of knowledge of sk_i, our simulator has access to $\mathsf{sk}_i = (u_i, v_i, x_i, z_i)$ for all the corrupted users. The mixing step consists in updating the ciphertexts, the keys and the signatures, and we show how to do it without using α such that $\mathfrak{g}'_0 = \mathfrak{g}_0^\alpha$ but, instead, just $\mathfrak{g}'_0$, sk_i, $\overline{C}_0 = (1, \ell, g, h)$ and the individual random coins γ_i, δ_i: from $\mathcal{B}_i$ a received ballot of a corrupted user, one can compute $\mathsf{vk}'_i = (\mathfrak{g}'_0, {\mathfrak{g}'_0}^{u_i}, {\mathfrak{g}'_0}^{v_i+\delta_i}, {\mathfrak{g}'_0}^{x_i}, {\mathfrak{g}'_0}^{y_i})$ and $\overline{C}'_i = \overline{C}_i \cdot \overline{C}_0^{\gamma_i}$, and then the signatures σ'_i and Σ'_i using the signing keys, and choosing $\tilde{\tau}'_i \overset{\$}{\leftarrow} \mathbb{Z}_p$. This simulation is perfect for the corrupted users: $\mathsf{Adv}_{\mathbf{G}_1(j)} = \mathsf{Adv}_{\mathbf{G}_0(j)}$.

Game $\mathbf{G}_2(j)$: We now modify the simulation of the honest ballots. In this game, we choose random $d, e \overset{\$}{\leftarrow} \mathbb{Z}_p$ for $h = g^d$ and $\ell = g^e$. Then we have simulated $\overline{C}_i = (g, \ell_i = \ell^{r_i}, a_i = g^{r_i}, b_i = h^{r_i})$ the ciphertext in $\mathcal{BB}\mathsf{ox}^{(0)}$ and we can set $\overline{C}'_i = (g, \ell'_i = \ell^{r'_i}, a'_i = g^{r'_i}, b'_i = h^{r'_i})$ the ciphertext in $\mathcal{BB}\mathsf{ox}^{(j)}$ for known random scalars $r_i, r'_i \overset{\$}{\leftarrow} \mathbb{Z}_p$, where r'_i is actually $r_i + \gamma_i$: γ_i is the accumulation of all the noises. All the signatures are still simulated using the signing keys (and $\tilde{\tau}'_i = R'_i \overset{\$}{\leftarrow} \mathbb{Z}_p$), with $\mathfrak{g}'_0 = \mathfrak{g}_0^\alpha$ for a random scalar α. This simulation is perfectly the same as above: $\mathsf{Adv}_{\mathbf{G}_2(j)} = \mathsf{Adv}_{\mathbf{G}_1(j)}$.

Before continuing, we study the format of the initial and randomized ballots: by denoting σ_i the initial signature in $\mathcal{BB}\mathsf{ox}^{(0)}$ and σ'_i the signature to generate in $\mathcal{BB}\mathsf{ox}^{(j)}$, we have the following relations:

$$e(\sigma_{i,0}, \mathfrak{g}_0) = e(g, \mathfrak{g}_i {\mathfrak{h}_i}^d {\mathfrak{l}_i}^e) \qquad e(\sigma_{i,1}, \mathfrak{g}_0) = e(g, \mathfrak{f}_i(\mathfrak{g}_i {\mathfrak{h}_i}^d {\mathfrak{l}_i}^e)^{r_i})$$
$$e(\sigma'_{i,0}, \mathfrak{g}'_0) = e(g, \mathfrak{g}'_i {\mathfrak{h}'_i}^d {\mathfrak{l}'_i}^e) \qquad e(\sigma'_{i,1}, \mathfrak{g}'_0) = e(g, \mathfrak{f}'_i(\mathfrak{g}'_i {\mathfrak{h}'_i}^d {\mathfrak{l}'_i}^e)^{r'_i})$$

If we formally denote $\sigma_{i,0} = g^{t_i}$ and $\sigma_{i,1} = g^{s_i}$, then we have

$$ {\mathfrak{g}_0}^{t_i} = \mathfrak{g}_i {\mathfrak{h}_i}^d {\mathfrak{l}_i}^e \text{ and } {\mathfrak{g}_0}^{s_i} = \mathfrak{f}_i(\mathfrak{g}_i {\mathfrak{h}_i}^d {\mathfrak{l}_i}^e)^{r_i} = \mathfrak{f}_i {\mathfrak{g}_0}^{t_i r_i}$$

which implies $s_i = u_i + t_i r_i$. Similarly, if we formally denote $\sigma'_{i,0} = {g'}^{t'_i}$ and $\sigma'_{i,1} = g^{s'_i}$, and set α as the product of all the α's and δ_i as aggregation of all the δ_i's (with α's) in the previous rounds plus this round, from

$$ {\mathfrak{g}_0}^{\alpha t'_i} = {\mathfrak{g}'_0}^{t'_i} = \mathfrak{g}'_i {\mathfrak{h}'_i}^d {\mathfrak{l}'_i}^e = {\mathfrak{g}_i}^\alpha {\mathfrak{h}_i}^{\alpha d} (\mathfrak{l}_i \mathfrak{g}_0^{\delta_i})^{\alpha e}$$
$$ {\mathfrak{g}_0}^{\alpha s'_i} = {\mathfrak{g}'_0}^{s'_i} = \mathfrak{f}'_i(\mathfrak{g}'_i {\mathfrak{h}'_i}^d {\mathfrak{l}'_i}^e)^{r'_i} = \mathfrak{f}_i^\alpha (\mathfrak{g}_i^\alpha \mathfrak{h}_i^{\alpha d} (\mathfrak{l}_i \mathfrak{g}_0^{\delta_i})^{\alpha e})^{r'_i}$$

we also have ${\mathfrak{g}_0}^{t'_i} = (\mathfrak{g}_i{\mathfrak{h}_i}^d\mathfrak{l}_i^e)\mathfrak{g}_0^{\delta_i e}$ and ${\mathfrak{g}_0}^{s'_i} = \mathfrak{f}_i(\mathfrak{g}_i\mathfrak{h}_i^d\mathfrak{l}_i^e)^{r'_i}{\mathfrak{g}_0}^{e\delta_i r'_i}$ which implies $s'_i = u_i + t'_i r'_i$. As consequence:

$$\sigma_{i,1} = g^{u_i} \cdot (g^{r_i})^{t_i} = g^{u_i} \cdot {a_i}^{t_i} \text{ and } \sigma'_{i,1} = g^{u_i} \cdot (g^{r'_i})^{t'_i} = g^{u_i} \cdot {a'_i}^{t'_i}$$

Game $\mathbf{G}_3(j)$: Let us randomly choose scalars $u_i, r_i, r'_i, t_i, t'_i$ and α, then, from $(g, \mathfrak{g}_0)$, we can set $\mathfrak{g}'_0 \leftarrow \mathfrak{g}_0^\alpha$, $a_i \leftarrow g^{r_i}$, $\sigma_{i,1} \leftarrow a_i^{t_i} g^{u_i}$, $\mathfrak{f}_i \leftarrow \mathfrak{g}_0^{u_i}$, as well as $a'_i \leftarrow g^{r'_i}$, $\sigma'_{i,1} \leftarrow {a'_i}^{t'_i} g^{u_i}$, $\mathfrak{f}'_i \leftarrow {\mathfrak{g}'_0}^{u_i}$. Then, one additionally chooses $x_i, y_i \stackrel{\$}{\leftarrow} \mathbb{Z}_p$ and sets

$$\begin{array}{llll} \mathfrak{g}_i \leftarrow \mathfrak{g}_0^{x_i} & \mathfrak{h}_i \leftarrow \mathfrak{g}_0^{y_i} & \mathfrak{l}_i \leftarrow (\mathfrak{g}_0^{t_i}/(\mathfrak{g}_i\mathfrak{h}_i^d))^{1/e} & \overline{C}_i \leftarrow (g, a_i^e, a_i, a_i^d) \\ \mathfrak{g}'_i \leftarrow {\mathfrak{g}'_0}^{x_i} & \mathfrak{h}'_i \leftarrow {\mathfrak{g}'_0}^{y_i} & \mathfrak{l}'_i \leftarrow ({\mathfrak{g}'_0}^{t'_i}/(\mathfrak{g}'_i{\mathfrak{h}'_i}^d))^{1/e} & \overline{C}'_i \leftarrow (g, {a'_i}^e, a'_i, {a'_i}^d) \end{array}$$

By construction

$$\begin{array}{ll} \mathfrak{g}_0^{t_i} = \mathfrak{g}_i\mathfrak{h}_i^d\mathfrak{l}_i^e & {\mathfrak{g}'_0}^{t'_i} = \mathfrak{g}'_i{\mathfrak{h}'_i}^d{\mathfrak{l}'_i}^e \\ \sigma_{i,1} = a_i^{t_i} g^{u_i} = g^{t_i r_i} \times g^{u_i} & \sigma'_{i,1} = {a'_i}^{t'_i} g^{u_i} = g^{t'_i r'_i} \times g^{u_i} \end{array}$$

With $\sigma_{i,0} \leftarrow g^{t_i}$ and $\sigma'_{i,0} \leftarrow g^{t'_i}$, σ_i and σ'_i are valid signatures of $(\overline{C}_i, \overline{C}_0)$ and $(\overline{C}'_i, \overline{C}_0)$ respectively. Then, the verification keys $\mathsf{vk}_i = (\mathfrak{g}_0, \mathfrak{f}_i, \mathfrak{l}_i, \mathfrak{g}_i, \mathfrak{h}_i)$ and $\mathsf{vk}'_i = (\mathfrak{g}'_0, \mathfrak{f}'_i, \mathfrak{l}'_i, \mathfrak{g}'_i, \mathfrak{h}'_i)$ are correctly related for the secret keys (u_i, v_i, x_i, y_i). From $\mathfrak{l}_i = (\mathfrak{g}_0^{t_i}/(\mathfrak{g}_i\mathfrak{h}_i^d))^{1/e} = \mathfrak{g}_0^{(t_i - x_i - dy_i)/e}$: we have $v_i = (t_i - x_i - dy_i)/e$. From $\mathfrak{l}'_i = ({\mathfrak{g}'_0}^{t'_i}/(\mathfrak{g}'_i{\mathfrak{h}'_i}^d))^{1/e} = {\mathfrak{g}'_0}^{(t'_i - x_i - dy_i)/e}$: we have $v'_i = (t'_i - x_i - dy_i)/e = (t'_i - t_i)/e + v_i$, which means that $\delta_i = (t'_i - t_i)/e$.

Using the signing key SK, we can complete and sign vk_i (with random R_i) and vk'_i (with random R'_i, which implicitly defines μ_i). As shown above, this perfectly simulates the view of the adversary for the honest ballots in the initial ballot-box $\mathcal{BB}\mathsf{ox}^{(0)}$, with $\mathcal{B}_i = (\overline{C}_i, \sigma_i, \mathsf{vk}_i, \Sigma_i, \tau_i)$ and a randomized version in the updated ballot-box $\mathcal{BB}\mathsf{ox}^{(j)}$, with $\mathcal{B}'_i = (\overline{C}'_i, \sigma'_i, \mathsf{vk}'_i, \Sigma'_i, \tau'_i)$: $\mathsf{Adv}_{\mathbf{G}_3(j)} = \mathsf{Adv}_{\mathbf{G}_2(j)}$.

Game $\mathbf{G}_4(j)$: Let us be given $\mathsf{Cred}(u_i, g; \mathfrak{g}_0, r_i, t_i)$ and $\mathsf{Cred}(u_i, g; \mathfrak{g}'_0, r'_i, t'_i)$, for random $u_i \stackrel{\$}{\leftarrow} \mathbb{Z}_p$, which provide all the required inputs from the first part of the simulation in the previous game (before choosing x_i, y_i). They all follow the distribution $\mathcal{D}_{g,\mathfrak{g}_0}(u_i, u_i)$. As we do not need to know α to randomize ballots for corrupted users, we can thus continue the simulation as above, in a perfectly indistinguishable way: $\mathsf{Adv}_{\mathbf{G}_4(j)} = \mathsf{Adv}_{\mathbf{G}_3(j)}$.

Game $\mathbf{G}_5(j)$: Let us be given two credentials of u_i and u'_i, $\mathsf{Cred}(u_i, g; \mathfrak{g}_0, r_i, t_i)$ and $\mathsf{Cred}(u'_i, g; \mathfrak{g}'_0, r'_i, t'_i)$, for random $u_i, u'_i \stackrel{\$}{\leftarrow} \mathbb{Z}_p$. Inputs follow the distribution $\mathcal{D}_{g,\mathfrak{g}_0}(u_i, u'_i)$ and we do as above. Under the Unlinkability Assumption (see Definition 4) the view is computationally indistinguishable: $\mathsf{Adv}_{\mathbf{G}_4(j)} < \mathsf{Adv}_{\mathbf{G}_5(j)} + \mathsf{negl}()$.

Game $\mathbf{G}_6(j)$: We receive a Multi Diffie-Hellman tuple $(\mathfrak{g}_0, \mathfrak{g}_i, \mathfrak{h}_i, \mathfrak{g}'_0, \mathfrak{g}'_i, \mathfrak{h}'_i) \stackrel{\$}{\leftarrow} \mathcal{D}^6_{\mathsf{mdh}}(\mathfrak{g}_0)$. So we know all the scalars, except x_i, y_i and α, which are implicitly defined by the input challenge. Then, by choosing $t_i, t'_i \stackrel{\$}{\leftarrow} \mathbb{Z}_p$, we can define $\mathfrak{l}_i, \mathfrak{l}'_i$ as in the previous game, and the ciphertexts and signatures are generated honestly with random scalars $r_i, r'_i \stackrel{\$}{\leftarrow} \mathbb{Z}_p$: $\mathsf{Adv}_{\mathbf{G}_6(j)} = \mathsf{Adv}_{\mathbf{G}_5(j)}$.

Game $\mathbf{G}_7(j)$: We now receive $(\mathfrak{g}_0, \mathfrak{g}_i, \mathfrak{h}_i, \mathfrak{g}'_0, \mathfrak{g}'_i, \mathfrak{h}'_i) \stackrel{\$}{\leftarrow} \mathcal{D}^6_{\$}(\mathfrak{g}_0)$. We do the simulation as above. The view of the adversary is indistinguishable under the DDH assumption in $\mathbb{G}_2$: $\mathsf{Adv}_{\mathbf{G}_6(j)} < \mathsf{Adv}_{\mathbf{G}_7(j)} + \mathsf{negl}()$.

In this game, $\mathsf{vk}'_i = (\mathfrak{g}'_0, \mathfrak{f}_i = {\mathfrak{g}'_0}^{u'_i}, \mathfrak{l}_i = {\mathfrak{g}'_0}^{v'_i}, \mathfrak{g}_i = {\mathfrak{g}'_0}^{x'_i}, \mathfrak{h}_i = {\mathfrak{g}'_0}^{y'_i})$, with $x'_i, y'_i \stackrel{\$}{\leftarrow} \mathbb{Z}_p$ because of the random tuple, $v'_i = v_i + (t'_i - t_i)/e$, for random t'_i and t_i, it is thus also random, and u'_i is chosen at random.

Game $\mathbf{G}_8(j)$: We now choose at random the signing keys $\mathsf{sk}_i = (u_i, v_i, x_i, y_i)$ and $\mathsf{sk}'_i = (u'_i, v'_i, x'_i, y'_i)$ in order to sign the ciphertexts: $\mathsf{Adv}_{\mathbf{G}_8(j)} = \mathsf{Adv}_{\mathbf{G}_7(j)}$.

With this last game, one can see that $\mathbf{G}_8(1) = \mathbf{G}_5$. Furthermore, for each round $j = N, \ldots, 1$, we have $\mathsf{Adv}_{\mathbf{G}_0(j)} \leq \mathsf{Adv}_{\mathbf{G}_8(j)} + \mathsf{negl}()$, while $\mathbf{G}_0(j-1) = \mathbf{G}_8(j)$: $\mathsf{Adv}_{\mathbf{G}_4} = \mathsf{Adv}_{\mathbf{G}_0}(N) \leq \mathsf{Adv}_{\mathbf{G}_8}(1) + \mathsf{negl}() = \mathsf{Adv}_{\mathbf{G}_5} + \mathsf{negl}()$. □

6 Applications

We now discuss use-cases of mix-nets: electronic voting and anonymous routing. In both cases, a mix-server can, on the fly, perform individual verifications and randomization of ballots, as well as the product of the $\mathfrak{f}_i$'s and the ciphertexts adaptively until the ballots are all sent. Eventually, at the closing time for a vote or at the end of a time lapse for routing, one just has to do and sign global proof of Diffie-Hellman tuples, and then output the ballots in a permuted order.

6.1 Electronic Voting

Our mix-net fits well the case of e-voting because after the multiple mixing steps, all the mix-servers can perform a second round to sign in a compact way the constant-size proof, certifying each of their contributions. The input size as well as the computation cost of the verifier are both independent on the number of mixing steps. To our knowledge it is the first scheme with this very nice property.

About security, as explained, soundness and privacy are guaranteed for the honest users only: honest users are sure that their votes are randomized in the output ballot-box, and their input-output ballots are unlinkable. This is of course the most important requirements. However, since the u_i's are used to guarantee that no ballots are deleted or inserted, this is important those values to be unknown to the mix-server.

In the Appendix C, we propose a second construction that uses Square Diffie-Hellman tuples $(\mathfrak{g}_r, \mathfrak{A}_i = \mathfrak{g}_r^{w_i}, \mathfrak{B}_i = \mathfrak{A}_i^{w_i})$ as tags to add in any one-time linearly homomorphic signature to obtain a linearly homomorphic signature with randomizable tags. Then, one can use $\prod \mathfrak{A}'_j = (\prod \mathfrak{A}_i)^\alpha$ instead of $\prod \mathfrak{f}'_j$ and $(\prod \mathfrak{f}_i)^\alpha$, in the Diffie-Hellman tuple, to guarantee the permutation of the verification keys. Only the privacy of the w_i's is required to guarantee the soundness.

The proof that $\prod M_i = \prod M'_i$ is actually never used in the previous security proofs, as it counts for privacy in e-voting only. Indeed, in our privacy security game we let the adversary choose the messages of the honest users. In a voting scheme, the adversary could not choose them and would like to learn the vote of a target voter. The first mix-server could take the vote (ciphertext) of this voter and ask several corrupted voters to duplicate this vote. The bias in the tally would reveal the vote of the target voter: the proof on the products of the plaintexts avoids this modification during the mixing. This does not exclude the attack of Cortier-Smyth [CS13] if the votes are publicly sent, as the corrupted voters could simply use the ciphertext for their own ballots.

6.2 Message Routing

Another important use case of mix-nets is in routing protocols where the mix-servers are proxy servers guaranteeing that no one can trace a request of a message. In this scenario, it is not possible to perform a second round on the mix-servers to obtain the multi-signature and the efficiency is thus linear in the number of mixing steps. It is still an open problem to avoid the second round while maintaining the independence in the number of mix-servers.

Acknowledgments

This work was supported in part by the European Community's Seventh Framework Programme (FP7/2007-2013 Grant Agreement no. 339563 – CryptoCloud).

References

ABC$^+$12. Jae Hyun Ahn, Dan Boneh, Jan Camenisch, Susan Hohenberger, abhi shelat, and Brent Waters. Computing on authenticated data. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 1–20. Springer, Heidelberg, March 2012.

AFG$^+$10. Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Structure-preserving signatures and commitments to group elements. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 209–236. Springer, Heidelberg, August 2010.

BBG05. Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 440–456. Springer, Heidelberg, May 2005.

BDN18. Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 435–464. Springer, Heidelberg, December 2018.

BF11a. Dan Boneh and David Mandell Freeman. Homomorphic signatures for polynomial functions. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 149–168. Springer, Heidelberg, May 2011.

BF11b. Dan Boneh and David Mandell Freeman. Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *PKC 2011*, volume 6571 of *LNCS*, pages 1–16. Springer, Heidelberg, March 2011.

BFI$^+$10. Olivier Blazy, Georg Fuchsbauer, Malika Izabachène, Amandine Jambert, Hervé Sibert, and Damien Vergnaud. Batch Groth-Sahai. In Jianying Zhou and Moti Yung, editors, *ACNS 10*, volume 6123 of *LNCS*, pages 218–235. Springer, Heidelberg, June 2010.

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.

BFPV11. Olivier Blazy, Georg Fuchsbauer, David Pointcheval, and Damien Vergnaud. Signatures on randomizable ciphertexts. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *PKC 2011*, volume 6571 of *LNCS*, pages 403–422. Springer, Heidelberg, March 2011.

BG12. Stephanie Bayer and Jens Groth. Efficient zero-knowledge argument for correctness of a shuffle. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 263–280. Springer, Heidelberg, April 2012.

Boy08. Xavier Boyen. The uber-assumption family (invited talk). In Steven D. Galbraith and Kenneth G. Paterson, editors, *PAIRING 2008*, volume 5209 of *LNCS*, pages 39–56. Springer, Heidelberg, September 2008.

Cha81. David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90, February 1981.

CS13. Véronique Cortier and Ben Smyth. Attacking and fixing helios: An analysis of ballot secrecy. *J. Comput. Secur.*, 21(1):89–148, January 2013.

Dam92. Ivan Damgård. Towards practical public key systems secure against chosen ciphertext attacks. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 445–456. Springer, Heidelberg, August 1992.

FHS19. Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. Structure-preserving signatures on equivalence classes and constant-size anonymous credentials. *Journal of Cryptology*, 32(2):498–546, April 2019.

FLSZ17. Prastudy Fauzi, Helger Lipmaa, Janno Siim, and Michal Zajac. An efficient pairing-based shuffle argument. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part II*, volume 10625 of *LNCS*, pages 97–127. Springer, Heidelberg, December 2017.

Fre12. David Mandell Freeman. Improved security for linearly homomorphic signatures: A generic framework. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 697–714. Springer, Heidelberg, May 2012.

FS87. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.

FS01. Jun Furukawa and Kazue Sako. An efficient scheme for proving a shuffle. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 368–387. Springer, Heidelberg, August 2001.

GI08. Jens Groth and Yuval Ishai. Sub-linear zero-knowledge argument for correctness of a shuffle. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 379–396. Springer, Heidelberg, April 2008.

GL07. Jens Groth and Steve Lu. A non-interactive shuffle with pairing based verifiability. In Kaoru Kurosawa, editor, *ASIACRYPT 2007*, volume 4833 of *LNCS*, pages 51–67. Springer, Heidelberg, December 2007.

Gro10. Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 321–340. Springer, Heidelberg, December 2010.

GS08. Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 415–432. Springer, Heidelberg, April 2008.

HT98. Satoshi Hada and Toshiaki Tanaka. On the existence of 3-round zero-knowledge protocols. In Hugo Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 408–423. Springer, Heidelberg, August 1998.

JMSW02. Robert Johnson, David Molnar, Dawn Xiaodong Song, and David Wagner. Homomorphic signature schemes. In Bart Preneel, editor, *CT-RSA 2002*, volume 2271 of *LNCS*, pages 244–262. Springer, Heidelberg, February 2002.

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.

Nef01. C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. In Michael K. Reiter and Pierangela Samarati, editors, *ACM CCS 2001*, pages 116–125. ACM Press, November 2001.

Sho97. Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 256–266. Springer, Heidelberg, May 1997.

Supplementary Material

A Some Primitives

A.1 Groth-Sahai Proofs

In this section, we recall the Groth-Sahai methodology [GS08] to prove a Diffie-Hellman tuple, how the proof can be verified, but also how one can update the proof.

Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, \mathfrak{g}, e) \leftarrow \mathcal{G}(\kappa)$ be an asymmetric pairing setting and we want to prove Diffie-Hellman tuples in $\mathbb{G}_2$, we set a tuple $(v_{1,1}, v_{1,2}, v_{2,1}, v_{2,2}) \in \mathbb{G}_1^4$, such that $(v_{1,1}, v_{1,2}, v_{2,1}, g \times v_{2,2})$ is not a Diffie-Hellman tuple.

Given a Diffie-Hellman tuple $(\mathfrak{g}, \mathfrak{g}', \mathfrak{A}, \mathfrak{A}')$ in $\mathbb{G}_2$, knowing the witness $\alpha \in \mathbb{Z}_p$ such that $\mathfrak{A} = \mathfrak{g}^\alpha$ and $\mathfrak{A}' = \mathfrak{g}'^\alpha$, one first commits α: $\mathsf{Com} = (c = v_{2,1}^\alpha v_{1,1}^\mu, d = v_{2,2}^\alpha g^\alpha v_{1,2}^\mu)$, for a random $\mu \xleftarrow{\$} \mathbb{Z}_p$, and one sets $\varTheta = \mathfrak{g}^\mu$ and $\varPsi = \mathfrak{A}^\mu$, which satisfy

$$\begin{aligned} e(c, \mathfrak{g}) &= e(v_{2,1}, \mathfrak{g}') \cdot e(v_{1,1}, \varTheta) & \qquad e(d, \mathfrak{g}) &= e(v_{2,2} \cdot g, \mathfrak{g}') \cdot e(v_{1,2}, \varTheta) \\ e(c, \mathfrak{A}) &= e(v_{2,1}, \mathfrak{A}') \cdot e(v_{1,1}, \varPsi) & \qquad e(d, \mathfrak{A}) &= e(v_{2,2} \cdot g, \mathfrak{A}') \cdot e(v_{1,2}, \varPsi) \end{aligned}$$

The proof $\mathsf{proof} = (\mathsf{Com}, \varTheta, \varPsi)$, when it satisfies the above relations, guarantees that $(\mathfrak{g}, \mathfrak{g}', \mathfrak{A}, \mathfrak{A}')$ is a Diffie-Hellman tuple. This proof is furthermore zero-knowledge, under the DDH assumption in $\mathbb{G}_1$: by switching $(v_{1,1}, v_{1,2}, v_{2,1}, g \times v_{2,2})$ into a Diffie-Hellman tuple, one can simulate the proof, as the commitment is perfectly hiding.

To verify the proof, instead of checking the four equations independently, one can apply a batch verification [BFI+10], and pack them in a unique one with random scalars $x_{1,1}, x_{1,2}, x_{2,1}$, $x_{2,2} \xleftarrow{\$} \mathbb{Z}_p$:

$$\begin{aligned} e(c^{x_{1,1}} d^{x_{1,2}}, \mathfrak{g}^{x_{2,1}} \mathfrak{A}^{x_{2,2}}) = e(&v_{2,1}^{x_{1,1}} (v_{2,2} \cdot g)^{x_{1,2}}, \mathfrak{g}'^{x_{2,1}} \mathfrak{A}'^{x_{2,2}}) \\ &\times e(v_{1,1}^{x_{1,1}} v_{1,2}^{x_{1,2}}, \varTheta^{x_{2,1}} \varPsi^{x_{2,2}}) \end{aligned}$$

One thus just has 3 pairing evaluations.

The interesting property of Groth-Sahai proofs is that it is possible from a Diffie-Hellman proof for $(\mathfrak{g}, \mathfrak{g}', \mathfrak{A}, \mathfrak{A}')$ to generate the Diffie-Hellman proof for $(\mathfrak{g}, \mathfrak{g}'' = \mathfrak{g}'^{\alpha'}, \mathfrak{A}, \mathfrak{A}'' = \mathfrak{A}'^{\alpha'})$ is a Diffie-Hellman tuple, just knowing the incremental witness α', whereas the new witness shoud be $\alpha\alpha'$, but is unknown to the prover: from the Diffie-Hellman proof $\mathsf{proof} = (\mathsf{Com}, \varTheta, \varPsi)$ for $(\mathfrak{g}, \mathfrak{g}', \mathfrak{A}, \mathfrak{A}')$ where

$$\mathsf{Com} = (c = v_{2,1}^\alpha v_{1,1}^\mu, d = v_{2,2}^\alpha v_{1,2}^\mu g^\alpha) \qquad \varTheta = \mathfrak{g}^\mu \qquad \varPsi = \mathfrak{A}^\mu$$

one can compute the proof proof' for $(\mathfrak{g}, \mathfrak{g}'' = \mathfrak{g}'^{\alpha'}, \mathfrak{A}, \mathfrak{A}'' = \mathfrak{A}'^{\alpha'})$, with $\mu' \xleftarrow{\$} \mathbb{Z}_p$ and:

$$\mathsf{Com}' = (c^{\alpha'} \cdot v_{1,1}^{\mu'}, d^{\alpha'} \cdot v_{1,2}^{\mu'}) \qquad \varTheta' = \varTheta^{\alpha'} \cdot \mathfrak{g}^{\mu'} \qquad \varPsi' = \varPsi^{\alpha'} \cdot \mathfrak{A}^{\mu'}$$

One implicitly updates α into $\alpha\alpha'$ and μ into $\alpha'\mu + \mu'$.

A.2 Multi-Signature

We now recall the signature of Boneh-Drijvers-Neven [BDN18] that provides a constant-size signature for an aggregation of multiple messages signed by multiple users. But the verification is also constant-time when the same message is signed by all the users (multi-signature). Since this is this latter case that is of interest for us, we focus in it. Let $\mathsf{MSparam} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, \mathfrak{g}, e) \leftarrow \mathcal{G}(\kappa)$ be an asymmetric pairing setting and let $\mathsf{H}_0 : \{0,1\}^* \to \mathbb{G}_2$ and $\mathsf{H}_1 : \{0,1\}^* \to \mathbb{Z}_p$ be two full-domain hash functions.

$\mathsf{MSKeygen}(\mathsf{MSparam})$**:** It chooses $\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_p$ and outputs $(\mathsf{sk}, \mathsf{vk} = \mathfrak{g}^{\mathsf{sk}})$;
$\mathsf{MSKeyAgg}(\{\mathsf{vk}_1, \ldots, \mathsf{vk}_N\})$**:** It outputs $\mathsf{avk} = \prod_{i=1}^{N} \mathsf{vk}_i^{\mathsf{H}_1(\mathsf{vk}_i, \{\mathsf{vk}_1, \ldots, \mathsf{vk}_N\})}$;
$\mathsf{MSSign}(\{\mathsf{vk}_1, \ldots, \mathsf{vk}_N\}, \mathsf{sk}_i, m)$**:** It outputs $\mathfrak{s}_i = \mathsf{H}_0(m)^{\mathsf{sk}_i}$.
From all the individual signatures $\mathfrak{s}_i$, any combiner (who can be one of the signers) computes the multi-signature $\mathsf{msig} = \prod_{j=1}^{N} \mathfrak{s}_j^{\mathsf{H}_1(\mathsf{vk}_i, \{\mathsf{vk}_1, \ldots, \mathsf{vk}_N\})}$;
$\mathsf{MSVerif}(\mathsf{avk}, m, \mathsf{msig})$**:** It outputs 1 if and only if $e(g, \mathsf{msig}) = e(\mathsf{H}_0(m), \mathsf{avk})$.

Since the aggregated verification key can be precomputed, verification just consists of two pairing evaluations.

B Complementary Material of our Mixnet

B.1 Verification Cost

For the reader convenience, we develop the equations to verify for all the signatures and the proofs:

$$\begin{array}{lllcl rr}
\sigma_{i,0} & 1 & & = & e(\sigma_{i,0}^{-1}, \mathfrak{g}_0)\ e(g, \mathfrak{g}_i)\ e(h, \mathfrak{h}_i)\ e(\ell, \mathfrak{l}_i) & & \\
\sigma'_{i,0} & 1 & & = & e({\sigma'}_{i,0}^{-1}, \mathfrak{g}'_0)\ e(g, \mathfrak{g}'_i)\ e(h, \mathfrak{h}'_i)\ e(\ell, \mathfrak{l}'_i) & & \\
\Sigma_{i,0} & e(\tau_{i,2}, \Sigma_{i,0}) & & = & e(g_3, \mathfrak{g}_0) & & \\
\Sigma'_{i,0} & e(\tau'_{i,2}, \Sigma'_{i,0}) & & = & e(g_3, \mathfrak{g}'_0) & & \\
\hline
\sigma_{i,1} & e(\sigma_{i,1}, \mathfrak{g}_0) & & = & e(g, \mathfrak{f}_i)\quad e(a_i, \mathfrak{g}_i)\ e(b_i, \mathfrak{h}_i)\ e(\ell_i, \mathfrak{l}_i) & \multicolumn{1}{|r}{3n} & +2 \\
\sigma'_{i,1} & & e({\sigma'}_{i,1}, \mathfrak{g}'_0) & = & e(g, \mathfrak{f}'_i)\quad e(a'_i, \mathfrak{g}'_i)\ e(b'_i, \mathfrak{h}'_i)\ e(\ell'_i, \mathfrak{l}'_i) & \multicolumn{1}{|r}{+3n} & +1 \\
\tau_i & e(\tau_{i,2}, \mathfrak{g}) & & = & e(g, \tau_{i,1}) & \multicolumn{1}{|r}{} & +0 \\
\tau'_i & e(\tau'_{i,2}, \mathfrak{g}) & & = & e(g, \tau'_{i,1}) & \multicolumn{1}{|r}{} & +0 \\
\Sigma_{i,1} & e(g_1, \mathfrak{g}_0) & e(\tau_{i,2}, \Sigma_{i,1}) & = & \qquad e(g_2, \mathfrak{f}_i)\ e(g_3, \mathfrak{l}_i)\ e(g_4, \mathfrak{g}_i)\ e(g_5, \mathfrak{h}_i) & \multicolumn{1}{|r}{+n} & +4 \\
\Sigma'_{i,1} & e(\tau'_{i,2}, \Sigma'_{i,1}) & e(g_1^{-1}, \mathfrak{g}'_0) & = & \qquad e(g_2, \mathfrak{f}'_i)\ e(g_3, \mathfrak{l}'_i)\ e(g_4, \mathfrak{g}'_i)\ e(g_5, \mathfrak{h}'_i) & \multicolumn{1}{|r}{+n} & \\
\mathsf{msig} & e(\mathsf{H}_0(\mathsf{proof}^{(N)}), \mathsf{avk}) & & = & e(g, \mathsf{msig}) & \multicolumn{1}{|r}{} & +1 \\
\multicolumn{5}{l}{\mathsf{proof}^{(N)} \text{ with } \mathfrak{F} = \prod_i \mathfrak{f}_i, \mathfrak{F}' = \prod_i \mathfrak{f}'_i,\ A = \prod_i a'_i / \prod a_i \text{ and } B = \prod_i b'_i / \prod b_i\text{:}} & \multicolumn{1}{|r}{} & \\
\multicolumn{5}{l}{e(c^{x_{1,1}} d^{x_{1,2}}, \mathfrak{g}^{x_{2,1}} \mathfrak{F}^{x_{2,2}}) = e(v_{2,1}^{x_{1,1}} (v_{2,2} \cdot g)^{x_{1,2}}, \mathfrak{g}'^{x_{2,1}} \mathfrak{F}'^{x_{2,2}}) e(v_{1,1}^{x_{1,1}} v_{1,2}^{x_{1,2}}, \Theta^{x_{2,1}} \Psi^{x_{2,2}})} & \multicolumn{1}{|r}{} & +3 \\
\multicolumn{5}{l}{e(g^{x'_{2,1}} A^{x'_{2,2}}, \mathfrak{c}^{x'_{1,1}} \mathfrak{d}^{x'_{1,2}}) = e(g'^{x'_{2,1}} B^{x'_{2,2}}, \mathfrak{v}_{2,1}^{x'_{1,1}} (\mathfrak{v}_{2,2} \cdot \mathfrak{g})^{x'_{1,2}}) e(\theta^{x'_{2,1}} \psi^{x'_{2,2}}, \mathfrak{v}_{1,1}^{x'_{1,1}} \mathfrak{v}_{1,2}^{x'_{1,2}})} & \multicolumn{1}{|r}{} & +3 \\
\hline
 & & & & & \multicolumn{1}{|r}{= 8n} & +14
\end{array}$$

One can remark that several pairings have common bases g, g_1, g_2, g_3, g_4, g_5 and $\mathfrak{g}_0 = \mathfrak{g}$, which can be combined together in order to decrease the number of pairings to be computed for the verification.

B.2 Correctness

We also show the correctness of our mix-net: if the input ballot box is correct and the mix-servers follow the protocol then the verifier outputs 1.

If the initial ballot-box $\mathcal{BB}\mathsf{ox}^{(0)} = (\overline{C}_i, \sigma_{i,1}, \mathsf{vk}_{i,1}, \Sigma_{i,1}, \tau_{i,1})_{i=1}^{n}$ is correct, then $\mathsf{Verif}(\mathsf{vk}_i, \overline{C}_i, \sigma_{i,1})$ $= 1$ and $\mathsf{Verif}^*(\mathsf{VK}, \tau_i, \mathsf{vk}_i, \Sigma_{i,1}) = 1$. The final ballot-box is $\mathcal{BB}\mathsf{ox}^{(N)} = (\overline{C}'_i, \sigma'_{i,1}, \mathsf{vk}'_{i,1}, \Sigma'_{i,1}, \tau'_{i,1})_{i=1}^{n'}$ and the proof of each mix-servers are $(\mathsf{proof}^{(k)}, \sigma^{(k)})_{k=1}^{N}$. If all the mix-servers follow the protocol then $n = n'$ and $\forall k, \mathsf{NIZK}_{\mathsf{DH}}\text{-}\mathsf{Verif}(\mathsf{proof}^{(k)}) = 1$ and $\mathsf{SVerif}(\mathsf{VK}_j, \mathsf{proof}^{(k)}, \sigma^{(k)}) = 1$. Let α_k be the witness of the proof $\mathsf{proof}^{(k)}$ and $\alpha = \prod_{k=1}^{N} \alpha_k$. One needs to verify if $\mathsf{Verif}(\mathsf{vk}'_i, \overline{C}'_i, \sigma'_{i,1}) = 1$ and $\mathsf{Verif}^*(\mathsf{VK}, \tau'_i, \mathsf{vk}'_i, \Sigma'_{i,1}) = 1$:

First one can remark that $\mathsf{Verif}(\mathsf{vk}'_i, \overline{C}'_0, \sigma'_{i,0}) = 1$ because

$$\begin{aligned}
e(\sigma'_{i,0}, \mathfrak{g}'_0) &= e(\sigma_{i,0} \cdot \ell^{\delta_i}, \mathfrak{g}_0)^\alpha = e(\sigma_{i,0}, \mathfrak{g}_0)^\alpha \cdot e(\ell^{\delta_i}, \mathfrak{g}_0)^\alpha \\
&= e(1, \mathfrak{f}_i)^\alpha e(\ell, \mathfrak{l}_i)^\alpha e(g, \mathfrak{g}_i)^\alpha e(h, \mathfrak{h}_i)^\alpha \cdot e(\ell^{\delta_i}, \mathfrak{g}_0)^\alpha \\
&= e(1, \mathfrak{f}'_i) e(g, \mathfrak{g}'_i) e(h, \mathfrak{h}'_i) \cdot e(\ell, \mathfrak{l}_i)^\alpha e(\ell, \mathfrak{g}_0^{\delta_i})^\alpha \\
&= e(1, \mathfrak{f}'_i) e(g, \mathfrak{g}'_i) e(h, \mathfrak{h}'_i) \cdot e(\ell, \mathfrak{l}_i^\alpha \cdot \mathfrak{g}_0^{\delta_i \alpha}) \\
&= e(1, \mathfrak{f}'_i) e(g, \mathfrak{g}'_i) e(h, \mathfrak{h}'_i) e(\ell, \mathfrak{l}'_i)
\end{aligned}$$

Now, one can check $\mathsf{Verif}(\mathsf{vk}'_i, \overline{C}'_i, \sigma'_{i,1}) = 1$ with the help of the previous computation:

$$\begin{aligned}
e(\sigma'_{i,1}, \mathfrak{g}'_0) &= e(\sigma'_{i,1}, \mathfrak{g}_0^\alpha) = e(\sigma_{i,1} \cdot \sigma_{i,0}^{\gamma_i} \cdot {\ell'_i}^{\delta_i}, \mathfrak{g}_0^\alpha) \\
&= e(\sigma_{i,1}, \mathfrak{g}_0)^\alpha \cdot e(\sigma_{i,0}^{\gamma_i}, \mathfrak{g}_0^\alpha) \cdot e({\ell'_i}^{\delta_i}, \mathfrak{g}_0^\alpha) \\
&= e(g, \mathfrak{f}_i)^\alpha e(\ell_i, \mathfrak{l}_i)^\alpha e(a_i, \mathfrak{g}_i)^\alpha e(b_i, \mathfrak{h}_i)^\alpha \cdot e(\sigma_{i,0}^{\gamma_i}, \mathfrak{g}_0^\alpha) \cdot e({\ell'_i}^{\delta_i}, \mathfrak{g}_0^\alpha) \\
&= e(g, \mathfrak{f}'_i) e(\ell_i, \mathfrak{l}_i^\alpha) e(a_i, \mathfrak{g}'_i) e(b_i, \mathfrak{h}'_i) \cdot e(\sigma_{i,0}^{\gamma_i}, \mathfrak{g}_0^\alpha) \cdot e({\ell'_i}^{\delta_i}, \mathfrak{g}_0^\alpha) \\
&= e(g, \mathfrak{f}'_i) e(a_i, \mathfrak{g}'_i) e(b_i, \mathfrak{h}'_i) \cdot e(\sigma_{i,0}^{\gamma_i}, \mathfrak{g}_0^\alpha) \cdot e(\ell_i, \mathfrak{l}_i^\alpha) e(\ell_i^{\delta_i} \cdot \ell^{\gamma_i \delta_i}, \mathfrak{g}_0^\alpha) \\
&= e(g, \mathfrak{f}'_i) e(a_i, \mathfrak{g}'_i) e(b_i, \mathfrak{h}'_i) \cdot e(\sigma_{i,0}^{\gamma_i}, \mathfrak{g}_0^\alpha) \cdot e(\ell_i, \mathfrak{l}_i^\alpha) e(\ell_i^{\delta_i}, \mathfrak{g}_0^\alpha) e(\ell^{\gamma_i \delta_i}, \mathfrak{g}_0^\alpha) \\
&= e(g, \mathfrak{f}'_i) e(a_i, \mathfrak{g}'_i) e(b_i, \mathfrak{h}'_i) \cdot e(\sigma_{i,0}^{\gamma_i}, \mathfrak{g}_0^\alpha) \cdot e(\ell_i, \mathfrak{l}_i^\alpha \cdot \mathfrak{g}_0^{\alpha \delta_i}) e(\ell_i \cdot \ell^{\gamma_i \delta_i}, \mathfrak{g}_0^\alpha) \\
&= e(g, \mathfrak{f}'_i) e(a_i, \mathfrak{g}'_i) e(b_i, \mathfrak{h}'_i) \cdot e(\sigma_{i,0}^{\gamma_i}, \mathfrak{g}_0^\alpha) \cdot e(\ell_i, \mathfrak{l}'_i) e(\ell^{\gamma_i \delta_i}, \mathfrak{g}_0^\alpha) \\
&= e(g, \mathfrak{f}'_i) e(\ell_i, \mathfrak{l}'_i) e(a_i, \mathfrak{g}'_i) e(b_i, \mathfrak{h}'_i) \cdot e(\sigma_{i,0}^{\gamma_i}, \mathfrak{g}_0^\alpha) \cdot e(\ell^{\gamma_i \delta_i}, \mathfrak{g}_0^\alpha) \\
&= e(g, \mathfrak{f}'_i) e(\ell_i, \mathfrak{l}'_i) e(a_i, \mathfrak{g}'_i) e(b_i, \mathfrak{h}'_i) \cdot e((\sigma_{i,0} \ell^{\delta_i})^{\gamma_i}, \mathfrak{g}_0^\alpha) \\
&= e(g, \mathfrak{f}'_i) e(\ell_i, \mathfrak{l}'_i) e(a_i, \mathfrak{g}'_i) e(b_i, \mathfrak{h}'_i) \cdot e(\sigma'_{i,0}, \mathfrak{g}_0^\alpha)^{\gamma_i} \\
&= e(g, \mathfrak{f}'_i) e(\ell_i, \mathfrak{l}'_i) e(a_i, \mathfrak{g}'_i) e(b_i, \mathfrak{h}'_i) \cdot e(1, \mathfrak{f}'_i)^{\gamma_i} e(\ell, \mathfrak{l}'_i)^{\gamma_i} e(g, \mathfrak{g}'_i)^{\gamma_i} e(h, \mathfrak{h}'_i)^{\gamma_i} \\
&= e(g, \mathfrak{f}'_i) e(\ell_i, \mathfrak{l}'_i) e(a_i, \mathfrak{g}'_i) e(b_i, \mathfrak{h}'_i) \cdot e(1^{\gamma_i}, \mathfrak{f}'_i) e(\ell^{\gamma_i}, \mathfrak{l}'_i) e(g^{\gamma_i}, \mathfrak{g}'_i) e(h^{\gamma_i}, \mathfrak{h}'_i) \\
&= e(g, \mathfrak{f}'_i) e(\ell_i \cdot \ell^{\gamma_i}, \mathfrak{l}'_i) e(a_i \cdot g^{\gamma_i}, \mathfrak{g}'_i) e(b_i \cdot h^{\gamma_i}, \mathfrak{h}'_i) \\
&= e(g, \mathfrak{f}'_i) e(\ell'_i, \mathfrak{l}'_i) e(a'_i, \mathfrak{g}'_i) e(b'_i, \mathfrak{h}'_i)
\end{aligned}$$

About the tags, one can see

$$e(g, \tau'_{i,1}) = e(g, \tau_{i,1}^{1/\mu_i}) = e(g, \tau_{i,1})^{1/\mu_i} = e(\tau_{i,2}, \mathfrak{g})^{1/\mu_i} = e(\tau_{i,2}^{1/\mu_i}, \mathfrak{g}) = e(\tau'_{i,2}, \mathfrak{g}).$$

For the certification, setting $\mathsf{VK} = (g_i)_i^6$ and $\tilde{\tau}_i = R_i$, one has $\tilde{\tau}'_i = R_i \mu_i$ and:

$$\begin{aligned}
e(\tau'_{i,2}, \varSigma'_{i,1}) &= e(g^{1/(R_i \mu_i)}, (\varSigma_{i,1} \cdot {\varSigma_{i,0}}^{\delta_i})^{\alpha \mu_i}) \\
&= e(g, \varSigma_{i,1}^{1/R_i})^\alpha \cdot e(g, \varSigma_{i,0}^{1/R_i})^{\delta_i \alpha} \\
&= e(g_1, \mathfrak{g}_0)^\alpha e(g_2, \mathfrak{f}_i)^\alpha e(g_3, \mathfrak{l}_i)^\alpha e(g_4, \mathfrak{g}_i)^\alpha e(g_5, \mathfrak{h}_i)^\alpha \cdot e(g, \varSigma_{i,0}^{1/R_i})^{\delta_i \alpha} \\
&= e(g_1, \mathfrak{g}'_0) e(g_2, \mathfrak{f}'_i) e(g_4, \mathfrak{g}'_i) e(g_5, \mathfrak{h}'_i) \cdot e(g_3, \mathfrak{l}_i^\alpha) e(g, \varSigma_{i,0}^{1/R_i})^{\delta_i \alpha} \\
&= e(g_1, \mathfrak{g}'_0) e(g_2, \mathfrak{f}'_i) e(g_4, \mathfrak{g}'_i) e(g_5, \mathfrak{h}'_i) \cdot e(g_3, \mathfrak{l}_i^\alpha) e(g_3, \mathfrak{g}_0^{\delta_i \alpha}) \\
&= e(g_1, \mathfrak{g}'_0) e(g_2, \mathfrak{f}'_i) e(g_4, \mathfrak{g}'_i) e(g_5, \mathfrak{h}'_i) \cdot e(g_3, \mathfrak{l}_i^\alpha \mathfrak{g}_0^{\delta_i \alpha}) \\
&= e(g_1, \mathfrak{g}'_0) e(g_2, \mathfrak{f}'_i) e(g_3, \mathfrak{l}'_i) e(g_4, \mathfrak{g}'_i) e(g_5, \mathfrak{h}'_i)
\end{aligned}$$

C LH-Sign From OT-LH-Sign with Square Diffie-Hellman Tuples

In this appendix, we propose a generic method to convert OT-LH-Sign into LH-Sign, using square Diffie-Hellman tuples. This requires the extractability assumption, in the generic bilinear group model.

C.1 Assumptions

Definition 17 (Square Discrete Logarithm (SDL) Assumption). In a group $\mathbb{G}$ of prime order p, it states that for any generator g, given $y = g^x$ and $z = g^{x^2}$, it is computationally hard to recover x.

Definition 18 (Decisional Square Diffie-Hellman (DSDH) Assumption). In a group $\mathbb{G}$ of prime order p, it states that for any generator g, the two following distributions are computationally indistinguishable:

$$\mathcal{D}_{\mathsf{sdh}}(g) = \{(g, g^x, g^{x^2}), x \xleftarrow{\$} \mathbb{Z}_p\} \qquad \mathcal{D}^3_{\$}(g) = \{(g, g^x, g^y), x, y \xleftarrow{\$} \mathbb{Z}_p\}.$$

It is worth noticing that the DSDH Assumption implies the SDL Assumption: if one can break SDL, from g, g^x, g^{x^2}, one can compute x and thus break DSDH.

C.2 Restricted Combinations of Vectors

When one wants to avoid any combination, and just allow to convert a signature of $\boldsymbol{M}$ into a signature of $\boldsymbol{M}^\alpha$, while they are all of the same format, one can use expanded vectors (as in Section 3.3), by concatenating a vector that satisfies this restriction: from multiple distinct (non-trivial) Square Diffie-Hellman tuples $(g_i, g_i^{w_i}, g_i^{w_i^2})$, a linear combination that is also a Square Diffie-Hellman tuple cannot use more than one input tuple. We prove it in two different cases: with random and independent bases g_i, but possibly public w_i's, or with a common basis $g_i = g$, but secret w_i's. More precisely, we can state the following theorems, which proofs can be found below.

We stress that in the first theorem, the w_i's are random and public (assumed distinct), but the bases g_i's are truly randomly and independently generated.

Theorem 19. *Given n valid Square Diffie-Hellman tuples $(g_i, a_i = g_i^{w_i}, b_i = a_i^{w_i})$, with w_i, for random $g_i \xleftarrow{\$} \mathbb{G}^*$ and $w_i \xleftarrow{\$} \mathbb{Z}_p^*$, outputting $(\alpha_i)_{i=1,\dots,n}$ such that $(G = \prod g_i^{\alpha_i}, A = \prod a_i^{\alpha_i}, B = \prod b_i^{\alpha_i})$ is a valid Square Diffie-Hellman, with at least two non-zero coefficients α_i, is computationally hard under the* DL *assumption.*

In the second scenario, the basis is common (for all i, $g_i = g$), but the w_i's are secret, still random and thus assumed distinct.

Theorem 20. *Given n valid Square Diffie-Hellman tuples $(g, a_i = g^{w_i}, b_i = a_i^{w_i})$ for any $g \in \mathbb{G}^*$ and random $w_i \xleftarrow{\$} \mathbb{Z}_p^*$, outputting $(\alpha_i)_{i=1,\dots,n}$ such that $(G = \prod g^{\alpha_i}, A = \prod a_i^{\alpha_i}, B = \prod b_i^{\alpha_i})$ is a valid Square Diffie-Hellman, with at least two non-zero coefficients α_i, is computationally hard under the* SDL *assumption.*

For the proofs below, we need to explicitly extract the linear combinations, hence the additional assumption that holds in the generic bilinear group model:

Definition 21 (Extractability Assumption). The extractability assumption states that given n vectors $(\boldsymbol{M}_j = (M_{j,i})_i)_j$, for any adversary that produces a new vector $\boldsymbol{M} = (M_i)_i$ such that $\boldsymbol{M} = \prod_j \boldsymbol{M}_j^{\alpha_j}$, there exists an extractor that outputs $(\alpha_j)_j$.

C.3 Proof of Theorem 19

Up to a guess, which is correct with probability greater than $1/n^2$, we can assume that $\alpha_1, \alpha_2 \neq 0$. We are given a discrete logarithm challenge Z, in basis g. We will embed it in either g_1 or g_2, by randomly choosing a bit b:

- if $b = 0$: set $X = Z$, and randomly choose $v \xleftarrow{\$} \mathbb{Z}_p$ and set $Y = g^v$

- if $b = 1$: set $Y = Z$, and randomly choose $u \xleftarrow{\$} \mathbb{Z}_p$ and set $X = g^u$

We set $g_1 \leftarrow X(= g^u)$, $g_2 \leftarrow Y(= g^v)$, with either u or v unknown, and randomly choose $\beta_i \in \mathbb{Z}_p$, for $i = 3, \ldots, n$ to set $g_i \leftarrow g^{\beta_i}$. Eventually, we randomly choose w_i, for $i = 1, \ldots, n$ and output $(g_i, a_i = g_i^{w_i}, b_i = a_i^{w_i})$ together with w_i, to the adversary which outputs $(\alpha_i)_{i=1,\ldots,n}$ such that $(G = \prod g_i^{\alpha_i}, A = \prod a_i^{\alpha_i} = G^w, B = \prod b_i^{\alpha_i} = A^w)$ for some unknown w. We thus have the following relations:

$$\left(\alpha_1 u + \alpha_2 v + \sum_{i=3}^{n} \alpha_i \beta_i\right) \cdot x = \alpha_1 u w_1 + \alpha_2 v w_2 + \sum_{i=3}^{n} \alpha_i \beta_i w_i$$
$$\left(\alpha_1 u w_1 + \alpha_2 v w_2 + \sum_{i=3}^{n} \alpha_i \beta_i w_i\right) \cdot x = \alpha_1 u w_1^2 + \alpha_2 v w_2^2 + \sum_{i=3}^{n} \alpha_i \beta_i w_i^2$$

If we denote $T = \sum_{i=3}^{n} \alpha_i \beta_i$, $U = \sum_{i=3}^{n} \alpha_i \beta_i w_i$, and $V = \sum_{i=3}^{n} \alpha_i \beta_i w_i^2$, that can be computed, we deduce that:

$$(\alpha_1 u w_1 + \alpha_2 v w_2 + U)^2 = (\alpha_1 u + \alpha_2 v + T)(\alpha_1 u w_1^2 + \alpha_2 v w_2^2 + V)$$

which leads to

$$\alpha_1 \alpha_2 (w_1^2 - w_2^2) uv + \alpha_1 (V - 2Uw_1 + Tw_1^2) u + \alpha_2 (V - 2Uw_2 + Tw_2^2) v + (TV - U^2) = 0$$

We consider two cases:

1. $K = \alpha_2 (w_1^2 - w_2^2) v + V - 2Uw_1 + Tw_1^2 = 0 \bmod p$;
2. $K = \alpha_2 (w_1^2 - w_2^2) v + V - 2Uw_1 + Tw_1^2 \neq 0 \bmod p$;

which can be determined by checking whether the equality below holds or not:

$$g^{-(V - 2Uw_1 + Tw_1^2)/(\alpha_2 (w_1^2 - w_2^2))} = Y.$$

One can note that case (1) and case (2) are independent of the bit b.

- If the case (1) happens, but $b = 0$, one aborts. If $b = 1$ (which holds with probability $1/2$ independently of the case) then we can compute $v = -(V - 2Uw_1 + Tw_1^2)/(\alpha_2 (w_1^2 - w_2^2)) \bmod p$ which is the discrete logarithm of Z in the basis g.
- Otherwise, the case (2) appears. If $b = 1$ one aborts. If $b = 0$ (which holds with probability $1/2$ independently of the case), v is known and we have $\alpha_1 K u + \alpha_2 (V - 2Uw_2 + Tw_2^2) v + (TV - U^2) = 0 \bmod p$, which means that the discrete logarithm of Z in the basis g is $u = -(\alpha_2 (V - 2Uw_2 + Tw_2^2) v + (TV - U^2))/(\alpha_1 K) \bmod p$. □

C.4 Proof of Theorem 20

Lemma 22. *Given any fixed value $\alpha \in \mathbb{Z}_p$ and n valid Square Diffie-Hellman tuples $(g, a_i = g^{w_i}, b_i = a_i^{w_i})$, for any $g \in \mathbb{G}$ and random $w_i \in \mathbb{Z}_p$, outputting $(\alpha_i)_{i=1,\ldots,n}$ such that $\alpha = \sum_{i=1}^{n} \alpha_i w_i$, with at least one non-zero coefficient α_i, is computationally hard under the* SDL *assumption.*

Proof. Up to a guess, which is correct with probability greater than $1/n$, we can assume that $\alpha_1 \neq 0$. We are given a square discrete logarithm challenge $(g, Z_1 = g^z, Z_2 = g^{z^2})$, in basis g. We set $a_1 \leftarrow Z_1$, $b_1 \leftarrow Z_2$, and randomly choose $w_i \xleftarrow{\$} \mathbb{Z}_p$, for $i = 2, \ldots, n$ to set $(a_i \leftarrow g^{w_i}, b_i \leftarrow a_i^{w_i})$. We then output (g, a_i, b_i), $i = 1, \ldots, n$, to the adversary which outputs $(\alpha_i)_{i=1,\ldots,n}$ and α such that $\alpha_1 z + \sum_{i=2}^{n} \alpha_i w_i = \alpha$. At this stage, we solve the square discrete logarithm problem by returning $z = (\alpha - \sum_{i=2}^{n} \alpha_i w_i)/\alpha_1 \bmod p$. □

We now come back to the proof of the theorem. Again, up to a guess, which is correct with probability greater than $1/n$, we can assume that $\alpha_1 \neq 0$. We are given a square discrete logarithm challenge $(g, Z_1 = g^z, Z_2 = g^{z^2})$, in basis g. We set $a_1 \leftarrow Z_1$, $a_2 \leftarrow Z_2$, and randomly choose $w_i \xleftarrow{\$} \mathbb{Z}_p$, for $i = 2, \ldots, n$ to set $(a_i \leftarrow g^{w_i}, b_i = a_i^{w_i})$. We then output (g, a_i, b_i), $i = 2, \ldots, n$, to the adversary that outputs $(\alpha_i)_{i=1,\ldots,n}$ such that $(G = \prod g^{\alpha_i}, A = \prod a_i^{\alpha_i} = G^w, B = \prod b_i^{\alpha_i} = A^w)$ for some unknown w. We thus have the following relations:

$$\left(\sum_{i=1}^{n} \alpha_i\right) \cdot w = \alpha_1 z + \sum_{i=2}^{n} \alpha_i w_i \qquad\qquad \left(\sum_{i=1}^{n} \alpha_i\right) \cdot w^2 = \alpha_1 z^2 + \sum_{i=2}^{n} \alpha_i w_i^2$$

which leads to

$$\left(\alpha_1 z + \sum_{i=2}^{n} \alpha_i w_i\right)^2 = \left(\alpha_1 + \sum_{i=2}^{n} \alpha_i\right) \times \left(\alpha_1 z^2 + \sum_{i=2}^{n} \alpha_i w_i^2\right).$$

If we denote $T = \sum_{i=2}^{n} \alpha_i w_i$, $U = \sum_{i=2}^{n} \alpha_i$, and $V = \sum_{i=2}^{n} \alpha_i w_i^2$, that can be computed from above scalars, we have $(\alpha_1 z + T)^2 = (\alpha_1 + U) \cdot (\alpha_1 z^2 + V)$, and thus

$$U\alpha_1 z^2 - 2T\alpha_1 z + (\alpha_1 + U)V - T^2 = 0 \bmod p.$$

Using Lemma 22 on the $n-1$ tuples (g, a_i, b_i), for $i = 2, \ldots, n$, the probability that $T = \sum_{i=2}^{n} \alpha_i w_i = 0$ is negligible, unless one can break the SDL Assumption. So we have $T \neq 0$, with two cases:

1. If $U \neq 0$ then, because computing square roots in $\mathbb{Z}_p$ is easy, one can solve the above quadratic equation for z that admits solutions, and obtain two solutions for z. By testing which one satisfies $g^z = Z_1$, one can find out the correct z and thus solve the SDL problem.
2. If $U = 0$, one can compute $z = (\alpha_1 V - T^2)/(2T\alpha_1) \bmod p$ and thus solve the SDL problem.

□

C.5 A First Generic Conversion from OT-LH-Sign to LH-Sign

Let $\Sigma = (\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Sign}, \mathsf{DerivSign}, \mathsf{Verif})$ be a $\mathsf{OT\text{-}LH\text{-}Sign}$, we complete it into $\Sigma' = (\mathsf{Setup}', \mathsf{Keygen}', \mathsf{NewTag}', \mathsf{VerifTag}', \mathsf{Sign}', \mathsf{DerivSign}', \mathsf{Verif}')$ as follows:

$\mathsf{Setup}'(1^\kappa)$**:** It runs $\mathsf{Setup}(1^\kappa)$ to obtain param and adds the tag space $\mathsf{param}' = (\mathsf{param}, \mathbb{Z}_p^* \times \mathbb{G}^*)$;

$\mathsf{Keygen}'(\mathsf{param}', n)$**:** It runs $\mathsf{Keygen}(\mathsf{param}, n+3)$;

$\mathsf{NewTag}'(\mathsf{sk})$**:** It chooses a random scalar $w \xleftarrow{\$} \mathbb{Z}_p^*$ and a random group element $h \xleftarrow{\$} \mathbb{G}$, and sets $\tilde{\tau} = \tau = (w, h)$;

$\mathsf{VerifTag}'(\mathsf{vk}, \tau)$**:** It checks whether $\tau = (w, h) \in \mathbb{Z}_p^* \times \mathbb{G}$ or not;

$\mathsf{Sign}'(\mathsf{sk}, \tilde{\tau} = (w, h), \boldsymbol{M})$**:** It extends $\boldsymbol{M}$ into $\boldsymbol{M}'$ with the three additional components (h, h^w, h^{w^2}), and signs it as $\sigma = \mathsf{Sign}(\mathsf{sk}, \boldsymbol{M}')$;

$\mathsf{DerivSign}'(\mathsf{vk}, \tau, \{\omega_i, \boldsymbol{M}_i, \sigma_i\}_{i=1}^{\ell})$**:** It simply computes $\sigma = \prod_i \sigma_i^{\omega_i}$ and $\tau' = (w, h' = \prod_i h^{\omega_i})$;

$\mathsf{Verif}'(\mathsf{vk}, \tau = (w, h), \boldsymbol{M}, \sigma)$**:** It first extends $\boldsymbol{M}$ into $\boldsymbol{M}'$ with the Square Diffie-Hellman tuple (h, h^w, h^{w^2}) and checks whether $\mathsf{Verif}(\mathsf{vk}, \boldsymbol{M}', \sigma) = 1$ or not.

One can note that the $\mathsf{DerivSign}'$ provides a signature under a new tag τ', but this is still consistent with the definition of the $\mathsf{LH\text{-}Sign}$. However, randomizability of the tag is not possible.

Theorem 23. *If Σ is $\mathsf{OT\text{-}LH\text{-}Sign}$ then Σ' is $\mathsf{LH\text{-}Sign}$ under the DL assumption.*

Proof. Since the tags are fully public, any NewTag'-query is answered by a random pair (w_i, g_i), and a Sign'-query is answered by simply forwarding a Sign-query to the Σ security game. Receiving the forgery $(\mathsf{vk}, \tau = (w, G), \boldsymbol{M}, \sigma)$, one first generates $\boldsymbol{M}'$ from $\boldsymbol{M}$ and τ and checks the validity, which means, according to the unforgeability of Σ, that there exist $(\alpha_i)_i$ such that $\boldsymbol{M}' = \prod \boldsymbol{M}_i'^{\alpha_i}$. The above extractability assumption provides these coefficients $(\alpha_i)_i$. If we just keep the 3 last components of each extended messages and the tags, we have square Diffie-Hellman triples $(g_i, a_i = g_i^{w_i}, b_i = a_i^{w_i})_i$, for random g_i and w_i (but possibly equal when the same tag is used several times), and the triple $(G = \prod g_i^{\alpha_i}, A = G^w = \prod a_i^{\alpha_i}, B = A^w = \prod b_i^{\alpha_i})$ extracted from the forgery. By combining the identical tags together, and so by summing in β_j the α_i's that correspond to the same triples (g_i, a_i, b_i), we have $(G = \prod g_j^{\beta_j}, A = G^w = \prod a_j^{\beta_j}, B = A^w = \prod b_j^{\beta_j})$, for random and distinct triples $(g_j, a_j, b_j)_j$. From Theorem 19, under the DL assumption, at most one coefficient is non-zero: none or β_J, and so at most one tag is represented: none or (g_J, a_J, b_J). Hence $\boldsymbol{M}$ is either $(1, \dots, 1)$ or $\prod \boldsymbol{M}_i^{\alpha_i}$ for i such that $(g_i, a_i, b_i) = (g_J, a_J, b_J)$. □

C.6 A Second Generic Conversion from OT-LH-Sign to LH-Sign

Let $\Sigma = (\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Sign}, \mathsf{DerivSign}, \mathsf{Verif})$ be a $\mathsf{OT\text{-}LH\text{-}Sign}$, we complete it into $\Sigma' = (\mathsf{Setup}', \mathsf{Keygen}', \mathsf{NewTag}', \mathsf{VerifTag}', \mathsf{Sign}', \mathsf{DerivSign}', \mathsf{Verif}')$ as follows:

$\mathsf{Setup}'(1^\kappa)$: It runs $\mathsf{Setup}(1^\kappa)$ to obtain param and adds the tag space $\mathsf{param}' = (\mathsf{param}, \mathbb{G}^3 \times \Pi)$. Note that we need the group $\mathbb{G}$ to be extended to a bilinear setting $(\mathbb{G}, \mathbb{G}_2, \mathbb{G}_T, p, g, \mathfrak{g}, e)$ for the proofs;

$\mathsf{Keygen}'(\mathsf{param}', n)$: It runs $\mathsf{Keygen}(\mathsf{param}, n+3)$;

$\mathsf{NewTag}'(\mathsf{sk})$: It chooses a random scalar $w \xleftarrow{\$} \mathbb{Z}_p^*$ and sets $\tilde{\tau} = w$ and $\tau = (g, g^w, g^{w^2}, \pi)$, where π is a zero-knowledge proof of valid square Diffie-Hellman tuple for (g, g^w, g^{w^2});

$\mathsf{VerifTag}'(\mathsf{vk}, \tau)$: It checks the proof π on (g, g^w, g^{w^2});

$\mathsf{Sign}'(\mathsf{sk}, \tilde{\tau} = w, \boldsymbol{M})$: It extends $\boldsymbol{M}$ into $\boldsymbol{M}'$ with the three additional components (g, g^w, g^{w^2}), and signs it as $\sigma = \mathsf{Sign}(\mathsf{sk}, \boldsymbol{M}')$;

$\mathsf{DerivSign}'(\mathsf{vk}, \tau = (\tau_1, \tau_2, \tau_3, \pi), \{\omega_i, \boldsymbol{M}_i, \sigma_i\}_{i=1}^{\ell})$: It computes $\sigma = \prod_i \sigma_i^{\omega_i}$, $\omega = \sum_i \omega_i$ and $\tau' = (\tau_1, \tau_2^\omega, \tau_3^\omega, \pi')$ with π' the updated proof of valid square Diffie-Hellman tuple;

$\mathsf{Verif}'(\mathsf{vk}, \tau = (\tau_1, \tau_2, \tau_3, \pi), \boldsymbol{M}, \sigma)$: It first checks whether $\mathsf{VerifTag}'(\mathsf{vk}, \tau) = 1$ or not, if the tag is valid, it extends $\boldsymbol{M}$ into $\boldsymbol{M}'$ with τ and checks whether $\mathsf{Verif}(\mathsf{vk}, \boldsymbol{M}', \sigma) = 1$ or not.

Note that for the $\mathsf{DerivSign}'$ to be possible, one needs an homomorphic zero-knowledge proof of valid square Diffie-Hellman tuple, as the Groth-Sahai techniques [GS08] allow in a bilinear setting: let $(v_{1,1}, v_{1,2}, v_{2,1}, v_{2,2}) \in \mathbb{G}_2^4$ be a Diffie-Hellman tuple, for a Square Diffie-Hellman tuple $(g, A = g^w, B = A^w) \in \mathbb{G}^3$ one can generate a commitment of w, $\mathsf{Com} = (c = v_{2,1}^w v_{1,1}^\mu, d = v_{2,2}^w v_{1,2}^\mu g^w) \in \mathbb{G}_2^2$, and the proofs $\mathsf{proof} = (\Theta = g^\mu, \Psi = A^\mu) \in \mathbb{G}^2$. The proof π thus consists of the pair $(\mathsf{Com}, \mathsf{proof})$, and is homomorphic. It is well-known to be perfectly-sound, and for the zero-knowledge property, one just has to switch from the Diffie-Hellman tuple $(v_{1,1}, v_{1,2}, v_{2,1}, v_{2,2})$ to a random tuple $(v_{1,1}, v_{1,2}, v_{2,1}, v_{2,2})$ because they are computationally indistinguishable under the DDH assumption in $\mathbb{G}_2$, or statistically indistinguishable in the generic group model. The latter assumption will be required for the security analysis below.

Theorem 24. *If Σ is $\mathsf{OT\text{-}LH\text{-}Sign}$ then Σ' is $\mathsf{LH\text{-}Sign}$, in the generic group model.*

Proof. Let us consider an adversary that asks several tags $(\tau_i)_i$ and signatures $(\sigma_i)_i$ on messages $(\boldsymbol{M}_i)_i$ and tags of its choice, and eventually produces a forgery $(\tau, \boldsymbol{M}, \sigma)$ with probability ε. A forgery means that

- the tag is valid, and so the proof π is accepted;
- the signature is valid;

- $\boldsymbol{M}$ is not in the spans of the messages signed under the same tag.

First, as the signature Σ' is based on the $\mathsf{OT\text{-}LH\text{-}Sign}$ Σ thanks to the concatenation of the message and (τ_1, τ_2, τ_3) in the tags, we know that necessarily $\boldsymbol{M'}$ (the completion of $\boldsymbol{M}$ with the triple in the tag) is a linear combination of the extended messages involved in the signing queries, unless one has broken the unforgeability of Σ, which can just happen with negligible probability.

As a consequence, the triple (τ_1, τ_2, τ_3) in the tag of the forgery is a linear combination of the Square Diffie-Hellman triples in the signing queries, with probability $\varepsilon' = \varepsilon - \mathsf{negl}()$:

- either (τ_1, τ_2, τ_3) is not a Square Diffie-Hellman tuple;
- or (τ_1, τ_2, τ_3) is a Square Diffie-Hellman tuple.

In the former case, where (τ_1, τ_2, τ_3) is not a Square Diffie-Hellman tuple, then we break the perfect soundness of Groth-Sahai proofs, as all the proofs for the honest tags have been generated honestly. Hence, the latter case should happen with probability greater than $\varepsilon" = \varepsilon' - \mathsf{negl}()$: (τ_1, τ_2, τ_3) is both a linear combination of the input triples but still a Square Diffie-Hellman tuple, with probability greater than ε''. Then, the Theorem 20 shows that (τ_1, τ_2, τ_3) is one of the input triples to a power α (or possibly $(1, 1, 1)$). However, to apply this theorem, we are given random Square Diffie-Hellman tuples as input and we should be able to generate the proofs of validity. To this aim, we switch the Groth-Sahai proofs in perfectly hiding mode: we replace a Non Diffie-Hellman tuple by a Diffie-Hellman tuple in the CRS, which is statistically indistinguishable to a generic adversary, as its probability to make the difference is N/p^2, where N is the number of group operations. So after this switch, from a list of Square Diffie-Hellman tuples, we simulate the proofs, and the adversary outputs a tuple (τ_1, τ_2, τ_3) that is both a linear combination of the input triples but still a Square Diffie-Hellman tuple, with probability greater than $\varepsilon'' - N/p^2$. As we are considering a forgery, several tags should be involved, which is excluded by the Theorem 20: ε'' is negligible, and so ε is negligible too.

C.7 Randomizable Tags

As in the Definition 11, we can randomize the tags together with the messages: but just in a computational way, and not in a statistical way. Indeed, from a message-signature $(\boldsymbol{M}, \sigma)$ for a tag $\tau = (\tau_1, \tau_2, \tau_3, \pi)$, one can derive the signature for the message $\boldsymbol{M'} = \boldsymbol{M}^\alpha$ for the tag $\tau' = (\tau_1^\alpha, \tau_2^\alpha, \tau_3^\alpha, \pi')$, where π' can be adapted from π and α. The triple $(\tau_1^\alpha, \tau_2^\alpha, \tau_3^\alpha)$ in the tag is not uniformy random, as w has not changed, but it is computationally unlinkable to (τ_1, τ_2, τ_3) under the DDH assumption. This is enough for our mix-net application.

C.8 Universal Tag

Whereas only messages signed under the same tag can be combined, a message signed under the tag $\tau_0 = (1, 1, 1, \pi)$, where $\pi = (1, 1, 1)$ is a proof for $w = 0$ with $\mu = 0$ in the commitment Com, can be combined with any message. Such a tag $(1, 1, 1, \pi)$, which was not in $\mathcal{T}$, is a *universal tag*. Indeed, multiplied to any Square Diffie-Hellman tuple, this is still a Square Diffie-Hellman tuple. This does not contradict the Theorems 19 and 20, as they only deal with non-trivial Square Diffie-Hellman triples. We can exploit this universal tag to optimize our construction of mix-net. Indeed, instead of having $\Sigma_{i,0}$ the $\mathsf{LH\text{-}Sign}$ of vk_0 for each user, it is possible to have $\Sigma_0 = \mathsf{Sign}^*(\mathsf{SK}, w, \mathsf{vk}_0)$ and still be able to randomize vk_i and adapt its signature $\Sigma_{i,1}$ keeping the tag τ_i per user.