A Certified-Input Mixnet from Two-Party Mercurial Signatures on Randomizable Ciphertexts

Masayuki Abe[1,2], Masaya Nanri[2], Miyako Ohkubo[3], Octavio Perez-Kempner[1], Daniel Slamanig[4], and Mehdi Tibouchi[1,2]

[1] NTT Social Informatics Laboratories, Tokyo, Japan
[2] Kyoto University, Kyoto, Japan
[3] Security Fundamentals Laboratory, CSRI, NICT
[4] Research Institute CODE, Universität der Bundeswehr München, Germany

Abstract. A certified-input mixnet introduced by Hébant *et al.* (PKC '20) employs homomorphically signed ciphertexts to reduce the complexity of shuffling arguments. However, the state-of-the-art construction relies on heavy Groth-Sahai proofs for key homomorphism, and only achieves honest-user security, limiting broader applicability.

This work proposes a novel certified-input mixnet achieving stronger security guarantees, alongside better efficiency. This is achieved by introducing a tailored signature scheme, *two-party mercurial signatures on randomizable ciphertexts*, that allows users and an authority to jointly sign ciphertexts supporting key, ciphertext, and signature randomization without compromising integrity and privacy.

We compare our approach to previous works that employ structured ciphertexts, implement our protocols, and provide performance benchmarks. Our results show that verifying the mixing process for 50,000 ciphertexts takes just 135 seconds on a commodity laptop using ten mixers, underscoring the practicality and efficiency of our approach.

Keywords: Mixnet, Certified Inputs, Mercurial Signatures, Voting.

1 Introduction

Mixnets [Cha81] use multiple servers to shuffle a set of encrypted messages in cascade to hide the relation between the initial input and resulting output. A central challenge has been how to efficiently guarantee the integrity of the messages. Hébant, Phan, and Pointcheval [HPP20] introduced a new paradigm, *certified-input mixnets*, that takes homomorphically signed ciphertexts as input and outputs randomized yet signed ciphertexts with a lightweight proof of correctness of the randomization of the keys, which is aggregated over all users' keys, together preserving the integrity of the embedded messages after shuffling. In contrast to prior approaches, which incur linear overhead in the number of mix servers on top of the ciphertexts (already linear in the number of users), their work

achieves constant overhead in the number of mix servers. Specifically, the verifier only needs to examine the input, the final output from the mixnet, and the proofs on the aggregated keys. These proofs can optionally be compressed into a single proof, eliminating the need for a linear number of proofs. Their approach requires a certification authority (CA) to certify inputs (as motivated by voting scenarios) and offers a highly scalable solution. This extended model also fits other practical applications such as oracle networks (*e.g.,* [MTV+23, CVM25]) including those deployed in platforms such as Chainlink [Cha25] that involve semi-trusted authorities and untrusted parties who run software as a service.

While promising, their state-of-the-art instantiation, referred to hereafter as HPP20, suffers from serious drawbacks, hindering real-world deployments. To clarify, we recap their construction at a high level. Each user holds a user-key uvk and a signature σ_a on it issued by the CA with its key avk. The user encrypts a message into a ciphertext ct and signs it with uvk into signature σ_u. The certified-input is thus structured as $(ct, \sigma_u, \sigma_a, \mathsf{uvk})$. Each mix-server randomize it into $(ct', \sigma'_u, \sigma'_a, \mathsf{uvk}')$ with shuffling, and provides a proof π of correct randomization of uvk in an aggregated form over all users' keys. The signatures and keys are to be homomorphic to allow randomization and aggregation. Unforgeability of σ_u guarantees that every signed output-ciphertext corresponds to a signed input-ciphertext. The proof π of correct randomization on aggregated keys ensures that every input user-key corresponds to an output user-key.

Firstly, we observe that their soundness and privacy are only guaranteed for *honest users*, and they actually break down in the presence of corrupt users. Suppose that there are corrupted users associated with keys $\mathsf{uvk}_1, \mathsf{uvk}_2$ and uvk_3 colluding with the first mix-server. They set up their keys to satisfy $\mathsf{uvk}_1 + \mathsf{uvk}_2 + \mathsf{uvk}_3 = 3 \cdot \mathsf{uvk}_1$ so that the first mix-server can replace their inputs with three randomized copies of the input from the first user without affecting to their aggregation. In the voting application, this is a serious threat for fairness as an adversary can replace some votes after other votes are cast.

Secondly, HPP20 involves several primitives: two linearly homomorphic signature schemes [LPJY13]—one for σ_u and the other for σ_a, a multi-signature scheme [BDN18] and Groth-Sahai proofs [GS08] included in π issued by each mix server. This results in a complex setup and use of an ad-hoc unlinkability assumption.

1.1 Our Contribution

We present a certified-input mixnet addressing previous drawbacks. Figure 1 illustrates the diagram of our *base* scheme whose details appear in Section 4. Following HPP20, to remove linearity in the number of mix servers, we discuss alternatives for mix-servers to jointly prove the correctness of their processing. Our key contributions include a new signature scheme, *Mercurial Signature on Randomizable Ciphertexts* (MSoRC), and its two-party signature generation protocol. We detail the advantages of our construction and contributions as follows.

Malicious-user Security. Our mixnet ensures soundness even in the presence of corrupt users. As shown in Figure 1, every ciphertext is signed using a joint key

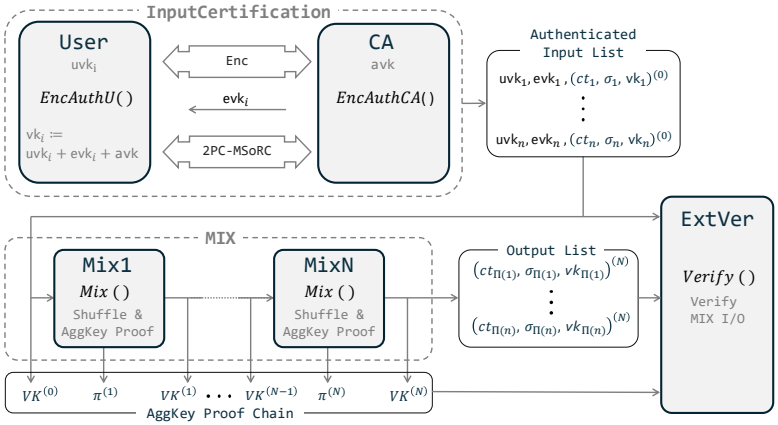

Fig. 1. Diagram of our *base* certified-input mixnet.

composed of the user's key (uvk_i), an ephemeral key (evk_i), and the authority's key (avk). The inclusion of evk prevents corrupt users from manipulating the distribution of the joint signature keys and ensures a one-to-one correspondence between input and output joint keys through a proof of correct key randomization on their aggregated form. We additionally allow the CA to randomize the ciphertexts, preventing the original user from opening them. While this does not directly enhance the standard security of mixnets, it is beneficial for achieving receipt-freeness in voting contexts, as discussed in Appendix A.

Simpler and More Efficient. Our approach replaces the two-layer structure of HPP20, where users sign ciphertexts and authorities sign user keys, with a two-party MSoRC signing protocol that produces a single signature verified by a joint key. This simplification eliminates the need for Groth-Sahai proofs executed by each mix-server. Instead, we simply require discrete logarithm proofs, further enhancing efficiency. Overall, our construction reduces setup complexity, computation, and verification costs compared to HPP20.

These desirable properties are attained with a moderate increase in complexity for the CA. Our two-party signing process necessitates two rounds of interaction, as opposed to the single round required in HPP20. Additionally, in HPP20, the CA's task can be completed independently of the ciphertext. While the CA's involvement is slightly more intensive in our approach, it's important to note that, in HPP20, users' keys are intended for one-time use and the CA must participate in each execution of the mixnet regardless.

New mercurial signature scheme. Our new primitive, MSoRC, is an independent contribution that we anticipate will have applications beyond mixnets. We first present a base MSoRC construction, which extends *signatures on randomizable ciphertexts* (SoRC) in [BF20] to allow key randomization, similar to *mercurial signatures* (MS) [CL19]. Then, we introduce its two-party signature generation, incorporating techniques from *interactive threshold mercurial signa-*

tures (TMS) in [ANKT25]. These constructions are secure against adversarially chosen encryption keys. However, for mixnets, a relaxed security notion with honestly generated encryption keys is sufficient. Consequently, we also present a more efficient variant that achieves an optimal signature size of three group elements [AGHO11].

Practical implementation. Our mixnet improves computation efficiency by a factor of 3.5× and communication by up to 3× compared to HPP20. It also significantly outperforms all previous works based on Rand-RCCA encryption discussed in Section 1.2. To evaluate practical performance, we provide a Rust implementation. To the best of our knowledge, this is the first practical implementation of mixnets under the certified input paradigm. In the worst-case scenario, for $n = 50\text{k}$ ciphertexts and $N = 10$ mixers, the mixing process takes approximately 40 seconds, and verifying the final mixing result takes around 135 seconds on a commodity laptop, without parallelization. All of our cryptographic building blocks can be easily implemented using existing libraries. Additionally, our modular design makes implementation tasks less error-prone.

1.2 Related Work

Signatures on Equivalence Classes and Randomizable Ciphertexts. Signatures on Equivalence Classes (EQS) [HS14, FHS19] are *malleable* structure-preserving signatures [AFG+10, AGHO11] (*i.e.,* pairing-based signatures with messages and public keys that are elements of a source group and whose verification is done using paring-product equations) defined over a message vector space. They allow a controlled form of malleability on message-signature pairs. EQS have further been studied to consider equivalence classes for the public key only [BHKS18] or both (latter introduced under the name of mercurial signatures in [CL19]). In addition, [BF20] considered a different equivalence relation for the message space and gave the first construction of SoRC [BFPV11] from EQS. In brief, it signs ElGamal ciphertexts and all randomizations of a ciphertext define an equivalence class. The motivation of SoRC is to build signatures on ciphertexts that could be adapted to randomizations of them. The SoRC construction from [BF20] (which is based on [FHS19]) provides a strong notion of *class-hiding* where an adapted message-signature pair looks like a completely random message-signature pair even when knowing the original message-signature pair. However, it only provides the same weak public-key class hiding guarantees of early constructions [CL19, CL21, CLPK22] (*i.e.,* original signers can identify adapted signatures for an adapted public key using their secret key). A stronger class-hiding notion for the public key was recently addressed in [ANKT25] where TMS are introduced. As it allows parties to produce a signature on their combined public keys, key-randomizability of the resulting signature provides a stronger class-hiding notion as long as parties keep their signing key private. We follow their two-party construction that is simpler and suffices for our purpose.

Verifiable Mixnets. Efficient proofs of shuffling have been continuously improved in the literature. The proof size and the verification work could be sublinear relative to the number of inputs, *e.g.,* [BG12]. In general, applying zk-SNARKs, *e.g.,* [GGPR13, Gro16, AHIV17, GWC19] to the shuffling relation the communication complexity can be reduced to poly-logarithmic. However, these approaches include several trade-offs, such as cumbersome setups, extensive common reference strings, heavy preprocessing requirements, or significant computational resource on the prover's side. Also, as noted in [HPP20], the proof grows linearly in the number of mix servers at the end. Such conditions are not necessarily acceptable for a scenario where casual users act as mix servers to protect their privacy autonomously.

While we focus on the certified input paradigm, other approaches ease the workload of verifiable shuffling by enforcing a specific structure on input ciphertexts to prevent malicious behaviour by mix-servers. Faonio *et al.* [FFHR19, FR22] use Re-randomizable Replayable CCA (Rand-RCCA) encryption [CKN03] to eliminate the need for a proof of shuffle, replacing it with NIZK proofs of plaintext knowledge for each ciphertext and NIZK proofs of membership at each mixing stage. Unfortunately, their approach requires a complex setup and incurs high computational costs. This is primarily because their Rand-RCCA scheme is based on Cramer-Shoup encryption [CS02], and the associated NIZK proofs involve elements in the target group of a pairing, significantly increasing proof size.

Another approach that diminishes a proof of shuffle appears in [CLW08]. It uses an escrowed linkable ring signature scheme and a regular signature to sign ciphertexts, publishing the former while hiding the latter among mix servers. However, as the ring includes all potential users, it does not easily scale.

Finally, there are works that explore post-quantum secure mixnets, such as [BHM20, ABG+21, HMS21, AKA+21, ABGS23a, ABGS23b]. Although these approaches require a careful selection of parameters and are far less efficient, exploring post-quantum security in the certified input paradigm remains a promising direction for future research.

1.3 Technical Overview

We provide a step-by-step overview of the two-party MSoRC construction and explain its role in our certified-input mixnet design.

1) SoRC. We begin by recalling SoRC from [BF20]. With verification key $(G, \hat{G}, \hat{X}_0, \hat{X}_1)$, its signature on ElGamal ciphertext (C_0, C_1) for encryption key (G, X) consists of group elements $(Z, S, \hat{S}, T)$. It is verified by three equations:

$$e(Z, \hat{S}) = e(C_0, \hat{X}_0)e(C_1, \hat{X}_1)e(G, \hat{G}),$$
$$e(S, \hat{G}) = e(G, \hat{S}), \text{ and } e(T, \hat{S}) = e(G, \hat{X}_0)e(X, \hat{X}_1)$$

where e is a bilinear map. The ciphertext and signature are malleable in the sense that anyone can re-randomize them; $(C_0', C_1') = (C_0 + rG, C_1 + rX)$ and

$(Z', S', \hat{S}', T') = ((Z+rT)/s, sS, s\hat{S}, T/s)$ for any r and $s \in \mathbb{Z}_p^*$. However, the key space is not malleable, as the verification equation involves $e(G, \hat{G})$, which rules out any key randomizations of the form $(\hat{X}_0^\rho, \hat{X}_1^\rho)$ for any $\rho \leftarrow\!\!\$\ \mathbb{Z}_p^*$ that would pass the first verification equation: $e(Z^\rho, \hat{S}) \neq e(C_0, \hat{X}_0^\rho)e(C_1, \hat{X}_1^\rho)e(G, \hat{G})$.

2) From SoRC to MSoRC. To obtain malleability on the key space, we turn the SoRC from [BF20] into a full-fledged MSoRC. Our MSoRC extends the signing key with one more element, $\hat{X}_2$, using it to sign a fixed generator G. The first verification equation changes to $e(Z, \hat{S}) = e(C_0, \hat{X}_0)e(C_1, \hat{X}_1)e(G, \hat{X}_2)$, and the key can be randomized within the equivalence class with factor ρ as $e(Z^\rho, \hat{S}) = e(C_0, \hat{X}_0^\rho)e(C_1, \hat{X}_1^\rho)e(G, \hat{X}_2^\rho)$. This prevents the ciphertext from being altered with $(C_0^{\rho'}, C_1^{\rho'})$ since G in $e(G, \hat{X}_2)$ must remain fixed. We prove security of our base MSoRC giving a reduction to the original SoRC.

3) Optimizing MSoRC. For honestly generated encryption key X, our MSoRC can be optimized to yield a signature, $(Z, \hat{S}, T)$, consisting of only three group elements without compromising the security. To see why, consider the case where x of $X = G^x$ is known to the adversary. Since the third verification equation is

$$e(T, \hat{S}) = e(G, \hat{X}_0)e(G^x, \hat{X}_1) = e(G, \hat{X}_0\hat{X}_1^x),$$

the adversary can compute T and $\hat{S}$ as $T = G$ and $\hat{S} = \hat{X}_0\hat{X}_1^x$ without knowing signing key x_0 and x_1. Thus, the second verification equation $e(S, \hat{G}) = e(G, \hat{S})$ has been involved to ensure that $\hat{S}$ has been computed with x_0 and x_1, even if decryption key x is known to the adversary. On the other hand, if x is not known to the adversary, the second equation is unnecessary and so does S. We prove this intuition rigorously in the Generic Group Model (GGM).

4) Two-party MSoRC. MSoRC provides the public key class-hiding only in a weak sense as the signer can trace randomized keys by using the signing key. Observe that the equivalence class of key $(\hat{X}_0, \hat{X}_1, \hat{X}_2)$ is defined by keys of the form $(\hat{X}_0^\rho, \hat{X}_1^\rho, \hat{X}_2^\rho)$. A key $(\hat{X}_0', \hat{X}_1', \hat{X}_2')$ is in the class if and only if $(\hat{X}_0')^{1/x_0} = (\hat{X}_1')^{1/x_1} = (\hat{X}_2')^{1/x_2} (= \hat{G}^\rho)$ holds for secret key (x_0, x_1, x_2). In [ANKT25], it is suggested to distribute the secret key among multiple parties, ensuring that no single party can perform tracing. However, achieving efficient distributed signing in the presence of malicious signers is a non-trivial challenge. Following the approach in [ANKT25], we additively distribute the secret key among two parties, a user and the CA, to fit to our certified-input mixnet scenario. The signing protocol for the two-party MSoRC follows a blind-compute-unblind structure, which allows us to simulate an honest party in the unforgeability proof when the other party is corrupted. We show that the unforgeability of the two-party MSoRC can be reduced to the unforgeability of the base MSoRC.

5) Mixnet from two-party MSoRC. We use the two-party MSoRC so that users and the CA jointly create a certified ciphertext as input to the mixnet. User i having ciphertext $(C_0, C_1)_i$ joins with the preliminary registered key, uvk_i, and the authority works with an ephemeral key evk_i and its long-term key avk. User key uvk_i as well as ephemeral key evk_i are published *in an authentic*

manner so that joint verification key $\mathsf{vk}_i := \mathsf{uvk}_i + \mathsf{evk}_i + \mathsf{avk}$ can be computed in public. The ephemeral key is included to ensure that every vk_i is independent. The randomized ciphertext $(C'_0, C'_1)_i$, MSoRC signature σ'_i, and verification key vk'_i (all publicly randomizable through adaptation functions of MSoRC), are the user's input to the mixnet. For simplicity, our communication model assumes the presence of a publicly verifiable authenticated channel – *i.e.,* all messages are recorded in a way that their authenticity can be publicly verified. This is equivalent to a bulletin board with authenticated writing, which is a standard assumption in e-voting applications, and the setting we adopt as well.

Considering $s_1, \ldots, s_N$ mix servers, s_j delivers $\mathcal{SS}\mathsf{et}^{(j)} := \{(C'_0, C'_1)_{\Pi(i)}, \sigma'_{\Pi(i)}, \mathsf{vk}'_{\Pi(i)}\}_{i\in[n]}$ for permutation $\Pi : [n] \to [n]$ and an authenticated NIZK proof of correct mixing to s_{j+1} using the statement from the previous round as the base point. The proof is: $\mathsf{NIZK}\{(\sum_{i=1}^{i=n} \mathsf{vk}'^{(j-1)}_{\Pi(i)}, \rho) : \sum_{i=1}^{i=n} \mathsf{vk}'^{(j)}_{\Pi(i)} = \rho \cdot \sum_{i=1}^{i=n} \mathsf{vk}'^{(j-1)}_{\Pi(i)}\}$.

This is where we replace Groth-Sahai with more lightweight Schnorr proofs thanks to our MSoRC structure. All servers authenticate their NIZK proof, which can be batch verified. Everyone can publicly verify the authenticated proofs to confirm the participation of each mix server while batch verification validates the output tuple. Only the initial tuples, the final ones, all the N short NIZK proofs, and server's public keys are needed for verification. This is because if the authenticated proofs verify, the output tuple implicitly validates the intermediate randomizations performed by each mix server. Alternatively, as in HPP20, the mix servers could perform a second round to produce a multi-signature on a single proof, making the final verification independent of N (cf. Section 4).

Security of MSoRC ensures that no collusion between mix servers and the CA can break public key unlinkability of honest users as long as one mix server is honest (*i.e.,* it correctly randomizes the tuples and permutes them). This holds even if the CA colludes with a subset of mix servers *and users*. Correctness of this process is ensured proving the correct randomization of verification keys, which is a discrete log proof on the sum of all of them.

2 Preliminaries

Notation. The set of integers from 1 to n is denoted as $[n]$. $\mathbb{Z}_p$ represents the ring of integers modulo p. For a set $\mathcal{S}$, $r \leftarrow\!\!\$\ \mathcal{S}$ denotes that r is sampled uniformly at random from $\mathcal{S}$. The security parameter κ is usually passed in unary form. Let $\mathcal{PP}$ be the set of public parameters. For each $\mathsf{pp} \in \mathcal{PP}$, let $\mathcal{M}_{\mathsf{pp}}$, $\mathcal{DK}_{\mathsf{pp}}$, $\mathcal{EK}_{\mathsf{pp}}$, $\mathcal{C}_{\mathsf{pp}}$, $\mathcal{R}_{\mathsf{pp}}$, $\mathcal{SK}_{\mathsf{pp}}$, $\mathcal{VK}_{\mathsf{pp}}$, and $\mathcal{S}_{\mathsf{pp}}$ denote the sets of messages, decryption keys, encryption keys, ciphertexts, ciphertext randomness, signature keys, verification keys, and signatures, respectively. Let BGGen be a PPT algorithm that on input 1^κ, returns public parameters $\mathsf{pp} \in \mathcal{PP}$ s.t. $\mathsf{pp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, G, \hat{G}, e)$, an asymmetric bilinear group where $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ are cyclic groups of prime order p with $\lceil \log_2 p \rceil = \kappa$, G and $\hat{G}$ are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an efficiently computable (non-degenerate) bilinear map. e is said to be of Type-3 if no efficiently computable isomorphisms between $\mathbb{G}_1$ and $\mathbb{G}_2$ are known.

ElGamal Encryption. ElGamal [ElG86] is an IND-CPA PKE scheme (KeyGen, Enc, Dec) in $\mathbb{G}_1$ under the the DDH assumption in $\mathbb{G}_1$. Key generation $\mathsf{KeyGen}(\mathsf{pp})$ chooses $\mathsf{dk} := x \leftarrow\!\!\$\ \mathbb{Z}_p^*$, sets $\mathsf{ek} := X \leftarrow xG$ and outputs $(\mathsf{dk}, \mathsf{ek})$. Encryption $\mathsf{Enc}(X, M)$ outputs ciphertext $(C_1, C_0) := (\mu G, M + \mu X)$ with $\mu \leftarrow\!\!\$\ \mathbb{Z}_p^*$. Decryption $\mathsf{Dec}(x, (C_0, C_1))$ outputs $M := C_1 - xC_0$.

Zero-Knowledge Proofs. Without loss of generality, we consider languages in NP defined in terms of a relation $\mathcal{L}_\mathcal{R} = \{\mathsf{x} |\ \exists\ \mathsf{w}$ s.t. $(\mathsf{x}, \mathsf{w}) \in \mathcal{R}_\mathcal{L}\}$, where $\mathsf{x} \in X$ and $\mathsf{w} \in W$ with $\mathcal{R}_\mathcal{L}$ being a subset of $X \times W$. With a zero-knowledge proof the prover proves to a verifier that $(\mathsf{x}, \mathsf{w}) \in \mathcal{R}_\mathcal{L}$ without disclosing any information about w. We will use Zero-Knowledge Proofs of Knowledge (ZKPoK) and Non-Interactive Zero-Knowledge Arguments (or simply NIZK). The former are three-round public coin, honest verifier zero-knowledge proofs that satisfy *knowledge soundness* (cf. [Gol01]). The latter are single-round protocols in the common reference string (crs) model whose syntax we recall next (cf. [DEF+25] and [CH20] for formal definitions). A NIZK proof system for a language $\mathcal{L}$ is defined by three algorithms: (1) Setup generates a crs and (optionally) a trapdoor; (2) Prove produces a proof for $(\mathsf{x}, \mathsf{w}) \in \mathcal{R}_\mathcal{L}$; (3) Verify verifies a proof w.r.t. an instance x. Couteau and Hartmann proposed a framework for building pairing-based NIZK for algebraic languages [CH20], an extension of linear languages. In particular, their framework is very well-suited as an alternative to GS proofs [GS08] due to its conceptual simplicity and because it provides fully adaptive soundness and perfect zero-knowledge with a single random group element as the crs. We will consider the following linear language $\mathcal{L}_\mathbf{A}$ for $\mathbf{A} = (\mathsf{A}_0, \mathsf{A}_1, \mathsf{A}_2) \in \mathbb{G}^3$ given by $\mathcal{R}_\mathsf{A} := \{(\mathbf{x}, w) : \mathbf{x} \in \mathbb{G}^3, w \in \mathbb{Z}_p\ s.t.\ \mathbf{x} = \mathbf{A}w\}$, which captures DDH relations. We show how to instantiate and batch verify a NIZK for $\mathcal{L}_\mathbf{A}$ in Appendix C. Moreover, it is also updatable, a feature that we discuss in Appendix C.2 and will use to optimize verification in our scheme.

3 Mercurial Signatures on Randomizable Ciphertexts

3.1 Definitions

Our definitions for MSoRC adapt the presentation from [BF20] to signatures on randomizable ciphertexts (similar to what [CL19] does for mercurial signatures when generalizing the ideas from [FHS19]). Thus, they can be seen as a merge between the original syntax and security properties of SoRC and MS schemes. For completeness, we include an algorithm $\mathsf{ConvertSK}$ in the syntax (it is not required by our mixnet but it could be useful in other applications of MSoRC). As in [CL19], let $\mathcal{R}$ be an equivalence relation where $[x]_\mathcal{R} = \{y \mid \mathcal{R}(x, y)\}$ denotes the equivalence class of which x is a representative. We loosely consider parametrized relations and say they are well-defined as long as the corresponding parameters are well-defined. We recall that signatures on randomizable ciphertexts are EQS where Adapt is analogous to ChgRep. More precisely, the equivalence class $[c]_\mathsf{ek}$ of a ciphertext c under encryption key ek is defined as all randomizations of c, that is, $[c]_\mathsf{ek} := \{c' \mid \exists\ r \in \mathcal{R}_\mathsf{pp} : c' = \mathsf{Rndmz}(\mathsf{ek}, c; r)\}$. Similarly, equivalence classes of

verification and secret keys are defined as $[\mathsf{vk}]_{\mathsf{vk}} := \{\mathsf{vk}' \mid \exists\ r \in \mathcal{R}_{\mathsf{pp}} : \mathsf{vk}' = r\mathsf{vk}\}$ and $[\mathsf{sk}]_{\mathsf{sk}} := \{\mathsf{sk}' \mid \exists\ r \in \mathcal{R}_{\mathsf{pp}} : \mathsf{sk}' = r\mathsf{sk}\}$, respectively.

Definition 1 (Mercurial Signature on Randomizable Ciphertexts). *A* MSoRC *scheme for parametrized equivalence relations $\mathcal{R}_c$, $\mathcal{R}_{\mathsf{pk}}$, $\mathcal{R}_{\mathsf{sk}}$ is a tuple of the following polynomial-time algorithms of which all except* Setup, KeyGen, *and* SKG *are implicitly parametrized by* pp *generated by* Setup*:*

$\mathsf{Setup}(1^\kappa) \to \mathsf{pp}$: *Outputs public parameters.*
$\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{ek}, \mathsf{dk})$: *Outputs encryption and decryption keys.*
$\mathsf{Enc}(\mathsf{ek}, m; r) \to c$: *Outputs a ciphertext c for a message m using randomness r.*
$\mathsf{Dec}(\mathsf{dk}, c) \to m$: *Outputs a message m.*
$\mathsf{Rndmz}(\mathsf{ek}, c; \mu) \to c'$: *Randomizes a ciphertext c into c' using random μ.*
$\mathsf{SKG}(\mathsf{pp}) \to (\mathsf{sk}, \mathsf{vk})$: *Outputs a signing key and a verification key.*
$\mathsf{Sign}(\mathsf{sk}, \mathsf{ek}, c; s) \to \sigma$: *Outputs a signature σ for c under* sk *using random s.*
$\mathsf{Verify}(\mathsf{vk}, \mathsf{ek}, c, \sigma) \to 0/1$: *Verifies (c, σ) w.r.t.* vk *and* ek.
$\mathsf{Adapt}(\sigma; \mu, \rho) \to \sigma'$: *Randomizes σ into σ' using random μ and ρ.*
$\mathsf{ConvertSK}(\mathsf{sk}, \rho) \to \mathsf{sk}'$: *Randomizes* sk *into* sk′ *using random ρ.*
$\mathsf{ConvertVK}(\mathsf{vk}, \rho) \to \mathsf{vk}'$: *Randomizes* vk *into* vk′ *using random ρ.*

Definition 2 (Correctness). *A* MSoRC *scheme is correct if for all sufficiently large κ, $\mathsf{pp} \in \mathsf{Setup}(1^\kappa)$, $(\mathsf{ek}, \mathsf{dk}) \in \mathsf{KeyGen}(\mathsf{pp})$, $(\mathsf{sk}, \mathsf{vk}) \in \mathsf{SKG}(\mathsf{pp})$, $m \in \mathcal{M}_{\mathsf{pp}}$, $r, \mu, \rho \in \mathcal{R}_{\mathsf{pp}}$, $\sigma \in \mathsf{Sign}(\mathsf{sk}, \mathsf{ek}, c)$ and $c \in \mathcal{C}_{\mathsf{pp}}$: $\mathsf{Dec}(\mathsf{dk}, \mathsf{Enc}(\mathsf{ek}, m; r)) = m$, $\Pr[\mathsf{Verify}(\mathsf{vk}, \mathsf{ek}, c, \sigma) = 1] = 1, \mathsf{ConvertSK}(\mathsf{sk}, \rho) \in [\mathsf{sk}]_{\mathsf{sk}} \wedge \mathsf{ConvertVK}(\mathsf{vk}, \rho) \in [\mathsf{vk}]_{\mathsf{vk}} \wedge \Pr[\mathsf{Verify}(\mathsf{ConvertVK}(\mathsf{vk}, \rho), \mathsf{ek}, \mathsf{Rndmz}(\mathsf{ek}, c; \mu), \mathsf{Adapt}(\sigma; \mu, \rho)) = 1] = 1$.*

Similar to mercurial signatures, MSoRC unforgeability should allow the adversary to output signatures under equivalent public keys (which are not considered a forgery). However, since MSoRC also deal with encryption keys, it is crucial to consider what happens to them and how they are managed in the unforgeability game. The unforgeability notion from BF20 [BF20] considers a forgery to be the case in which the adversary can produce a signature on an encryption of a message for an encryption key that has not been queried for that message. This strong unforgeability notion lets the adversary produce signatures under any encryption key pair of its choice. While such notion enables applications such as blind signatures [BFPV11], we observe that the encryption keys used in mixnets are either managed by the CA or by some other set of authorities (if a distributed key generation protocol is used to distribute trust) but not the users. Therefore, we can relax the unforgeability requirement so that it's the challenger who picks the encryption key pair instead of the adversary.[1] We formalize both variants as UNF-I and UNF-II next.

Definition 3 (UNF-I). *A* MSoRC *scheme is unforgeable if the advantage of any* PPT *adversary $\mathcal{A}$ defined by $\boldsymbol{Adv}^{\mathsf{UNF-I}}_{\mathsf{MSoRC}}(1^\kappa, \mathcal{A}) := Pr[\boldsymbol{Exp}^{\mathsf{UNF-I}}_{\mathsf{MSoRC}}(1^\kappa, \mathcal{A}) \Rightarrow \mathsf{true}] \leq \epsilon(\kappa)$, where $\boldsymbol{Exp}^{\mathsf{UNF-I}}_{\mathsf{MSoRC}}(1^\kappa, \mathcal{A})$ is shown in Fig. 2.*

Definition 4 (UNF-II). *A* MSoRC *scheme is unforgeable if the advantage of any* PPT *adversary $\mathcal{A}$ defined by $\boldsymbol{Adv}^{\mathsf{UNF-II}}_{\mathsf{MSoRC}}(1^\kappa, \mathcal{A}) := Pr[\boldsymbol{Exp}^{\mathsf{UNF-II}}_{\mathsf{MSoRC}}(1^\kappa, \mathcal{A}) \Rightarrow \mathsf{true}] \leq \epsilon(\kappa)$, where $\boldsymbol{Exp}^{\mathsf{UNF-II}}_{\mathsf{MSoRC}}(1^\kappa, \mathcal{A})$ is shown in Fig. 3.*

[1] This key observation allows us to obtain an even more efficient MSoRC construction.

$\underline{\text{Experiment } \mathbf{Exp}^{\mathsf{UNF\text{-}I}}_{\mathsf{MSoRC}}(1^\kappa, \mathcal{A})}$
$Q \leftarrow \emptyset; \mathsf{pp} \leftarrow\!\!\$\ \mathsf{Setup}(1^\kappa); (\mathsf{sk}, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{SKG}(\mathsf{pp}); (\mathsf{vk}^*, \mathsf{ek}^*, c^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk},\cdot,\cdot)}(\mathsf{vk})$
$\mathbf{return}\ (\mathsf{ek}^*, c^*) \notin Q\ \wedge [\mathsf{vk}^*]_{\mathsf{vk}} = [\mathsf{vk}]_{\mathsf{vk}} \wedge \mathsf{Verify}(\mathsf{vk}^*, \mathsf{ek}^*, c^*, \sigma^*)$
$\underline{\text{Oracle } \mathsf{Sign}(\mathsf{sk}, \mathsf{ek}, c)} : Q \leftarrow Q \cup \{\mathsf{ek}\} \times [c]_{\mathsf{ek}}; \mathbf{return}\ \mathsf{Sign}(\mathsf{sk}, \mathsf{ek})$

Fig. 2. Unforgeability experiment (UNF-I).

$\underline{\text{Experiment } \mathbf{Exp}^{\mathsf{UNF\text{-}II}}_{\mathsf{MSoRC}}(1^\kappa, \mathcal{A})}$
$Q \leftarrow \emptyset; \mathsf{pp} \leftarrow\!\!\$\ \mathsf{Setup}(1^\kappa); (\mathsf{sk}, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{SKG}(\mathsf{pp}); (\mathsf{dk}, \mathsf{ek}) \leftarrow\!\!\$\ \mathsf{KeyGen}(\mathsf{pp})$
$(\mathsf{vk}^*, c^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk},\cdot,\cdot)}(\mathsf{vk}, \mathsf{ek}); \mathbf{return}\ c^* \notin Q \wedge [\mathsf{vk}^*]_{\mathsf{vk}} = [\mathsf{vk}]_{\mathsf{vk}} \wedge \mathsf{Verify}(\mathsf{vk}^*, \mathsf{ek}, c^*, \sigma^*)$
$\underline{\text{Oracle } \mathsf{Sign}(\mathsf{sk}, \mathsf{ek}, c)} : Q \leftarrow Q \cup [c]_{\mathsf{ek}}; \mathbf{return}\ \mathsf{Sign}(\mathsf{sk}, \mathsf{ek})$

Fig. 3. Unforgeability experiment (UNF-II).

An MSoRC should also provide an encryption scheme with IND-CPA security and full class-hiding, as previously defined in [BF20].

Definition 5 (IND-CPA security & Full Class-Hiding [BF20]). *A* MSoRC *scheme is IND-CPA and full class-hiding if:*

IND-CPA: *the advantage of any* PPT *adversary* $\mathcal{A}$ *defined by* $\boldsymbol{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{MSoRC},\mathcal{A}}(\kappa) := 2 \cdot Pr\left[\boldsymbol{Exp}^{\mathsf{IND\text{-}CPA}}_{\mathsf{MSoRC},\mathcal{A}}(\kappa) \Rightarrow \mathsf{true}\right] - 1 = \epsilon(\kappa)$.
Full class-hiding: *the advantage of any* PPT *adversary* $\mathcal{A}$ *defined by* $\boldsymbol{Adv}^{\mathsf{Full\text{-}CH}}_{\mathsf{MSoRC},\mathcal{A}}(\kappa) := 2 \cdot Pr\left[\boldsymbol{Exp}^{\mathsf{Full\text{-}CH}}_{\mathsf{MSoRC},\mathcal{A}}(\kappa) \Rightarrow \mathsf{true}\right] - 1 = \epsilon(\kappa)$.
where $\boldsymbol{Exp}^{\mathsf{IND\text{-}CPA}}_{\mathsf{MSoRC},\mathcal{A}}(\kappa)$ *and* $\boldsymbol{Exp}^{\mathsf{Full\text{-}CH}}_{\mathsf{MSoRC},\mathcal{A}}(\kappa)$ *are defined as:*

Experiment $\boldsymbol{Exp}^{\mathsf{IND\text{-}CPA}}_{\mathsf{MSoRC},\mathcal{A}}(\kappa)$	*Experiment* $\boldsymbol{Exp}^{\mathsf{Full\text{-}CH}}_{\mathsf{MSoRC},\mathcal{A}}(\kappa)$
$\mathsf{pp} \leftarrow\!\!\$\ \mathsf{Setup}(1^\kappa); b \leftarrow\!\!\$\ \{0,1\}; r \leftarrow\!\!\$\ \mathcal{R}_{\mathsf{pp}}$	$\mathsf{pp} \leftarrow\!\!\$\ \mathsf{Setup}(1^\kappa); b \leftarrow\!\!\$\ \{0,1\}; r \leftarrow\!\!\$\ \mathcal{R}_{\mathsf{pp}}$
$(\mathsf{dk}, \mathsf{ek}) \leftarrow\!\!\$\ \mathsf{KeyGen}(\mathsf{pp})$	$(\mathsf{dk}, \mathsf{ek}) \leftarrow\!\!\$\ \mathsf{KeyGen}(\mathsf{pp})$
$(\mathsf{st}, m_0, m_1) \leftarrow \mathcal{A}(\mathsf{ek}); c \leftarrow \mathsf{Enc}(\mathsf{ek}, m_b, r)$	$(\mathsf{st}, c) \leftarrow \mathcal{A}(\mathsf{ek}); c_0 \leftarrow\!\!\$\ \mathcal{C}_{\mathsf{pp}}; c_1 \leftarrow \mathsf{Rndmz}(\mathsf{ek}, c; r)$
$b' \leftarrow\!\!\$\ \mathcal{A}(\mathsf{st}, c); \mathbf{return}\ b = b'$	$b' \leftarrow \mathcal{A}(\mathsf{st}, c_b); \mathbf{return}\ b = b'$

We consider signature adaptations for a new representative of the public key, extending the definition from [BF20].

Definition 6 (Adaption). *A* MSoRC *schemeis adaptable (under malicious keys) if for all sufficiently large* κ*, all* $\mathsf{pp} \in \mathsf{Setup}(1^\kappa)$*,* $(\mathsf{vk}, \mathsf{ek}, c, \sigma) \in \mathcal{VK}_{\mathsf{pp}} \times \mathcal{EK}_{\mathsf{pp}} \times \mathcal{C}_{\mathsf{pp}} \times \mathcal{S}_{\mathsf{pp}}$ *that satisfy* $\mathsf{Verify}(\mathsf{vk}, \mathsf{ek}, c, \sigma) = 1$ *and all* $(\mu, \rho) \in \mathcal{R}^2_{\mathsf{pp}}$*, the output of* $\mathsf{Adapt}(\sigma; \mu, \rho)$ *is uniformly distributed over the set* $\{\sigma' \in \mathcal{S}_{\mathsf{pp}} | \mathsf{Verify}(\mathsf{ConvertVK}(\mathsf{vk}, \rho), \mathsf{ek}, \mathsf{Rndmz}(\mathsf{ek}, c, \mu), \sigma') = 1\}$.

We will also consider an interactive signing protocol as defined below.

$\mathsf{ISign}_{\mathsf{P}_0}(\mathsf{sk}_0, \mathsf{ek}, c) \leftrightarrow \mathsf{ISign}_{\mathsf{P}_1}(\mathsf{sk}_1, \mathsf{ek}, c) \rightarrow \sigma$: This algorithm is run interactively. It produces a signature σ for c under sk, implicitly defined as $\mathsf{sk}_0 + \mathsf{sk}_1$.

Consequently, we define unforgeability and public-key class-hiding assuming at least one honest signer. To prove security, we introduce a key generation

$\underline{\textit{Experiment}\ \mathbf{Exp}^{\mathsf{UNF-III}}_{\mathsf{MSoRC}}(1^\kappa, \mathcal{A})}$
$Q \leftarrow \emptyset$; $\mathsf{pp} \leftarrow\!\!\$\ \mathsf{Setup}(1^\kappa)$; $(b, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp})$; $(\mathsf{dk}, \mathsf{ek}) \leftarrow\!\!\$\ \mathsf{KeyGen}(\mathsf{pp})$
$(\mathsf{sk}_i, \mathsf{vk}_i)_{i\in\{0,1\}} \leftarrow\!\!\$\ \mathsf{TKGen}(\mathsf{pp})$; $\mathsf{vk} \leftarrow \mathsf{vk}_0 + \mathsf{vk}_1$
$(\mathsf{vk}^*, c^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{ISign}_{1-b}(\mathsf{sk}_{1-b},\cdot,\cdot)}(\mathsf{st}, \mathsf{vk}_0, \mathsf{vk}_1, \mathsf{sk}_b, \mathsf{ek})$
return $c^* \notin Q \wedge [\mathsf{vk}^*]_{\mathsf{vk}} = [\mathsf{vk}]_{\mathsf{vk}} \wedge \mathsf{Verify}(\mathsf{vk}^*, \mathsf{ek}, c^*, \sigma^*) = 1$
$\underline{\textit{Oracle}\ \mathsf{ISign}_{1-b}(\mathsf{sk}_{1-b}, \mathsf{ek}, c)}$: $Q \leftarrow Q \cup [c]_{\mathsf{ek}}$; **return** $\mathsf{ISign}_{1-b}(\mathsf{sk}_{1-b}, \mathsf{ek}, c)$

Fig. 4. Unforgeability w.r.t an interactive signing protocol.

algorithm that is run by a trusted third party that produces $(\mathsf{vk}, \mathsf{sk})$ as in SKG but such that $\mathsf{vk} = \mathsf{vk}_0 + \mathsf{vk}_1$ and $\mathsf{sk} = \mathsf{sk}_0 + \mathsf{sk}_1$ (in practice, each party will run SKG independently). We require the following property adapted from [ANKT25].

Definition 7 (Security of key generation). TKGen *is secure if it outputs* vk *with the same distribution as* SKG*, and there exists a simulator,* $\mathsf{SimTKGen}$*, s.t. for any sufficiently large* κ*, any* $\mathsf{pp} \in \mathsf{Setup}(1^\kappa)$*,* $(\mathsf{vk}, \mathsf{sk}) \in \mathsf{SKG}(\mathsf{pp})$*, and* $b \in \{0, 1\}$*,* $\mathsf{SimTKGen}(\mathsf{vk}, b)$ *outputs* sk_b *and* $\{\mathsf{vk}_0, \mathsf{vk}_1\}$*. The joint distribution of* $(\mathsf{vk}, \mathsf{vk}_0, \mathsf{vk}_1, \mathsf{sk}_b)$ *is indistinguishable from that of* $\mathsf{TKGen}(\mathsf{pp})$*.*

For unforgeability, we let the adversary choose one of the parties and leak its corresponding keys. The encryption key pair is generated by the challenger, which suffices for our application. However, we emphasize that the definition below can easily be generalized to adversarially chosen keys, as earlier discussed.

Definition 8 (UNF-III). *A* MSoRC *scheme is unforgeable if the advantage of any* PPT *adversary* $\mathcal{A}$ *having access to an interactive signing oracle defined by* $\boldsymbol{Adv}^{\mathsf{UNF-III}}_{\mathsf{MSoRC}}(1^\kappa, \mathcal{A}) := Pr[\boldsymbol{Exp}^{\mathsf{UNF-III}}_{\mathsf{MSoRC}}(1^\kappa, \mathcal{A}) \Rightarrow \mathsf{true}] \leq \epsilon(\kappa)$*, where* $\boldsymbol{Exp}^{\mathsf{UNF-III}}_{\mathsf{MSoRC}}(1^\kappa, \mathcal{A})$ *is shown in Fig. 4.*

For public key class-hiding, we adapt definitions from [CL19] and [ANKT25] (*i.e.,* considering an interactive signing protocol). This allows us to obtain a stronger notion of public key class-hiding when one of the parties is honest. In other words, full public key class hiding holds if the parties don't collude. Following the naming convention from [ANKT25], we formalize this notion as *public key unlinkability.* As we shall see, this notion suffices for the considered applications.

Definition 9 (PK-UNL). *A* MSoRC *scheme is public key unlinkable if the advantage of any* PPT *adversary* $\mathcal{A}$ *defined by* $\boldsymbol{Adv}^{\mathsf{PK-UNL}}_{\mathsf{MSoRC}}(1^\kappa, \mathcal{A}) := 2 \cdot Pr\left[\boldsymbol{Exp}^{\mathsf{PK-UNL}}_{\mathsf{MSoRC}}(1^\kappa, \mathcal{A}) \Rightarrow \mathsf{true}\right] - 1 \leq \epsilon(\kappa)$*, where* $\boldsymbol{Exp}^{\mathsf{PK-UNL}}_{\mathsf{MSoRC}}(1^\kappa, \mathcal{A})$ *is shown in Fig. 5.*

3.2 Single-Signer Construction

In Fig. 6, we present the base MSoRC with a single signer. Our departure point is the SoRC from [BF20], which is an EQS based on [FHS19] that signs ElGamal ciphertexts. In [BF20], a signature consists of four group elements $Z = \frac{1}{s}(x_0 C_0 +$

$\underline{\text{Experiment } \mathbf{Exp}^{\mathsf{PK\text{-}UNL}}_{\mathsf{MSoRC}}(1^\kappa, \mathcal{A})}$
$\mathsf{pp} \leftarrow\!\$\ \mathsf{Setup}(1^\kappa); \rho \leftarrow\!\$\ \mathcal{R}_{\mathsf{pp}}; b \leftarrow\!\$\ \{0,1\}; (\widetilde{\mathsf{sk}}, \widetilde{\mathsf{vk}}) \leftarrow\!\$\ \mathsf{TKGen}(\mathsf{pp})$
$(\mathsf{sk}_i, \mathsf{vk}_i)_{i\in\{0,1\}} \leftarrow\!\$\ \mathsf{TKGen}(\mathsf{pp}); \mathsf{vk}' \leftarrow \mathsf{ConvertVK}(\widetilde{\mathsf{vk}} + \mathsf{vk}_b, \rho)$
$b' \leftarrow\!\$\ \mathcal{A}^{\mathsf{ISign}(\mathsf{sk}_b,\cdot,\cdot,\cdot)}(\widetilde{\mathsf{sk}}, \widetilde{\mathsf{vk}}, \mathsf{vk}', \mathsf{vk}_0, \mathsf{vk}_1); \mathbf{return}\ b = b'$
$\underline{\text{Oracle } \mathsf{ISign}(\mathsf{sk}_b, \mathsf{ek}, c, \mathsf{vk})}$
$\mathbf{if}\ \mathsf{vk} = \mathsf{vk}'\ \mathbf{then}\ \sigma \leftarrow\!\$\ \mathsf{ISign}_b(\mathsf{sk}_b, \mathsf{ek}, c)$
$\mathbf{return}\ \mathsf{Adapt}(\sigma; \rho)\ \mathbf{elseif}\ \mathsf{vk} = \mathsf{vk}_i\ \mathbf{return}\ \mathsf{ISign}(\mathsf{sk}_i, \mathsf{ek}, c)$

Fig. 5. Public key unlinkability experiment (PK-UNL).

$\underline{\mathsf{MSoRC.Setup}(1^\kappa) :}\ \mathsf{pp} := (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, G, \hat{G}, e) \leftarrow\!\$\ \mathsf{BGGen}(1^\kappa); \mathbf{return}\ (\mathsf{pp})$
$\underline{\mathsf{MSoRC.KeyGen}(\mathsf{pp}) :}\ \mathsf{dk} := x \leftarrow\!\$\ \mathbb{Z}_p^*; \mathsf{ek} := X \leftarrow xG; \mathbf{return}\ (\mathsf{dk}, \mathsf{ek})$
$\underline{\mathsf{MSoRC.SKG}(\mathsf{pp}) :}\ \mathsf{sk} := (x_0, x_1, x_2) \leftarrow \mathbb{Z}_p^*; \mathsf{vk} := (x_0\hat{G}, x_1\hat{G}, x_2\hat{G}); \mathbf{return}\ (\mathsf{sk}, \mathsf{vk})$
$\underline{\mathsf{MSoRC.Enc}(X, M; r) :}\ \mathbf{return}\ (rG, M + rX)$
$\underline{\mathsf{MSoRC.Dec}(x, (C_0, C_1)) :}\ \mathbf{return}\ M := C_1 - xC_0$
$\underline{\mathsf{MSoRC.Rndmz}(X, (C_0, C_1); \mu) :}\ \mathbf{return}\ (C_0 + \mu G, C_1 + \mu X)$
$\underline{\mathsf{MSoRC.Sign}((x_0, x_1, x_2), X, (C_0, C_1)) :}$
$s \leftarrow\!\$\ \mathbb{Z}_p^*; Z := \frac{1}{s}(x_0C_0 + x_1C_1 + x_2G); S := sG; \hat{S} := s\hat{G}; T := \frac{1}{s}(x_0G + x_1X)$
$\mathbf{return}\ (Z, S, \hat{S}, T)$
$\underline{\mathsf{MSoRC.ConvertSK}((x_0, x_1, x_2), \rho) :}\ \mathbf{return}\ (\rho x_0, \rho x_1, \rho x_2)$
$\underline{\mathsf{MSoRC.ConvertVK}((\hat{X}_0, \hat{X}_1, \hat{X}_2), \rho) :}\ \mathbf{return}\ (\rho\hat{X}_0, \rho\hat{X}_1, \rho\hat{X}_2)$
$\underline{\mathsf{MSoRC.Adapt}((Z, S, \hat{S}, T); \mu, \rho) :}$
$s' \leftarrow\!\$\ \mathbb{Z}_p^*; Z' := \frac{\rho}{s'}(Z + \mu T); S' := s'S; \hat{S}' := s'\hat{S}; T' := \frac{\rho}{s'}T; \mathbf{return}\ (Z', S', \hat{S}', T')$
$\underline{\mathsf{MSoRC.Verify}((\hat{X}_0, \hat{X}_1, \hat{X}_2), X, (C_0, C_1), (Z, S, \hat{S}, T)) :}$
$\mathbf{return}\ e(Z, \hat{S}) = e(C_0, \hat{X}_0)e(C_1, \hat{X}_1)e(G, \hat{X}_2)$
$\wedge\, e(T, \hat{S}) = e(G, \hat{X}_0)e(X, \hat{X}_1) \wedge e(S, \hat{G}) = e(G, \hat{S})$

Fig. 6. Our base MSoRC scheme.

$x_1C_1 + G), S = sG, \hat{S} = s\hat{G}$ and $T = \frac{1}{s}(x_0G + x_1X)$, where (C_0, C_1) is the ciphertext, X it's encryption key, and (x_0, x_1) the scheme's signing key. Without G, $(Z, S, \hat{S})$ is the EQS from [FHS19]. The idea from [BF20] was to embed G into Z so that Z can only be adapted to ciphertext randomizations using the additional element T. To turn the SoRC from [BF20] into a full-fledged MSoRC we extend the secret key to include one more element x_2 and use it to sign G in Z. This way, Z can be adapted to a new key representative, as well as to a ciphertext randomization if T is used.

Correctness of our base scheme follows by inspection. ElGamal is IND-CPA if the DDH assumption holds, which we assume. Full class-hiding was already proven in [BF20] giving a reduction to DDH. Likewise, signature adaption follows directly from that of the original SoRC ([BF20], Proposition 2). Next, we

prove unforgeability and public key unlinkability. As in related work ([ANKT25, BF20]), we consider the stand-alone model and adversaries in the GGM.

Theorem 1. *Our base* MSoRC *is unforgeable in the GGM with respect to Definition 3, and public key unlinkable under corruption of at most one party.*

Proof. **Unforgeability.** Security of our base scheme (Def. 3) is reduced to that of [BF20]. We consider a reduction $\mathcal{B}$ playing the role of the adversary against [BF20]. $\mathcal{B}$ receives $\mathsf{pk} = (\hat{X}_0, \hat{X}_1)$ from the challenger, it picks $\alpha \leftarrow\!\!\$\ Z_p^*$, sets $\mathsf{pk}' := (\alpha\hat{X}_0, \alpha\hat{X}_1, \alpha\hat{G})$ for our scheme and forwards it to $\mathcal{A}$. Whenever $\mathcal{A}$ asks for a signature on $(C_0^{(i)}, C_1^{(i)}, X^{(i)})$, $\mathcal{B}$ forwards to the signing oracle of [BF20]. On receiving $\sigma^{(i)} = (Z^{(i)}, T^{(i)}, S^{(i)}, \hat{S}^{(i)})$, it sets $\sigma^{(i)'} = (\alpha Z^{(i)}, \alpha T^{(i)}, S^{(i)}, \hat{S}^{(i)})$ and returns it to $\mathcal{A}$. Whenever $\mathcal{A}$ outputs $(Z^*, T^*, S^*, \hat{S}^*)$ and (C_0^*, C_1^*, X^*) for public key $\mathsf{pk}^* = \beta\mathsf{pk}'$, $\mathcal{B}$ outputs $(\frac{1}{\alpha\beta}Z^*, \frac{1}{\alpha\beta}T^*, S^*, \hat{S}^*)$ for the same query. We note that $\mathcal{B}$ is a generic forger and thus, it can obtain β. To see how, we proceed as done in [CL19] (Claim 1). Since $\mathcal{A}$ is a generic forger, the forged key must be computed as a linear combination of previously seen elements. Thus, for all $i \in \{0, 1, 2\}$:

$$\hat{X}_i^* = \chi^1\hat{G} + \chi_0^1\hat{X}_0 + \chi_1^1\hat{X}_1 + \chi_2^0\hat{X}_2 + \sum_{j=1}^{k}\chi_{s,j}^1\hat{S}_j$$

Taking the discrete logarithm base $\hat{G}$, we get:

$$x_i^* = \chi^1 + \chi_0^1 x_0 + \chi_1^1 x_1 + \chi_2^0 x_2 + \sum_{j=1}^{k}\chi_{s,j}^1 s_j$$

The above is a multivariate polynomial of degree $O(k)$ in $x_0, x_1, x_2, s_1, \ldots, s_k$. Consider the probability that two formally different polynomials collide such that $x_i^* = \beta x_i$, but $\mathcal{B}$ cannot obtain $\beta \in \mathbb{Z}_p^*$ despite seeing $\mathcal{A}$'s queries to the group and signing oracles and their results. By the Schwartz-Zippel lemma, such probability is $O(\frac{k}{p})$, which is negligible.

Public key unlinkability. We show that adapted signatures are independent of b, *i.e.,* the adversary gains no information by knowing one of the shares of the corresponding secret key. For any tuple $(X, (C_0, C_1))$, an adapted signature from one computed using $\widetilde{\mathsf{sk}}$, sk_b and a uniformly random ρ verifies under $\mathsf{vk}' = \rho(\widetilde{\mathsf{sk}} + \mathsf{sk}_b)$ and has the following distribution for uniformly random values s and δ: $Z = \frac{1}{s}(\rho(\widetilde{\mathsf{sk}}^0 + \mathsf{sk}_b^0)C_0 + \rho(\widetilde{\mathsf{sk}}^1 + \mathsf{sk}_b^1)C_1 + \rho(\widetilde{\mathsf{sk}}^1 + \mathsf{sk}_b^1)G)$, $T = \frac{1}{s}(\rho(\widetilde{\mathsf{sk}}^0 + \mathsf{sk}_b^0)G + \rho(\widetilde{\mathsf{sk}}^1 + \mathsf{sk}_b^1)X)$, $S = sG$, and $\hat{S} = s\hat{G}$. Since ρ is uniformly random, it perfectly hides b and the adversary gains no information dependent on b. □

3.3 Two-Party Construction

We can extend our construction to support a two-party *interactive* signing protocol as shown in Fig. 7. We do so using the techniques from [ANKT25] to build TMS, and all elements are computed analogously (*e.g.,* we compute a blinded

version of Z and T, with each party proving the correctness of each step via short ZKPoK's). ZKPoK's are defined as follows:

- $\mathsf{ZKPoK}[s_0 : S_0 = s_0 G \wedge \hat{S}_0 = s_0 \hat{G}]$,
- $\mathsf{ZKPoK}[(s_0, x_0^0, x_1^0, x_2^0) : T_0 = \frac{1}{s_0}(T_1 + x_0^0 G + x_1^0 X) \wedge S_0 = s_0 G \wedge Z_0 = \frac{1}{s_0}(Z_1 + x_0^0 C_0 + x_1^0 C_1 + x_2^0 G) \wedge \hat{X}_0^0 = x_0^0 \hat{G} \wedge \hat{X}_1^0 = x_1^0 \hat{G} \wedge \hat{X}_2^0 = x_2^0 \hat{G}]$,
- $\mathsf{ZKPoK}[(r, x_0^1, x_1^1, x_2^1) : T_1 = rS_0 + x_0^1 G + x_1^1 X \wedge Z_1 = rS_0 + x_0^1 C_0 + x_1^1 C_1 + x_2^1 G \wedge \hat{X}_0^1 = x_0^1 \hat{G} \wedge X_1^1 = x_1^1 \hat{G} \wedge X_1^2 = x_2^1 \hat{G}]$,
- $\mathsf{ZKPoK}[(r, s_1) : T = \frac{1}{s_1}(T_0 - rG) \wedge S = s_1 S_0 \wedge \hat{S} = s_1 \hat{S}_0 \wedge Z = \frac{1}{s_1}(Z_0 - rG)]$.

We stress that all ZKPoK involved are as simple to implement as a Schnorr proof. Let us now argue that the interactive variant produces signatures under the same distribution. Looking closer at how Z and T are computed, we have:

$$\begin{aligned} Z = \frac{1}{s_1}(Z_0 - rG) &= \frac{1}{s_1}\left(\frac{1}{s_0}\Big(Z_1 + x_0^0 C_0 + x_1^0 C_1 + x_2^0 G\Big) - rG\right) \\ &= \frac{1}{s_1}\left(\frac{1}{s_0}\Big(rs_0 G + \Big(x_0^0 + x_0^1\Big)C_0 + \Big(x_1^0 + x_1^1\Big)C_1 + \Big(x_2^0 + x_2^1\Big)G\Big) - rG\right) \\ &= \frac{1}{s_0 s_1}\Big(\Big(x_0^0 + x_0^1\Big)C_0 + \Big(x_1^0 + x_1^1\Big)C_1 + \Big(x_2^0 + x_2^1\Big)G\Big) \end{aligned}$$

Similarly, T is computed as:

$$\begin{aligned} T = \frac{1}{s_1}(T_0 - rG) &= \frac{1}{s_1}\left(\frac{1}{s_0}\Big(T_1 + x_0^0 G + x_1^0 X\Big) - rG\right) \\ &= \frac{1}{s_0 s_1}\Big(\Big(x_0^0 + x_0^1\Big)G + \Big(x_1^0 + x_1^1\Big)X\Big) + \frac{1}{s_1}\Big(\frac{1}{s_0} rs_0 G - rG\Big) \\ &= \frac{1}{s_0 s_1}\Big(\Big(x_0^0 + x_0^1\Big)G + \Big(x_1^0 + x_1^1\Big)X\Big) \end{aligned}$$

It follows that $s_0 s_1$, $x_0^0 + x_0^1$, $x_1^0 + x_1^1$ and $x_2^0 + x_2^1$ correspond to s, x_0, x_1 and x_2 in the single party variant. It remains to be seen that our two-party variant is also unforgeable (other properties are obviously taken over from the single-party construction). We reduce unforgeability of the two-party MSoRC to that of single-party MSoRC in a similar way as done in [ANKT25]. Consequently, we obtain the following theorem.

Theorem 2. *Our two-party* MSoRC *from Fig. 7 is unforgeable under Definition 8 (considering adversarially chosen encryption keys) if the single-party base* MSoRC *is unforgeablein the sense of Definition 3, and all* ZKPoK*'s are secure.*

Proof. For an adversary $\mathcal{A}'$ against the unforgeability game of Def. 8, we construct a simulator that, given access to $\mathcal{A}'$, plays the role of the adversary in the unforgeability game of Def. 3. The simulator gets pp and vk from the challenger. It then calls $\mathcal{A}$ on pp to obtain b and executes $\mathsf{SimTKGen}(\mathsf{vk}, b)$ to get $(\mathsf{sk}_b, \mathsf{vk}_0, \mathsf{vk}_1)$. Now the simulator invokes $\mathcal{A}'$ with $(\mathsf{sk}_b, \mathsf{vk}_0, \mathsf{vk}_1)$ as input. From this point onwards, $\mathcal{A}'$ can make signing queries and in the following we show that regardless the corruption case, the simulator is able to simulate the honest party and that such interaction is indistinguishable from the real execution

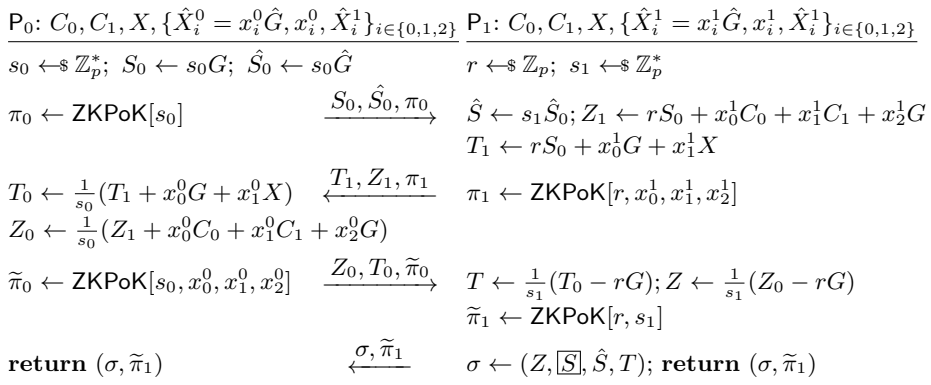

Fig. 7. Our two-party interactive signing algorithm.

in the view of $\mathcal{A}'$. Whenever $\mathcal{A}'$ queries a message, the simulator forwards the query to it's signing oracle and obtains a signature $(Z', S', \hat{S}', T')$. From there, the simulator proceeds as shown in Fig. 8 (left side for the case where $b = 0$ or right side for the case where $b = 1$), as corresponds.

We observe that in the first case (Fig. 8, left side), a real computation of Z_1 is indistinguishable from that of Z' as the former includes a uniformly random factor and the latter is uniformly random. This is also the case for T_1 and T'. Moreover, the zero-knowledge property of π_1 conceals this information. Looking at the second round, the simulated nature of σ cannot be distinguished by $\mathcal{A}'$ due to the soundness of both $\widetilde{\pi}_0$ and $\widetilde{\pi}_1$. The second case (Fig. 8, right side) is analogous to the first one. In both, the simulator outputs whatever $\mathcal{A}'$ outputs. Hence, whenever $\mathcal{A}'$ wins, the simulator wins. □

We claim that signature element S (boxed value S in Fig. 7) can be removed if encryption keys are honestly generated, and use this optimization to instantiate MSoRC in our mixnet construction.

3.4 Optimization

Namely, instead of allowing the adversary to choose the encryption key pair as done in Def. 3, we work with Def. 4 so that the challenger picks the encryption key pair. This relaxed security suffices for the optimized MSoRC to build mixnets where the encryption key is not under the user's control. This allows us to further optimize the previous construction by dropping S to obtain a shorter signature with optimal size.

This modification also reduces the number of pairings used in verification by two. Correctness, IND-CPA, full-class hiding, signature adaption and public key unlinkability directly follow from the previous proofs. Unforgeability needs to be proven from scratch as we cannot reduce the security of this version to that of the original scheme (signatures no longer have four elements). We provide a proof of the following theorem in Appendix D.

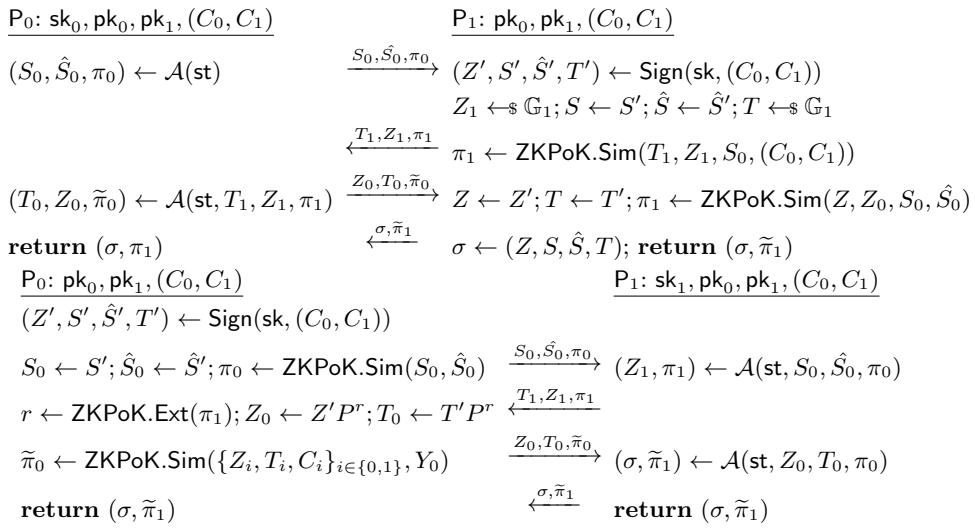

Fig. 8. Simulator's algorithm for corrupted P_0 (above) and for corrupted P_1 (below).

Theorem 3 (Unforgeability of our optimized MSoRC). *Our optimized scheme is unforgeable in the GGM as per definitions 3 and 8 if all ZKPoK's are secure.*

4 Mixnet from Two-Party MSoRC

4.1 Entities and Communication Model

As outlined before, our mixnet considers three main entities: users, CA, and mixers. As for the trust model, any users may be corrupted and behave maliciously. The CA is trusted to adhere to the protocol and thus follows the protocol as specified, but it is not trusted in terms of privacy. Mixers are not trusted to follow the protocol but at least one of them is trusted in terms of privacy. Our mixnet involves two more entities; an external verifier and a polling authority (PA). The external verifier assures the correctness of the input and output lists for the sake of soundness, and the involvement of all mix servers for the sake of privacy. PA is an entity that is responsible to decrypt the output ciphertexts. It is trusted in terms of privacy but not for soundness. To ease the trust, PA could be distributed among a set of entities that hold the decryption key and decrypt the output tuples in a distributed manner. Since this is a common approach, we do not discuss it in detail. These entities communicate through a publicly verifiable authenticated channel.

4.2 Construction

Our mixnet operates in four main phases: Setup, Input Certification, Mixing, and Verification, as illustrated in Figure 1. These are followed by a final decryption

phase performed by PA, which we omit here as it is straightforward. Below, we describe each phase in detail.

Setup Phase. All parameters and keys are sampled in this phase. For ease of exposition, we present this phase as a single algorithm, $\mathsf{Setup}(1^\kappa)$, run by a trusted party. The public parameters pp are published, and the secret keys are delivered to the respective parties privately.

$\underline{\mathsf{Setup}(1^\kappa):}$
$\mathsf{pp}_1 \leftarrow\!\!\$\ \mathsf{MSoRC.Setup}(1^\kappa)$ // $\mathsf{pp}_1 = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, G, \hat{G}, e)$
$\mathsf{pp}_2 \leftarrow\!\!\$\ \mathsf{NIZK.Setup}(1^\kappa)$
$(\mathsf{dk}, \mathsf{ek}) \leftarrow \mathsf{MSoRC.KeyGen}(\mathsf{pp}_1)$ // to PA
$(\mathsf{ask}, \mathsf{avk}) \leftarrow \mathsf{MSoRC.SKG}(\mathsf{pp}_1)$ // to CA
foreach $i \in [n]$ $(\mathsf{usk}_i, \mathsf{uvk}_i) \leftarrow \mathsf{MSoRC.SKG}(\mathsf{pp}_1)$ // to user i
$\mathsf{pp} := (\mathsf{pp}_1, \mathsf{pp}_2, \mathsf{ek}, \mathsf{avk}, \mathsf{uvk}_{i\in[n]})$ // publish

In practice, each key generation algorithm is performed independently by the respective party, *i.e.,* PA, CA, and each user. This assumes the standard certified-key setting, where all entities must prove knowledge of their secret keys.

Input Certification Phase. In this phase, ciphertexts from users are certified to form the initial input for the subsequent mixing phase. Each user and the CA collaboratively execute the protocols $\mathsf{EncAuth}_{u_i}$ and $\mathsf{EncAuth}_{\mathsf{CA}}$, respectively. At a high level, this process consists of the user encrypting their message, followed by a two-party MSoRC signing protocol (as depicted in Fig. 7) on the resulting ciphertext. The protocol is detailed below.

$\underline{\mathsf{EncAuth}_{u_i}(\mathsf{usk}_i, \mathsf{uvk}_i, \mathsf{avk}, M_i, \mathsf{ek}):}$		$\underline{\mathsf{EncAuth}_{\mathsf{CA}}(\mathsf{ask}, \mathsf{avk}, \mathsf{uvk}_i, \mathsf{ek}):}$
$\mathsf{ct}_i \leftarrow \mathsf{MSoRC.Enc}(\mathsf{ek}, M_i; \gamma)$		
$\pi \leftarrow \mathsf{ZKPoK}[(\gamma, \mathsf{usk}_i) : \mathsf{st1}]$	$\xrightarrow{\mathsf{ct}_i, \pi}$	$\mathsf{ct}'_i \leftarrow \mathsf{MSoRC.Rndmz}(\mathsf{ek}, \mathsf{ct}_i; \mu)$ $\pi' \leftarrow \mathsf{ZKPoK}[\mu : \mathsf{st2}]$
	$\xleftarrow{\mathsf{ct}'_i, \pi', \mathsf{evk}_i}$	$(\mathsf{esk}_i, \mathsf{evk}_i) \leftarrow\!\!\$\ \mathsf{MSoRC.SKG}()$
$\sigma_i \leftarrow \mathsf{MSoRC.ISign}_{\mathsf{P}_0}(\mathsf{usk}_i, \mathsf{ct}'_i)$	$\xleftrightarrow{\mathsf{MSoRC.ISign}}$	$\sigma_i \leftarrow \mathsf{MSoRC.ISign}_{\mathsf{P}_1}(\boxed{\mathsf{esk}_i + \mathsf{ask}}, \mathsf{ct}'_i)$
return $(\sigma_i, \mathsf{ct}'_i, \mathsf{evk}_i)$		**return** $(\sigma_i, \mathsf{ct}'_i, \mathsf{esk}_i, \mathsf{evk}_i)$

The protocol involves two zero-knowledge proofs, which are implicitly verified, and the protocol aborts if any verification fails. The first proof, provided by the user, establishes knowledge of (γ, usk_i) such that $\mathsf{st1} := (C_0 = \gamma G \wedge \mathsf{uvk}_i = \mathsf{usk}_i \hat{G})$, where C_0 is the first component of the ciphertext $\mathsf{ct}_i = (\gamma G, M_i + \gamma \mathsf{ek})$. The second proof, $\mathsf{st2}$, demonstrates that ct'_i is a correct re-randomization of ct_i, i.e., $\mathsf{st2} := (\mathsf{ct}'_i = \mathsf{MSoRC.Rndmz}(\mathsf{ek}, \mathsf{ct}_i; \mu))$. In the following, we clarify the rationale behind our design choices:
Ciphertext randomization by the CA is optional. While not essential for our mix-net's soundness or privacy, randomizing the ciphertext by the CA is beneficial in applications such as receipt-free voting, as it prevents users from demonstrating that a ciphertext decrypts to a specific message.

User's ZKPoK ensures plaintext knowledge and authentication. This proof prevents replay attacks in which an adversarial user could wait for another to submit a ciphertext, randomize it, and then obtain a signature on the same message.
Ephemeral key pairs $(\mathsf{esk}_i, \mathsf{evk}_i)$ on the CA side. Introduction of an ephemeral key pair by the CA protects against maliciously crafted user keys. Without this, a malicious user could generate keys in a correlated manner and collude with the first mix server to replace ciphertexts. The inclusion of the ephemeral public key ensures that each verification key is independent of the user's key generation. In the algorithm, we highlight this modification by indicating that the CA signs with $\mathsf{esk}_i + \mathsf{ask}$ instead of ask alone (boxed value in the protocol). Consequently, $\mathsf{MSoRC.ISign}$ produces a signature that verifies under $\mathsf{uvk}_i + \mathsf{evk}_i + \mathsf{avk}$.

Upon completion of the protocol between the CA and user i, the resulting tuple $(\sigma_i, \mathsf{ct}'_i, \mathsf{uvk}_i, \mathsf{evk}_i)$ is published in an authenticated manner. It is crucial to ensure that evk_i is the ephemeral key generated by the CA. For clarity and consistency, we assume that the CA is responsible for publishing these tuples.

This phase concludes once all users have completed the protocol with the CA. Let $\mathcal{CIL} := (\sigma_i, \mathsf{ct}'_i, \mathsf{uvk}_i, \mathsf{evk}_i)_{i\in[n]}$ denote the certified input list. For each entry, define $\mathsf{vk}_i := \mathsf{uvk}_i + \mathsf{evk}_i + \mathsf{avk}$; we treat vk_i as a virtual value, computed in this way whenever referenced. From this point onward, each tuple $(\mathsf{ct}'_i, \sigma_i, \mathsf{vk}_i)$ produced in this phase is relabeled with the superscript (0) as $(\mathsf{ct}_i^{(0)}, \sigma_i^{(0)}, \mathsf{vk}_i^{(0)})$. We then define the initial shuffle set as $\mathcal{SS}\mathsf{et}^{(0)} := (\mathsf{ct}_i^{(0)}, \sigma_i^{(0)}, \mathsf{vk}_i^{(0)})_{i\in[n]}$.

Mixing Phase. In this phase, the mix servers $1, \ldots, N$ operate sequentially in a cascade. Each mix server receives an input list, processes it, and passes the resulting output list to the next server. Let $\Pi[n]$ denote the set of all permutations over $[n]$, and let NIZK denote a non-interactive zero-knowledge proof system. Each mix server j executes the Mix algorithm as shown in Fig. 9.

Statement $\mathsf{st3} := (\mathsf{VK}^{(j)} = \mathsf{MSoRC.ConvertVK}(\mathsf{VK}^{(j-1)}, \rho))$, ensures correct transformation of the aggregated key. The algorithm proceeds as follows:

1. It first parses $\mathcal{SS}\mathsf{et}^{(j-1)}$ (the previous mix server's output) and samples fresh randomness μ, ρ, and a permutation Π_j from their respective domains.
2. For each verified input tuple $(\mathsf{ct}_i^{(j-1)}, \sigma_i^{(j-1)}, \mathsf{vk}_i^{(j-1)})$, it randomizes the verification key, signature, and ciphertext using the random factors ρ and μ via the corresponding MSoRC functions, and stores the result in a position determined by Π_j in the output list.
3. Finally, it aggregates the verification keys in both the input and output, and proves that they are correctly related via the common randomness ρ. With our MSoRC, this proof reduces to a lightweight equality of discrete logarithms among three pairs of elements.

Each mix server j publishes its output $(\mathcal{SS}\mathsf{et}^{(j)}, \mathsf{VK}^{(j)}, \pi^{(j)})$. The output list $\mathcal{SS}\mathsf{et}^{(j)}$ is then provided as input to the next mix server. The pair $(\mathsf{VK}^{(j)}, \pi^{(j)})$ forms a link in the proof chain that will be checked during the verification phase. The mixing phase concludes when the final mix server N publishes its output. Mix servers do not verify the proof from previous servers. While they could

$\underline{\mathsf{Mix}(\mathcal{SS}\mathsf{et}^{(j-1)})}$
$(\mathsf{ct}_i^{(j-1)}, \sigma_i^{(j-1)}, \mathsf{vk}_i^{(j-1)})_{i\in[n]} \leftarrow \mathcal{SS}\mathsf{et}^{(j-1)}$ //parse input list
$\mu, \rho \leftarrow\!\!\$\ \mathbb{Z}_p^*$; $\Pi_j \leftarrow\!\!\$\ \Pi[n]$ //sample random factors
foreach $i \in [n]$ **do** //randomize & permute
  if $\mathsf{MSoRC.Verify}(\mathsf{vk}_i^{(j-1)}, \mathsf{ct}_i^{(j-1)}, \sigma_i^{(j-1)}) \neq 1$ **abort**
  $\mathsf{vk}_{\Pi_j(i)}^{(j)} \leftarrow \mathsf{MSoRC.ConvertVK}(\mathsf{vk}_i^{(j-i)}, \rho)$
  $\sigma_{\Pi_j(i)}^{(j)} \leftarrow \mathsf{MSoRC.Adapt}(\sigma_i^{(j-1)}; \mu, \rho)$
  $\mathsf{ct}_{\Pi_j(i)}^{(j)} \leftarrow \mathsf{MSoRC.Rndmz}(\mathsf{ek}, \mathsf{ct}_i^{(j-1)}; \mu)$
$\mathcal{SS}\mathsf{et}^{(j)} := (\mathsf{ct}_i^{(j)}, \sigma_i^{(j)}, \mathsf{vk}_i^{(j)})_{i\in[n]}$ //compose output list
$\mathsf{VK}^{(j-1)} := \sum \mathsf{vk}_i^{(j-1)}$; $\mathsf{VK}^{(j)} := \sum \mathsf{vk}_i^{(j)}$ //aggregate I/O keys
$\pi^{(j)} \leftarrow \mathsf{NIZK.Prove}[\rho : \mathsf{st3}]$ //proof on aggregated keys
return $(\mathcal{SS}\mathsf{et}^{(j)}, \mathsf{VK}^{(j)}, \pi^{(j)})$

Fig. 9. Mix algorithm.

$\underline{\mathsf{Verify}(\mathcal{CIL}, \pi^{(1)}, \ldots, \pi^{(N)}, \mathsf{VK}^{(1)}, \ldots, \mathsf{VK}^{(N-1)}, \mathcal{SS}\mathsf{et}^{(N)}) :}$
$(\sigma_i, \mathsf{ct}_i', \mathsf{uvk}_i, \mathsf{evk}_i)_{i\in[n]} \leftarrow \mathcal{CIL}$ //parse I/O lists
$(\mathsf{ct}_i^{(N)}, \sigma_i^{(N)}, \mathsf{vk}_i^{(N)})_{i\in[n]} \leftarrow \mathcal{SS}\mathsf{et}^{(N)}$
foreach $i \in [n]$ **do** //check I/O entries
  if $\mathsf{MSoRC.Verify}(\mathsf{uvk}_i + \mathsf{evk}_i + \mathsf{avk}, \mathsf{ct}_i', \sigma_i) \neq 1$ **return** 0
  if $\mathsf{MSoRC.Verify}(\mathsf{vk}_i^{(N)}, \mathsf{ct}_i^{(N)}, \sigma_i^{(N)}) \neq 1$ **return** 0
$\mathsf{VK}^{(0)} := \sum_{i=1}^n \mathsf{uvk}_i + \mathsf{evk}_i + \mathsf{avk}$; $\mathsf{VK}^{(N)} := \sum_{i=1}^n \mathsf{vk}_i^{(N)}$ //aggregate I/O keys
foreach $j \in [N]$ **do** //check proof chain
  if $\mathsf{NIZK.Verify}(\pi^{(j)}, \mathsf{VK}^{(j-1)}, \mathsf{VK}^{(j)}) \neq 1$ **return** 0
return 1

Fig. 10. Verify algorithm.

perform this check and abort early upon failure, we defer all proof verification to the final verification phase for simplicity. However, each server does verify every entry in its input list using $\mathsf{MSoRC.Verify}$. Notably, this verification is essential for privacy rather than soundness, as detailed in the security proof.

Verification Phase. Any external party can verify the mixing process. Although all outputs are publicly available, the verifier only requires the certified input list $\mathcal{CIL}$, the sequence of proofs $(\pi^{(1)}, \ldots, \pi^{(N)})$, the aggregated keys $(\mathsf{VK}^{(1)}, \ldots, \mathsf{VK}^{(N-1)})$, and the final output list $\mathcal{SS}\mathsf{et}^{(N)}$ as input. Notably, the verifier does not need access to any intermediate output lists $\mathcal{SS}\mathsf{et}^{(j)}$ for $j = 1, \ldots, N-1$. The verification algorithm is shown in Fig. 10. The phase concludes when the verifier outputs 1 (success) or 0 (failure). In the event of failure, the application layer can determine the appropriate response. Since all data are authenticated and publicly available, identifying the source of any inconsistency is straightforward.

4.3 Achieving Constant-Size Proof

Following HPP20, we can eliminate the linear dependency on N by introducing a second round of interaction using the multi-signature from [BDN18](see Appendix F for details). Each mixer computes its own proof and a partial proof π', which is updated by multiplying in its witness. Unlike HPP20, which relies on GS proofs, we use the more efficient updatable proof system from Couteau and Hartmann (CH20) [CH20] (see Appendix C.2). The first mixer creates a proof, and each subsequent mixer updates it; at the end, π' attests to the relation between the initial and final tuples, allowing the entire process to be verified with a single proof. Mixers batch verify all proofs and jointly sign π' if they agree. Both the last individual proof and π' show knowledge of a witness for the final tuple, but π' relates it directly to the initial input. Batch verification ensures all servers participated honestly. Updating π' is efficient, requiring only a multiplication in $\mathbb{Z}_p$ and two exponentiations in $\mathbb{G}_2$ for the optimized two-key version of our MSoRC.

By allowing a second round of interaction, we can further simplify the above approach, originally envisioned in HPP20 and adapted to our setting. Since the NIZK proofs used in this work are simple discrete logarithm proofs, we can dispense with the CH20 proof system and instead rely on standard Schnorr proofs. Concretely, the process proceeds as follows:

1. Each mixer computes its individual proof $\pi^{(j)} \leftarrow \mathsf{NIZK.Prove}(\mathsf{VK}^{(j-1)}, \mathsf{VK}^{(j)}, \rho^{(j)})$ as a Schnorr proof, and records the randomizer $\rho^{(j)}$ used.
2. After the mixing phase, the mixers engage in a second round of interaction. First, they batch-verify all individual proofs as before. Then, they jointly produce a distributed Schnorr proof $\pi' \leftarrow \mathsf{NIZK.Prove}(\mathsf{VK}^{(0)}, \mathsf{VK}^{(N)}, \rho')$, where $\rho' = \prod_j \rho^{(j)}$. Finally, they sign π' using a multi-signature scheme.

4.4 Security Model

We strengthen the security model from HPP20 so that soundness and privacy hold against malicious users. For soundness, we guarantee that an adversary cannot successfully modify or replace messages of any user, including malicious ones. Similarly, our privacy notion ensures that messages in the input shuffle set are unlinkable from those in the output, even if some users and mixers collude.

Definition 10 (Soundness). *A mixnet is said to be sound 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 certification and encryption keys.*
2. *The adversary* $\mathcal{A}$ *then*

– *decides on the corrupted users* $\mathcal{I}^*$ *and generates itself their keys* $(\mathsf{uvk}_i)_{i\in\mathcal{I}^*}$
– *decides on the set* $\mathcal{I}$ *of users that will generate a message*
– *proves knowledge of the secret keys for each corrupted user in* $\mathcal{I}^*$ *to get the* MSoRC *signatures* σ_i *and ephemeral verification keys* evk_i *for ciphertexts of its choice*

– *generates the tuples* $(\mathcal{T}_i)_{i\in\mathcal{I}^*}$ *for the corrupted users and provides 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 tuples* $(\mathcal{T}_i)_{i\in\mathcal{I}\setminus\mathcal{I}^*}$*. The initial certified input list is thus defined by* $\mathcal{CIL} := (\mathcal{T}_i)_{i\in\mathcal{I}}$*.*
4. *The adversary mixes* $\mathcal{CIL}$ *in a provable way into* $(\mathcal{SS}\mathsf{et}, \mathsf{proof})$*.*

The adversary wins if $\mathsf{Verify}(\mathcal{CIL}, \mathcal{SS}\mathsf{et}, \mathsf{proof}) = 1$ *but* $\{\mathsf{Dec}^*(\mathcal{CIL})\} \neq \{\mathsf{Dec}^*(\mathcal{SS}\mathsf{et})\}$*, where* Dec^* *extracts the plaintexts using the decryption key.*

Theorem 4 (Soundness). *Our* Mixnet *is sound in the certified key setting if our* MSoRC *scheme is unforgeable and the proof system used is sound.*

Proof Sketch. We first note that if verification passes, soundness of the NIZKPoK proof guarantees that $\forall\ \mathsf{vk}'_i \in \mathcal{SS}\mathsf{et} \wedge \mathsf{vk}_i \in \mathcal{CIL} : \sum \mathsf{vk}'_i = \sum \alpha\mathsf{vk}_i$. This, together with the unforgeability of MSoRC, implies that $\forall\ \mathsf{vk}'_i : \mathsf{vk}'_i = \alpha(\mathsf{usk}_i + \mathsf{esk}_i + \mathsf{ask})\hat{G}$ since $[\mathsf{vk}'_i]_{\mathsf{vk}} = [\mathsf{vk}_i]_{\mathsf{vk}}$. Observe that for each usk_i (regardless of whether it is maliciously chosen or not), the value $\mathsf{esk}_i + \mathsf{ask}$ (framed in the signing protocol description) "fixes" the corresponding equivalence class. The class is unique and is outside the adversary's control since esk_i is chosen independently from usk_i and uniformly by the CA. This proves that the verification keys in the output shuffle set are a permutation of the ones in the input shuffle set. Consequently, the ciphertexts in the output shuffle set are also a permutation of the ciphertexts from the input shuffle set.

As originally defined in HPP20, the above soundness game (and proof) does not depend on the number of mixing steps. Instead, it considers a global mixing from the input to the output shuffle set. This reflects the use of a single (constant-size) proof. That said, we stress that one can apply a similar reasoning to prove that our base construction is also sound. More in detail, assume that the last proof $\pi^{(N)}$ verifies (*i.e.,* $\mathsf{VK}^{(N)}$ is a (correct) randomization of $\mathsf{VK}^{(N-1)}$) but there exists some $\mathsf{VK}^{(k-1)}$ for $0 < k \leq N-1$ such that $\mathsf{VK}^{(k)}$ is not a (correct) randomization of $\mathsf{VK}^{(k-1)}$. As outlined in the previous proof, such situation would lead to a contradiction. If the MSoRC scheme is unforgeable (*i.e.,* for all $[\mathsf{vk}_i^{(k)}]_{\mathsf{vk}} = [\mathsf{vk}_i^{(k-1)}]_{\mathsf{vk}}$) and $\pi^{(k)}$ verifies, then the proof system is not sound.

In the privacy game, the adversary provides two possible permutations for the case where the mix server follows the protocol and it wins if it can identify the permutation used. As expected, we will require the presence of one honest mixer to guarantee that at least one honest permutation is done during mixing.

Definition 11 (Privacy). *A mixnet is said to provide privacy 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 certification and encryption keys.*
2. *The adversary* $\mathcal{A}$ *then*

– *decides on the corrupted users* $\mathcal{I}^*$ *and generates itself their keys* $(\mathsf{uvk}_i)_{i\in\mathcal{I}^*}$

– *decides on the corrupted mixers* $\mathcal{J}^*$ *and generates itself their keys* $(\mathsf{spk}_i)_{i\in\mathcal{J}^*}$
– *decides on the set* $\mathcal{I}$ *of users that will generate a message*
– *decides on the set* $\mathcal{J}$ *of mixers that will make mixes*
– *proves its knowledge of the secret keys for each corrupted user in* $\mathcal{I}^*$ *to get the* MSoRC *signatures* σ_i *and keys* evk_i *for ciphertexts of its choice*
– *generates the message tuples* $(\mathcal{T}_i)_{i\in\mathcal{I}^*}$ *for corrupted users*

3. *The challenger generates keys for honest mixers* $(\mathsf{ssk}_j, \mathsf{spk}_j)_{j\in\mathcal{J}\setminus\mathcal{J}^*}$ *and the keys of the honest users* $(\mathsf{usk}_i, \mathsf{uvk}_i)_{i\in\mathcal{I}\setminus\mathcal{I}^*}$ *and their message tuples* $(\mathcal{T}_i)_{i\in\mathcal{I}^*}$.

The initial certified input list is thus defined by $\mathcal{CIL} = (\mathcal{T}_i)_{i\in\mathcal{I}}$. *The challenger randomly chooses a bit* b *and then enters into a loop for* $j \in \mathcal{J}$ *with the attacker:*

– *if* $j \in \mathcal{J}^*$, $\mathcal{A}$ *builds itself the new shuffle set* $\mathcal{SS}\mathsf{et}^{(j)}$ *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, then the challenger runs the mixing with* $\Pi_{j,b}$, *and provides the output* $(\mathcal{SS}\mathsf{et}^{(j)}, \mathsf{proof}^{(j)})$

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

Theorem 5 (Privacy). *Our* Mixnet *is private in the certified key setting if at least one mix server is honest, assuming the public key unlinkability and signature adaption of our* MSoRC *scheme, and the* SXDH *assumption.*

Proof. We analyze what happens when an honest mixer runs the protocol, showing that in the adversary's view the output shuffle set and proof are independent from the permutation chosen and any other information available to the adversary. Without loss of generality, we consider an honest mixer j that gets $\mathcal{SS}\mathsf{et}^{(j-1)} = \{((C_0, C_1)_i, \sigma_i, \Sigma_i, \mathsf{vk}_i)\}_{i\in[n]}^{(j-1)}$ and $\mathsf{proof}^{(j-1)}$. Soundness guarantees that $\mathcal{SS}\mathsf{et}^{(j-1)}$ is well-formed with respect to the initial tuple $\mathcal{SS}\mathsf{et}^{(0)}$. The challenger, running mixer j:

1. randomizes all $\mathsf{vk}_i \in \mathcal{SS}\mathsf{et}^{(j-1)}$ with $\rho^{(j)}$ to get $\mathsf{vk}_i^{(j)}$. Public key unlinkability of MSoRC guarantees that $\mathsf{vk}_i^{(j)}$ is unlinkable to the adversary (even knowing the secret key and previous randomizers of corrupted users and mixers).
2. randomizes each $(C_0, C_1)_i^{(j-1)}$ with $\mu^{(j)}$ and adapts $\Sigma_i^{(j-1)}$ with $\mu^{(j)}$ and $\rho^{(j)}$ to get $(C_0, C_1)_i^{(j)}$ and $\Sigma_i^{(j)}$. On the one hand, security of ElGamal under DDH ensures that $(C_0, C_1)_i^{(j)}$ is unlinkable to $(C_0, C_1)_i^{(j-1)}$. On the other hand, signature adaption of MSoRC guarantees that $\Sigma_i^{(j)}$ looks like a freshly computed signature for $(C_0, C_1)_i^{(j)}$ and thus, unlinkable to $\Sigma_i^{(j-1)}$ for all i.

Multiple Ciphertexts. As discussed in Appendix A (where we present a detailed discussion on applying our scheme to e-voting), the encryption key pair can be distributed among a set of trustees (*e.g.,* as in [CGGI13]). Besides, longer plaintexts may have to be supported for complex voting rules or to allow redundant encoding for the convenience of final counting. The authors of [BF20] discussed how their SoRC scheme can be generalized to sign a vector of ElGamal ciphertexts without increasing signature size. The idea is to define a key

vector so that multiple ciphertexts can be encrypted using the same randomness. Our construction is compatible with such generalization, allowing users to obtain a single signature for multiple ciphertexts. Given an encryption key $\mathbf{ek} = (\mathsf{ek}_1, \cdots, \mathsf{ek}_n)$, a signing key $(x_0, \cdots, x_{n+1})$, a ciphertext consisting of $C_0 = rG$ and $C_i = M_i + r\mathsf{ek}_i$ for $1 \leq i \leq n$, the signature is:

$$Z := \frac{1}{s}\left(\sum_{i=0}^{i=n} x_i C_i + x_{n+1} G\right), T := \frac{1}{s}\left(x_0 G + \sum_{i=1}^{i=n} x_i \mathsf{ek}_i\right), \hat{S} := s\hat{G}$$

This way, users can encrypt, *e.g.,* the ranking preference for each candidate keeping the signature size constant. Since every vote is decrypted individually, the validity of each vote can be verified at decryption time and malformed votes can be discarded. This contrasts with homomorphic voting schemes like [CFSY96] for which adding such functionality is costly and non-trivial.

4.5 Performance Evaluation

Comparison. We compare our mixnet with the works by Hébant *et al.* [HPP20] and Faonio and Russo [FR22] (Rand-RCCA), presenting asymptotic computational and communication costs.[2]

Computational and communication costs for verification in HPP20 consider the use of a multi-signature as originally reported by the authors. Consequently, for HPP20, we include verification costs of the individual proofs required to produce the multi-signature as part of the mixing computational costs. HPP20 does not consider a bulletin board to authenticate the mixers' proofs. Instead, it considers the use of a generic signature scheme. To make a fair comparison, we consider the use of BLS [BLS04] as it is highly efficient and compatible with their setting. In our case, we consider the standard scenario where verification depends linearly on the number of mixers and the use of an aggregate signature scheme to authenticate all proofs emulating a bulletin board. We use the sequential aggregate signature (SAS) from Pointcheval and Sanders [PS16] (see Appendix F for details) as it suits our setting. In our case, the mixers do not need to verify individual proofs, but they verify the partial aggregate signature before mixing, favouring a comparison with HPP20. Therefore, we report the computational cost that corresponds to the last mixer who has to perform N exponentiations in $\mathbb{G}_2$ to verify the messages from all previous servers. For in and out communication we include the server's public keys needed to verify the signatures and related messages (considering their original representation with sizes in source group instead of $\mathbb{Z}_p$). We recall that our construction achieves a stronger security model compared to HPP20 and is, therefore, more competitive.

Comparison with [FR22] requires us to make some assumptions since NIZK proofs $\mathsf{NIZK}_{\mathsf{snd}}$ and $\mathsf{NIZK}_{\mathsf{mx}}$ are not fully specified in their works [FFHR19, FR22,

[2] We note that Faonio *et al.* initially proposed the use of Rand-RCCA PKE as a building block to construct mixnets in [FFHR19]. There, the Rand-RCCA PKE needs to provide public verifiability. In [FR22] the authors manage to get rid of such requirement.

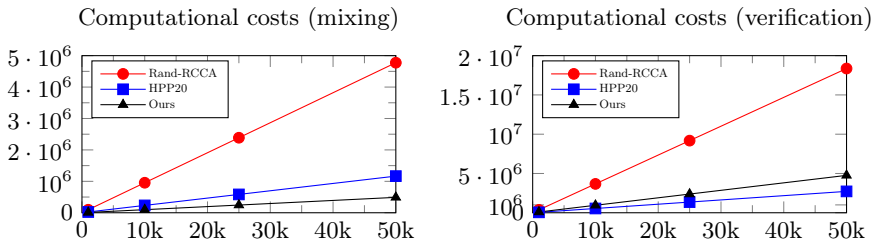

Fig. 11. Computational costs considering group exponentiations and pairings for a number of users from 1k to 50k and 10 mixers. Y axis is the total count of all group exponentiations and pairings relative to exponentiation cost in $\mathbb{G}_1$.

FHR23]. Consequently, we make the simplifying assumptions (which are in their favor) that for $\mathsf{NIZK}_{\mathsf{mx}}$ we have a simple adaptively sound QA-NIZK due to Kiltz and Wee [KW15], which under SXDH has a proof size of $2\mathbb{G}_1$ elements, and a Groth-Sahai NIZK for $\mathsf{NIZK}_{\mathsf{snd}}$ (just considering pairing product equations) with a size of $4\mathbb{G}_1 + 4\mathbb{G}_2$ elements. This allows us to compare the three approaches in detail, providing concrete costs as shown in Appendix G.

We consider the BLS12-381 curve where sizes of group elements in bits are as follows: $|\mathbb{G}_2| = 2 \cdot |\mathbb{G}_1|$, $|\mathbb{G}_1| = 2 \cdot |\mathbb{Z}_p|$, $|\mathbb{Z}_p| = 256$ and $|\mathbb{G}_T| = 12 \cdot 381$. For the scalar multiplications in the groups $\mathbb{G}_1$ and $\mathbb{G}_2$, the exponentiation in group $\mathbb{G}_T$ as well as pairing computation, we have that scalar multiplications in $\mathbb{G}_1$ are the cheapest and the operations in $\mathbb{G}_2$, $\mathbb{G}_T$ and P are a factor of 2 as well as 7 more expensive than in $\mathbb{G}_1$. These relations were taken based on the measurements of these operations using the BLS12-381 curve and we use them to determine dominant operations in our asymptotic computational costs.

Computational comparisons are shown in Fig. 11 whereas communication costs are shown in Fig. 12. Firstly, we observe that HPP20 and our approach only linearly depend on the parameters n and N. In contrast [FR22] have a dependency on $n \cdot N$ in the verification costs and generally higher computational and bandwidth costs overall. When taking a closer comparison with the more scaleable solution (HPP20), our effort for verification is comparable (even for the variant where we are linear in N as typically $N \leq 10$), but in all other aspects we improve. For instance, if one sets $n = 1000$ and $N = 10$, mixing is around $3.5x$ more efficient with our approach and bandwidth savings are around $1.5x$ (for inputs as well as outputs to mixing) and around $3x$ for the optimized case (and $1.1x$ for the unoptimized one).

Experimental Results. We implemented a prototype of our protocols in Rust using the blasters library [Lab21], which implements the pairing-friendly BLS12-381 curve. BLAKE3 [OANWO20] was used to instantiate hash functions. Source code and documentation to reproduce our results are available on GitHub [Nan25]. We used Rust's Criterion library and the nightly compiler with no extra optimizations to run the benchmarks on a MacBook Pro M3 with 32GB of RAM.

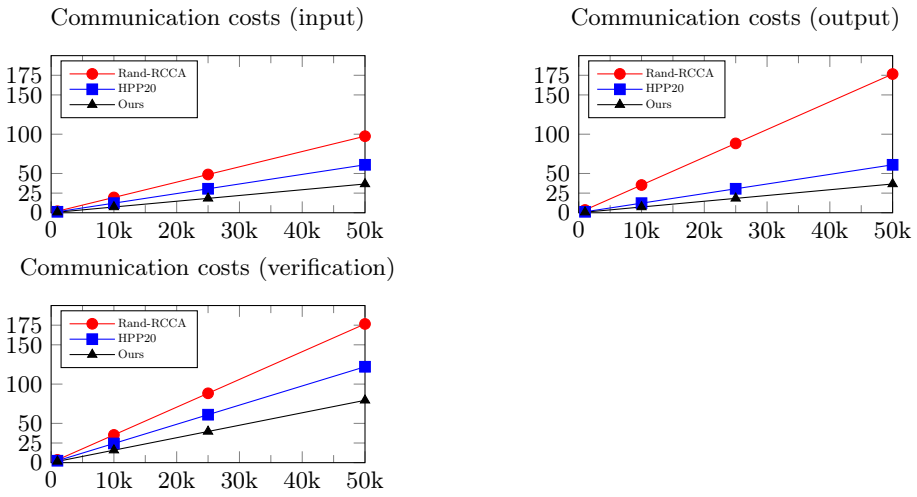

Fig. 12. Communication costs in MB for a number of users from 1k to 50k and 10 mixers.

Table 1. Running times of each protocol in seconds.

n	InputVerification	Mix	Verify ($N = 5$)	Verify ($N = 10$)
1k	2.7	0.8	2.7	2.7
10k	27.1	8.3	27	27
25k	67.6	20.7	67.4	67.4
50k	135	41.3	134.6	134.5

The (interactive) signing protocol of our MSoRC scheme (Fig. 7) takes 6.4ms while EncAuth (which includes the ZKPoK's) takes 8.1ms.

Running times of other protocols are summarized in Table 1, confirming the linear complexity of our mixnet scheme. In all cases, the standard deviation was below 1s. We note that a pairing takes around 380 microseconds while a multi-exponentiation for $N = 10$ takes 737 microseconds. Thus, for a small N, the difference between our standard Verify (where we verify N proofs in batch) and the constant-size variant of it (where a single proof is verified) is negligible.

Our prototype does not make use of parallelization libraries such as Rayon. However, our scheme is highly compatible with such techniques due to the individual processing of tuples during mixing and verification. Moreover, practical deployments would use proper servers, allowing our solution to scale further.

5 Conclusion

In this work, we introduced a new certified-input mixnet construction based on the novel concept of mercurial signatures on randomizable ciphertexts (MSoRC).

Our approach advances the state of the art by achieving both improved efficiency and stronger security guarantees compared to previous frameworks such as HPP20. In particular, our two-party MSoRC provides public-key unlinkability, which is crucial for protecting voter privacy in e-voting systems against collusion between users, mix servers, and the certificate authority.

The modularity of our construction facilitates integration and future improvements, making it adaptable to evolving cryptographic primitives. Through careful design and optimization, we demonstrated that our protocol is both scalable and practical. Our implementation and benchmarks show that, for 50k voters and 10 mix servers, the worst-case mixing time is around 40 seconds, and the entire process completes in under 5 minutes on a commodity laptop, without any parallelization. We believe our contributions will enable the development of more efficient and practical mixnet protocols for privacy-sensitive applications.

References

Abe98. Masayuki Abe. Universally verifiable mix-net with verification work independent of the number of mix-servers. In Kaisa Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 437–447. Springer, Heidelberg, May / June 1998.

Abe99. Masayuki Abe. Mix-networks on permutation networks. In Kwok-Yan Lam, Eiji Okamoto, and Chaoping Xing, editors, *ASIACRYPT'99*, volume 1716 of *LNCS*, pages 258–273. Springer, Heidelberg, November 1999.

ABG+21. Diego F. Aranha, Carsten Baum, Kristian Gjøsteen, Tjerand Silde, and Thor Tunge. Lattice-based proof of shuffle and applications to electronic voting. In Kenneth G. Paterson, editor, *CT-RSA 2021*, volume 12704 of *LNCS*, pages 227–251. Springer, Heidelberg, May 2021.

ABGS23a. Diego F. Aranha, Carsten Baum, Kristian Gjøsteen, and Tjerand Silde. Verifiable mix-nets and distributed decryption for voting from lattice-based assumptions. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023*, pages 1467–1481. ACM, 2023.

ABGS23b. Diego F. Aranha, Carsten Baum, Kristian Gjøsteen, and Tjerand Silde. Verifiable mix-nets and distributed decryption for voting from lattice-based assumptions. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, CCS '23, page 1467–1481, New York, NY, USA, 2023. Association for Computing Machinery.

ABR23. Diego F. Aranha, Michele Battagliola, and Lawrence David Roy. Faster coercion-resistant e-voting by encrypted sorting. In *Proceedings of E-Vote-ID 2023*. Tartu University Press, June 2023. 8th International Joint Conference on Electronic Voting, E-Vote-ID ; Conference date: 03-10-2023 Through 06-10-2023.

Adi08. Ben Adida. Helios: Web-based open-audit voting. In Paul C. van Oorschot, editor, *USENIX Security 2008*, pages 335–348. USENIX Association, July / August 2008.

AF04. Masayuki Abe and Serge Fehr. Adaptively secure feldman VSS and applications to universally-composable threshold cryptography. In Matthew K. Franklin, editor, *Advances in Cryptology - CRYPTO 2004, 24th Annual*

International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings, volume 3152 of *Lecture Notes in Computer Science*, pages 317–334. Springer, 2004.

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.

AGHO11. Masayuki Abe, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Optimal structure-preserving signatures in asymmetric bilinear groups. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 649–666. Springer, Heidelberg, August 2011.

AH01. Masayuki Abe and Fumitaka Hoshino. Remarks on mix-network based on permutation networks. In Kwangjo Kim, editor, *PKC 2001*, volume 1992 of *LNCS*, pages 317–324. Springer, Heidelberg, February 2001.

AHIV17. Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 2087–2104. ACM Press, October / November 2017.

AKA+21. Khaleel Ahmad, Afsar Kamal, Khairol Amali Bin Ahmad, Manju Khari, and Rubén González Crespo. Fast hybrid-mixnet for security and privacy using NTRU algorithm. *J. Inf. Secur. Appl.*, 60:102872, 2021.

ANKT25. Masayuki Abe, Masaya Nanri, Octavio Perez Kempner, and Mehdi Tibouchi. Interactive threshold mercurial signatures and applications. In Kai-Min Chung and Yu Sasaki, editors, *Advances in Cryptology – ASIACRYPT 2024*, pages 69–103, Singapore, 2025. Springer Nature Singapore.

BCC+09. Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. Randomizable proofs and delegatable anonymous credentials. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 108–125. Springer, Heidelberg, August 2009.

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.

BF20. Balthazar Bauer and Georg Fuchsbauer. Efficient signatures on randomizable ciphertexts. In Clemente Galdi and Vladimir Kolesnikov, editors, *SCN 20*, volume 12238 of *LNCS*, pages 359–381. Springer, Heidelberg, September 2020.

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.

BG02. Dan Boneh and Philippe Golle. Almost entirely correct mixing with applications to voting. In Vijayalakshmi Atluri, editor, *ACM CCS 2002*, pages 68–77. ACM Press, November 2002.

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.

BGR98. Mihir Bellare, Juan A. Garay, and Tal Rabin. Fast batch verification for modular exponentiation and digital signatures. In Kaisa Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 236–250. Springer, Heidelberg, May / June 1998.

BGR12. Sergiu Bursuc, Gurchetan S. Grewal, and Mark D. Ryan. Trivitas: Voters directly verifying votes. In Aggelos Kiayias and Helger Lipmaa, editors, *E-Voting and Identity*, pages 190–207, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

BHKS18. Michael Backes, Lucjan Hanzlik, Kamil Kluczniak, and Jonas Schneider. Signatures with flexible public key: Introducing equivalence classes for public keys. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 405–434. Springer, Heidelberg, December 2018.

BHM20. Xavier Boyen, Thomas Haines, and Johannes Müller. A verifiable and practical lattice-based decryption mix net with external auditing. In Liqun Chen, Ninghui Li, Kaitai Liang, and Steve A. Schneider, editors, *ESORICS 2020, Part II*, volume 12309 of *LNCS*, pages 336–356. Springer, Heidelberg, September 2020.

BLS04. Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. *Journal of Cryptology*, 17(4):297–319, September 2004.

CCM08. Michael R. Clarkson, Stephen Chong, and Andrew C. Myers. Civitas: Toward a secure voting system. In *2008 IEEE Symposium on Security and Privacy*, pages 354–368. IEEE Computer Society Press, May 2008.

CFSY96. Ronald Cramer, Matthew K. Franklin, Berry Schoenmakers, and Moti Yung. Multi-autority secret-ballot elections with linear work. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 72–83. Springer, Heidelberg, May 1996.

CGGI13. Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachène. Distributed elgamal à la pedersen: Application to helios. In *Proceedings of the 12th ACM Workshop on Workshop on Privacy in the Electronic Society*, WPES '13, page 131–142, New York, NY, USA, 2013. Association for Computing Machinery.

CGY24. Véronique Cortier, Pierrick Gaudry, and Quentin Yang. Is the JCJ voting system really coercion-resistant? In *37th IEEE Computer Security Foundations Symposium (CSF)*, CSF 2024, Enschede, Netherlands, 2024. IEEE. This is the long version of the paper published at CSF 2024.

CH20. Geoffroy Couteau and Dominik Hartmann. Shorter non-interactive zero-knowledge arguments and ZAPs for algebraic languages. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 768–798. Springer, Heidelberg, August 2020.

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

Cha25. Chainlink. Chainlink network. Online, 2025.

CKLM12. Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Malleable proof systems and applications. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 281–300. Springer, Heidelberg, April 2012.

CKLM13. Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Verifiable elections that scale for free. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 479–496. Springer, Heidelberg, February / March 2013.

CKN03. Ran Canetti, Hugo Krawczyk, and Jesper Buus Nielsen. Relaxing chosen-ciphertext security. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 565–582. Springer, Heidelberg, August 2003.

CL19. Elizabeth C. Crites and Anna Lysyanskaya. Delegatable anonymous credentials from mercurial signatures. In Mitsuru Matsui, editor, *CT-RSA 2019*, volume 11405 of *LNCS*, pages 535–555. Springer, Heidelberg, March 2019.

CL21. Elizabeth C. Crites and Anna Lysyanskaya. Mercurial signatures for variable-length messages. *PoPETs*, 2021(4):441–463, October 2021.

CL24. Yi-Hsiu Chen and Yehuda Lindell. Feldman's verifiable secret sharing for a dishonest majority. *IACR Cryptol. ePrint Arch.*, page 31, 2024.

CLPK22. Aisling Connolly, Pascal Lafourcade, and Octavio Perez-Kempner. Improved constructions of anonymous credentials from structure-preserving signatures on equivalence classes. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *PKC 2022, Part I*, volume 13177 of *LNCS*, pages 409–438. Springer, Heidelberg, March 2022.

CLW08. Sherman S. M. Chow, Joseph K. Liu, and Duncan S. Wong. Robust receipt-free election system with ballot secrecy and verifiability. In *NDSS 2008*. The Internet Society, February 2008.

CP93. David Chaum and Torben Pryds Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, *Advances in Cryptology — CRYPTO' 92*, pages 89–105, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

CS02. Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 45–64. Springer, Heidelberg, April / May 2002.

CS11. Véronique Cortier and Ben Smyth. Attacking and fixing helios: An analysis of ballot secrecy. In Michael Backes and Steve Zdancewic, editors, *CSF 2011 Computer Security Foundations Symposium*, pages 297–311. IEEE Computer Society Press, 2011.

CVM25. Diego Castejon-Molina, Dimitrios Vasilopoulos, and Pedro Moreno-Sanchez. Mixbuy: Contingent payment in the presence of coin mixers. *Proc. Priv. Enhancing Technol.*, 2025(1):671–706, 2025.

DEF+25. Bernardo David, Felix Engelmann, Tore Frederiksen, Markulf Kohlweiss, Elena Pagnin, and Mikhail Volkhov. Updatable privacy-preserving blueprints, 2025.

ElG86. Taher ElGamal. On computing logarithms over finite fields. In Hugh C. Williams, editor, *CRYPTO'85*, volume 218 of *LNCS*, pages 396–402. Springer, Heidelberg, August 1986.

FFHR19. Antonio Faonio, Dario Fiore, Javier Herranz, and Carla Ràfols. Structure-preserving and re-randomizable RCCA-secure public key encryption and its applications. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part III*, volume 11923 of *LNCS*, pages 159–190. Springer, Heidelberg, December 2019.

FGHP09. Anna Lisa Ferrara, Matthew Green, Susan Hohenberger, and Michael Østergaard Pedersen. Practical short signature batch verification. In Marc Fischlin, editor, *CT-RSA 2009*, volume 5473 of *LNCS*, pages 309–324. Springer, Heidelberg, April 2009.

FHR23. Antonio Faonio, Dennis Hofheinz, and Luigi Russo. Almost tightly-secure re-randomizable and replayable CCA-secure public key encryption. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023, Part II*, volume 13941 of *LNCS*, pages 275–305. Springer, Heidelberg, May 2023.

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.

FR22. Antonio Faonio and Luigi Russo. Mix-nets from re-randomizable and replayable cca-secure public-key encryption. In Clemente Galdi and Stanislaw Jarecki, editors, *Security and Cryptography for Networks*, pages 172–196, Cham, 2022. Springer International Publishing.

FS87. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology — CRYPTO' 86*, pages 186–194, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg.

GGPR13. Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645. Springer, Heidelberg, May 2013.

Gol01. Oded Goldreich. *Foundations of Cryptography: Basic Tools*, volume 1. Cambridge University Press, Cambridge, UK, 2001.

Gro16. Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Heidelberg, May 2016.

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.

GWC19. Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. https://eprint.iacr.org/2019/953.

Hea07. James Heather. Implementing stv securely in pret a voter. In *20th IEEE Computer Security Foundations Symposium (CSF'07)*, pages 157–169, 2007.

HL10. Carmit Hazay and Yehuda Lindell. *Efficient Secure Two-Party Protocols: Techniques and Constructions*. Information Security and Cryptography. Springer, Berlin, Heidelberg, 2010.

HMMP23. Thomas Haines, Rafieh Mosaheb, Johannes Müller, and Ivan Pryvalov. Sok: Secure e-voting with everlasting privacy. *Proc. Priv. Enhancing Technol.*, 2023(1):279–293, 2023.

HMQA23. Thomas Haines, Johannes Müller, and Iñigo Querejeta-Azurmendi. Scalable coercion-resistant e-voting under weaker trust assumptions. In *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*, SAC '23, page 1576–1584, New York, NY, USA, 2023. Association for Computing Machinery.

HMS21. Javier Herranz, Ramiro Martínez, and Manuel Sánchez. Shorter lattice-based zero-knowledge proofs for the correctness of a shuffle. In *Financial Cryptography and Data Security. FC 2021 International Workshops*

- *CoDecFin, DeFi, VOTING, and WTSC, Virtual Event, March 5, 2021, Revised Selected Papers*, volume 12676 of *Lecture Notes in Computer Science*, pages 315–329. Springer, 2021.

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

HS14. Christian Hanser and Daniel Slamanig. Structure-preserving signatures on equivalence classes and their application to anonymous credentials. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 491–511. Springer, Heidelberg, December 2014.

JCJ05. Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society*, WPES '05, page 61–70, New York, NY, USA, 2005. Association for Computing Machinery.

Kat23. Jonathan Katz. Round optimal robust distributed key generation. *IACR Cryptol. ePrint Arch.*, page 1094, 2023.

KER+22. Christian Killer, Moritz Eck, Bruno Rodrigues, Jan von der Assen, Roger Staubli, and Burkhard Stiller. Provotumn: Decentralized, mix-net-based, and receipt-free voting system. In *2022 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–9, 2022.

KLN23. Markulf Kohlweiss, Anna Lysyanskaya, and An Nguyen. Privacy-preserving blueprints. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 594–625. Springer, Heidelberg, April 2023.

KW15. Eike Kiltz and Hoeteck Wee. Quasi-adaptive NIZK for linear subspaces revisited. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 101–128. Springer, Heidelberg, April 2015.

Lab21. Protocol Labs. High performance implementation of bls12 381. Online, 2021.

LOS+06. Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential aggregate signatures and multisignatures without random oracles. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 465–485. Springer, Heidelberg, May / June 2006.

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.

LQT20. Wouter Lueks, Iñigo Querejeta-Azurmendi, and Carmela Troncoso. VoteAgain: A scalable coercion-resistant voting system. In Srdjan Capkun and Franziska Roesner, editors, *USENIX Security 2020*, pages 1553–1570. USENIX Association, August 2020.

MMR22. David Mestel, Johannes Müller, and Pascal Reisert. How efficient are replay attacks against vote privacy? a formal quantitative analysis. In *2022 IEEE 35th Computer Security Foundations Symposium (CSF)*, pages 179–194, 2022.

MN06. Tal Moran and Moni Naor. Receipt-free universally-verifiable voting with everlasting privacy. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 373–392. Springer, Heidelberg, August 2006.

MN07. Tal Moran and Moni Naor. Split-ballot voting: everlasting privacy with distributed trust. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM CCS 2007*, pages 246–255. ACM Press, October 2007.

MRV16. Paz Morillo, Carla Ràfols, and Jorge Luis Villar. The kernel matrix Diffie-Hellman assumption. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 729–758. Springer, Heidelberg, December 2016.

MTV+23. Varun Madathil, Sri Aravinda Krishnan Thyagarajan, Dimitrios Vasilopoulos, Lloyd Fournier, Giulio Malavolta, and Pedro Moreno-Sanchez. Cryptographic oracle-based conditional payments. In *30th Annual Network and Distributed System Security Symposium, NDSS 2023, San Diego, California, USA, February 27 - March 3, 2023*. The Internet Society, 2023.

Nan25. Masaya Nanri. Implementation of mixnets from msorc. https://github.com/octaviopk9/esorics_msorc, 2025.

OANWO20. Jack O'Connor, Jean-Philippe Aumasson, Samuel Neves, and Zooko Wilcox-O'Hearn. Blake3. Online, 2020.

Oka97. Tatsuaki Okamoto. Receipt-free electronic voting schemes for large scale elections. In Bruce Christianson, Bruno Crispo, T. Mark A. Lomas, and Michael Roe, editors, *Security Protocols, 5th International Workshop*, volume 1361 of *LNCS*, pages 25–35, Paris, France, April 7–9 1997. Springer, Heidelberg.

PB09. Kun Peng and Feng Bao. A design of secure preferential e-voting. In Peter Y. A. Ryan and Berry Schoenmakers, editors, *E-Voting and Identity, Second International Conference, VoteID 2009, Luxembourg, September 7-8, 2009. Proceedings*, volume 5767 of *Lecture Notes in Computer Science*, pages 141–156. Springer, 2009.

Poi23. David Pointcheval. Linearly-homomorphic signatures for short randomizable proofs of subset membership. In Melanie Volkamer, David Duenas-Cid, Peter B. Roenne, Peter Y. A. Ryan, Jurlind Budurushi, Oksana Kulyk, Adrià Rodríguez-Pérez, Iuliia Spycher-Krivonosova, Michael Kirsten, Alexandre Debant, and Nicole J. Goodman, editors, *Eight International Joint Conference on Electronic Voting, E-Vote-ID 2024, Luxembourg City, Luxembourg, October 3-6, 2023, Proceedings*, volume P347 of *LNI*. Gesellschaft für Informatik e.V., 2023.

Poi24. David Pointcheval. Efficient universally-verifiable electronic voting with everlasting privacy. In Clemente Galdi and Duong Hieu Phan, editors, *Security and Cryptography for Networks - 14th International Conference, SCN 2024, Amalfi, Italy, September 11-13, 2024, Proceedings, Part I*, volume 14973 of *Lecture Notes in Computer Science*, pages 323–344. Springer, 2024.

PS16. David Pointcheval and Olivier Sanders. Short randomizable signatures. In Kazue Sako, editor, *CT-RSA 2016*, volume 9610 of *LNCS*, pages 111–126. Springer, Heidelberg, February / March 2016.

PS18. David Pointcheval and Olivier Sanders. Reassessing security of randomizable signatures. In Nigel P. Smart, editor, *CT-RSA 2018*, volume 10808 of *LNCS*, pages 319–338. Springer, Heidelberg, April 2018.

Sch80. J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, oct 1980.

Sch91. C. P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

SK95. Kazue Sako and Joe Kilian. Receipt-free mix-type voting scheme - a practical solution to the implementation of a voting booth. In Louis C. Guillou and Jean-Jacques Quisquater, editors, *EUROCRYPT'95*, volume 921 of *LNCS*, pages 393–403. Springer, Heidelberg, May 1995.

Yan23. Quentin Yang. *Résistance à la coercition en vote électronique : conception et analyse. (Coercion-resistance in electronic voting : design and analysis).* PhD thesis, University of Lorraine, Nancy, France, 2023.

Appendix

A Application to Receipt-Free E-voting

As evidenced by the vast literature (see *e.g.,* [SK95, Abe98, Abe99, AH01, BG02, Adi08, CKLM13, LQT20, KER+22, ABGS23b]), voting (or e-voting) is by far the most popular application of mixnets. We demonstrate that our mixnet construction naturally supports a receipt-free e-voting scheme.

Our scheme follows the standard blueprint of mix-type e-voting. There are voters, a certificate authority (CA), mix servers (MX), and tally servers (TA). We implicitly use a trustful bulletin board (BB) that records all published data authentically and in a non-erasable manner. The election process consists of four phases, *i.e.,* setup, registration, vote casting, and tallying, which correspond to our mixnet procedures.

Setup phase. Setup and MixKG are executed by relevant entities. Each voter u_i generates a key pair $(\mathsf{usk}_i, \mathsf{uvk}_i)$. CA generates the public parameters and key pair $(\mathsf{ask}, \mathsf{avk})$. The tally servers generate an ElGamal encryption key pair, dk and ek, by running a secure distributed key generation protocol, *e.g.,* [Kat23, AF04, CL24]. All public parameters and verification keys are published authentically.

Registration phase. Once the voting phase begins, u_i decides their vote M_i and engages in $\mathsf{EncAuth}$ with the CA. This process is one-time for each voter. Voter u_i obtains an encrypted and signed ballot $(\sigma_i, C'_i, \mathsf{uvk}_i, \mathsf{evk}_i)$. In $\mathsf{EncAuth}$, CA's proof of re-randomization, $\pi' \leftarrow \mathsf{ZKPoK}[\mu : C'_i = \mathsf{Rndmz}(\mathsf{ek}, C_i; \mu)]$, must be done in a simulatable manner for the sake of receipt-freeness. The standard five-round augmentation of sigma-protocols provides fully simulatable zero-knowledge (see, *e.g.,* Chapter 6 of [HL10]). A sigma-protocol for disjunctive coupling of the statement with a knowledge of secret-key usk_i gives a non-interactive designated verifier proof in the ROM that also suffices for the purpose.

Casting phase. Each voter casts their ballot $(\sigma_i, C'_i, \mathsf{uvk}_i)$ on BB. Communication happens over a public channel, and the process is done only once. At the end of this phase, all evk_i corresponding to cast ballots are published by the CA.

Tallying phase. InputVerification is invoked to screen irregular votes. It is a public process that can be executed by, *e.g.,* a representative of mix servers. Each mix server executes Mix in order and Verify at the end. Once the verification passes, the tallying servers decrypt every verified ciphertext with distributed ElGamal decryption and publish a proof of correct decryption. The final result is publicly computed from the decryption result published on BB.

A.1 Security

First, we clarify which authority is trusted for which property.

- CA: Trusted for verifiability, which relies on the unforgeability of the CA's signatures. Untrusted for ballot privacy. Trusted for receipt-freeness.
- MX: Untrusted for verifiability and receipt freeness. At least 1 server is trusted for privacy.
- TA: Untrusted for verifiability and receipt freeness. At least k-out-of-N servers are trusted for privacy.
- BB: Trusted for all properties. It authentically holds data, *i.e.,* it is publicly verifiable who wrote what.

No trust is assumed on voters for any property.

Receipt-freeness. Receipt-freeness inherently requires a moment when the coercer does not monitor or control every user. We require absence of the coercer during the execution of EncAuth. Concretely, we consider Moran and Naor's standard notion of receipt-freeness [MN06], where the adversary cannot fully monitor/control a voter. It considers an untappable channel (analogous to a private polling booth) as a minimal assumption, and it is assumed that the adversary is absent while the voter is using it since the shoulder attack would work otherwise. Hence, we consider the absence of the coercer during the execution of the signing process, a minimal assumption in the previous sense. If the coercer attempts an attack before, the user can always submit another ciphertext, ignoring the coercer. Since the ciphertext is randomized by the CA, the user cannot prove to the coercer that it used the coercer's ciphertext. After the user obtains the signature, it can only be adapted to a ciphertext randomization, so it's not possible to change the encrypted vote. More formally, we define receipt-freeness in a way that user u_i completed the registration with vote M_i of its own choice and can create a fake view of EncAuth concerning a forced vote $\widetilde{M}_i$ that is indistinguishable from the actual view. We recall EncAuth to explain how u_i creates such a fake view.

1. Follow the first step of $\mathsf{EncAuth}_{u_i}$ with $\widetilde{M}_i$ to create $(\widetilde{C}_i,\pi)$.
2. Pick C'_i and evk_i from the real view. Simulate a proof of re-encryption for C'_i and $\widetilde{C}_i$: $\pi' \leftarrow \mathsf{ZKPoK}[C'_i = \mathsf{Rndmz}(\mathsf{ek}, \widetilde{C}_i)]$.
3. Use the real view for MSoRC.ISign.

The simulated view differs from the proper distribution at C'_i and π. Distinguishing C'_i being re-randomization of $\widetilde{C}_i$ or not is infeasible if the DDH assumption holds in $\mathbb{G}_1$. Simulation of π' is due to the quality of the zero-knowledge simulator. Accordingly, the fake view is indistinguishable from the real one if the SXDH assumption holds for BGGen.

Thus, vote selling or buying is of no use. We stress that users can be coerced before the execution of $\mathsf{EncAuth}$. If users are mandated to use a given vote $\widetilde{M}_i$, during $\mathsf{EncAuth}$, users can use their real choice M_i during the signing process. The computationally bounded coercer will have no way to distinguish between these cases. If users are coerced afterward, the unforgeability of MSoRC guarantees that ciphertext-signature pairs can only be adapted to the same plaintext.

Verifiability, fairness, and voter privacy. A voting result is *correct* if it is equal to the outcome obtained by applying the tally computation on the votes M_i of voters who completed the registration and casting phases (Note that, in our scheme, M_i is uniquely determined for each transcript of a completed registration.). A voting scheme is *universally verifiable* when any third party (verifier) accepts the final voting result if and only if it is correct. The "only if" part can be relaxed by incorporating computational assumptions or the trust model. The above verifiability captures the notion of fairness that no votes can be altered once votes are cast.

Our scheme is verifiable since the mixnet is sound, all proofs made by MX are publicly verifiable, and the distributed decryption by TA is also sound and publicly verifiable. Note that the soundness of the mixnet requires the unforgeability of the signatures of CA. Hence, CA is trusted in a way that it would not do anything that risks the unforgeability of MSoRC (*e.g.,* share its secret key).

Verifiability also depends on the fact that every input to the mixnet comes from one voter as the security of the mixnet only concerns one-to-one correspondence between the input ciphertexts and the resulting plaintexts. Verifiablity captures the one-voter-one-vote principle, which must be considered separately. Our design choice is to authenticate users at the registration and casting phases to maintain structural consistency between the voting scheme and the underlying mixnet for easier understanding. We could also choose CA to send the encrypted ballot to BB on behalf of each user at the end of each registration. In this case, $\mathsf{InputVerification}$ can be replaced with the trust of CA. This would not change the trust model since CA is trusted for soundness in our original construction.

We note that voting with authentication inherently reveals who has voted or not. Some consider this as a benefit for democracy, while others view it as a risk to privacy. Practical non-cryptographic countermeasures have been considered, *e.g.,* CA casting null votes for absentees. Another approach would be that if CA sends the ballots to BB on voters' behalf, uvk_i and evk_i in a ballot are replaced with $\mathsf{uvk}_i + \mathsf{evk}_i$. It protects absentees' privacy and provides so-called everlasting privacy [CFSY96, MN07, HMMP23], which claims privacy against unbound adversaries under trust assumptions. A drawback would be that it requires more trust in CA.

Common threats for mix-type voting. A *replay attack* violates a particular voter's privacy by copying a victim's encrypted ballot and seeing if the same vote appears at the end. This is a common risk for mix-type voting with public bulletin boards where ballots are published successively during the casting phase. Many voting schemes have been proven vulnerable to these attacks [MMR22] and possible alternatives to mitigate them should be compatible with receipt-freeness. Our scheme prevents this by letting the voters prove their knowledge of the plaintext, and it accommodates receipt-freeness thanks to the re-encryption.

An *italian attack* [Hea07] can also violate the privacy of a particular voter and it is effective for coercion. In preferential voting, there could be some rarely chosen combination of preferences. The coercer can pick such a rare choice and ask a victim to submit it. As it appears at the end, the coercer can see if the victim obeyed. It is an unavoidable threat against any open-ballot voting with a large space of choices. We refer to [PB09, Yan23] for more discussion.

A.2 Comparison

Table 2. Comparison of trust model and voting properties. V = Voter, CA = Certification Authority, MX = Mix Servers, TA = Tallying Authority, U = Untrusted, T = Trusted, (x, N) = x-out-of-N trust. RF = Receipt Freeness, CR = Coercion Resistance, RA = Replay Attack Resistance, F = Fairness, UV = Universal Verifiability. See text for details on each term.

Scheme	**Privacy**				**Soundness**				**Properties**			
	V	CA	MX	TA	V	CA	MX	TA	RF/CR	RA	F	UV
Rand-RCCA	U	-	$(1, N)$	(k, N)	U	-	U	(k, N)	×	✓	✓	✓
HPP20	**T**	U	$(1, N)$	(k, N)	**T**	**T**	U	(k, N)	×	×	×	×
Ours	U	U	$(1, N)$	(k, N)	U	**T**	U	(k, N)	✓	✓	✓	✓

Table 2 compares the trust model and voting properties of our construction with previously discussed mix-type voting schemes that follow the same blueprint. "Privacy" stands for the infeasibility of associating individual votes and voters when all voters are honest. The "Soundness" columns show which entity must be trusted to guarantee a correct outcome. Namely, if an authority marked as **T** acts in a way that betrays the defined trust, the result of the election can be incorrect, *i.e.,* different from what is directly computed from the plain input, and it is not necessarily noticed by the public. "U" for soundness means that, if the result of the election is obtained, it is correct without assuming any trustful behavior on the respective authority. The "Properties" columns show if the respective property is achieved even if voters and all authorities marked as **U** are corrupted. In Appendix A.3, we extend the comparison to voting schemes that follow a different paradigm.

HPP20 and Rand-RCCA. The model from [FFHR19, FR22, FHR23] does not discuss any authentication mechanism, but we assume users can post signed

ciphertexts to the BB using a previously registered key with the CA (although the corresponding entry in the table is left empty as it's not defined in their work). Since they include proofs of plaintext knowledge, replay attacks can be avoided, but they cannot provide receipt-freeness (nor coercion-resistance). Privacy and verifiability are ensured by their *verify-then-decrypt* protocol. Considering HPP20, as discussed in Appendix B, their model only provides guarantees for honest users, and hence, they cannot achieve any of the properties required for e-voting. That said, recent works by Pointcheval [Poi23, Poi24] have addressed receipt-freeness, also using linearly-homomorphic signatures with randomizable tags as in HPP20. However, [Poi24] adopts the homomorphic encrypted tally paradigm rather than the mixnet-based approach.

A.3 Extended Comparison

Voting schemes are generally required to provide *ballot privacy* (no coalition of malicious parties can learn the voter's vote), *verifiability* (voters can verify that their vote was cast and counted as cast) and *coercion resistance* (a coercer who interacts with a voter during the voting phase cannot determine if coercion was successful or not from the election outcome). Sometimes, a weak form of coercion resistance called *receipt-freeness* [Oka97] is also considered. This notion states that voters cannot prove how they voted to a potential coercer. Additionally, some notion of *fairness* is considered alongside *integrity* to ensure that no partial tally is leaked, and no ballot can be altered during the tally phase. Such guarantees are of utmost importance considering corruption scenarios during the tally phase, which can incorporate information from exit polls to influence the outcome. Similarly to the coercion case, robust notions of verifiability usually cover fairness. Last but not least, security against *replay attacks* protects honest users from malicious ones that try to cast the same vote. Many voting schemes have been proven vulnerable to these attacks [MMR22] and alternatives to mitigate them should be compatible with receipt-freeness.

In this section, we focus on JCJ [JCJ05, CCM08, BGR12, CGY24, ABR23] and VoteAgain [LQT20, HMQA23] that are well-studied mix-type coercion resistant schemes in the literature. Furthermore, VoteAgain also aims for scalability and thus its suitable for comparison with our work.

JCJ & variants – Fake credentials. The voting scheme by Jakobsson, Juels and Catalano (JCJ) [JCJ05] is the standard benchmark for coercion resistance. In this model, users manage real and fake credentials. Whenever they are under the influence of a coercer, users can vote using their fake credentials to convince the coercer that their vote was cast. However, the protocol only counts votes from *real* credentials, whose use is indistinguishable from the fake ones in the coercer's view. Subsequent work identified security and efficiency issues in JCJ, proposing several improvements (see *e.g.,* Civitas/Trivitas [CCM08, BGR12] and CHide [CGY24, ABR23]). Under the JCJ framework, the most efficient protocol under a strong resistance-coercion definition is [ABR23] and has computational complexity $O(n\ log\ n)$ due to sorting. In all cases, users must keep their real

credentials safe and protect them from the coercer. Our work is closer to the JCJ model because we require the absence of a coercer at the beginning.

VoteAgain [LQT20]. Lueks, Querejeta-Azurmendi and Troncoso proposed a voting scheme based on the revoting paradigm, which assumes that the user will be free from the coercer at some point before the voting phase ends. Since each voter can vote multiple times, votes must be filtered so that only the last vote is counted as valid, and coercers cannot identify which votes have been filtered. To achieve better scalability, VoteAgain trades off trust for efficiency. Indeed, its security model makes several trust assumptions: 1) the adversary never gets access to the voter's credentials, 2) the authority is trusted, and 3) a tally server, responsible for filtering the votes is also trusted. Follow-up work [HMQA23] by Haines, Muller and Querejeta-Azurmendi slightly improved trust assumptions but still required all the previous considerations. Besides, the computational complexity is also $O(n\ log\ n)$ due to the insertion of $log\ n$ dummies for every ballot. In this regard, we stress that VoteAgain and JCJ consider different definitions and corruption scenarios for coercion-resistance, which are incomparable in many ways.

Our Work. Ballot privacy, verifiability and fairness follow from the stronger privacy and soundness notions of our mixnet protocol. This contrasts with HPP20, which was unable to provide fairness as evidenced in Appendix B. Receipt-freeness was also already addressed before (recall the randomization on the user's ciphertext done by the CA during the interactive signing). For coercion-resistance the situation is slightly different as our model contrasts with other works in the literature and each of them introduces its tailored definition. However, as previously outlined, unforgeability and perfect adaption of our MSoRC scheme together with receipt-freeness do provide a form of coercion-resistance. Our work achieves all the previously-mentioned properties with $O(n)$ complexity under *minimal* trust assumptions. In particular, we only require an authenticated communication with the BB whereas JCJ and VoteAgain require an *anonymous channel*, which is a much stronger assumption and even harder to achieve in practice. Besides, we also stress that our (public) verifiability also covers the case where malicious mixers omit certain checks such as a correct key randomization deliberately as such behaviours are easily caught when verifying the proofs and mixers' signatures.

B Mixnets from Linearly Homomorphic Signatures

This section presents HPP20's mixnet framework [HPP20], the cornerstone upon which we build upon. Simply put, it is based on the idea that each ciphertext can be handled independently, and servers (mixers) are responsible for randomizing and permuting them. Their shuffle approach comprises four algorithms: MixSetup (global parameters), MixKG (key material for the CA, servers and users), MixInit (run by users to cast their messages) and Mix (run by servers to mix messages), and MixVerify (verifies the outcome).

First, users run $\mathsf{MixInit}$ to send a tuple $\mathcal{T}_i = (C_i, \sigma_i, \mathsf{vk}_i, \Sigma_i)$ where C_i is an ElGamal ciphertext containing the user's plaintext message, σ_i is the user's one-time linearly homomorphic signature for C_i, and Σ_i is the CA's linearly homomorphic signature for vk_i (the public key against σ_i verifies). Notably, this requires a rather complex set up of tags to randomize each signature, and the use of "canonical vectors" to enforce correct randomizations of keys and ciphertexts. This contrasts with our approach that, thanks to the use of MSoRC, removes the need for different signature schemes.

Once all N users in the system have submitted their tuples, the initial shuffle set $\mathcal{SS}\mathsf{et}^{(0)} = (\mathcal{T}_i)_{i=1}^{n}$ is assembled. Subsequently, the Mix process takes place and every server $\mathcal{S}_j$ outputs a new shuffle set $\mathcal{SS}\mathsf{et}^{(j)} = \{(C_{\Pi(i)}, \sigma_{\Pi(i)}, \mathsf{vk}_{\Pi(i)}, \Sigma_{\Pi(i)})_{i\in[n]}^{(j)}, (\pi^{(j)}, \sigma^{(j)})\}$, containing the server's NIZK proof and signature $(\pi^{(j)}, \sigma^{(j)})$ to verify the the correct randomization of each element of $\mathcal{T}_{\Pi(i)}$.

The linear dependence on N for the server's proofs and signatures $(\pi^{(k)}, \sigma^{(k)})_{k=1}^{N}$ can be removed using Groth-Sahai proofs. As explained in HPP20, each server can compute a partial (updatable) proof $\mathsf{proof}^{(j)}$ from $\mathsf{proof}^{(j-1)}$. Servers verify the individual proofs and the final proof $\mathsf{proof}^{(N)}$ to then sign $\mathsf{proof}^{(N)}$ using the multi-signature scheme from Boneh-Drijvers-Neven [BDN18]. As a result, only the initial and last shuffle sets ($\mathcal{SS}\mathsf{et}^{(0)}$ and $\mathcal{SS}\mathsf{et}^{(N)}$) and a single proof-signature pair are required to run Verify.

Security Model. HPP20 requires soundness and privacy for *honest users*. Informally, soundness means that all plaintexts of *honest users* in the input shuffle set are in the output shuffle set. Likewise, privacy means that messages of *honest users* are unlinkable from the input shuffle set to the output shuffle set. For soundness, only the initial input shuffle set and output shuffle set are considered.

Definition 12 (Soundness for Honest Users [HPP20]). *A mixnet is said to be 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 certification and encryption keys.*
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 secret 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 message*
- *generates the message tuples* $(\mathcal{T}_i)_{i\in\mathcal{I}^*}$ *for corrupted users but provides the messages* $(M_i)_{i\in\mathcal{I}\setminus\mathcal{I}^*}$ *for the honest ones*
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 tuples* $(\mathcal{T}_i)_{i\in\mathcal{I}\setminus\mathcal{I}^*}$*. The initial shuffle set is thus defined by* $\mathcal{SS}\mathsf{et} = (\mathcal{T}_i)_{i\in\mathcal{I}}$*.*
4. *The adversary mixes* $\mathcal{SS}\mathsf{et}$ *in a provable way into* $(\mathcal{SS}\mathsf{et}', \mathsf{proof}')$*.*

The adversary wins if $\mathsf{Verify}(\mathcal{SS}\mathsf{et}, \mathcal{SS}\mathsf{et}', \mathsf{proof}') = 1$ *but* $\{\mathsf{Dec}^*(\mathcal{SS}\mathsf{et})\} \neq \{\mathsf{Dec}^*(\mathcal{SS}\mathsf{et}')\}$*, where* Dec^* *extracts the plaintexts using the*

decryption key, but ignores messages of non-honest users (using the private keys of honest users) and sets of plaintexts can have repetitions.

The privacy games allows the adversary to provide two possible permutations for honest mix servers so that the challenger uses one of them. The adversary's goal is to identify which was the permutation used, capturing the *unlinkability* notion behind the privacy definition.

Definition 13 (Privacy for Honest Users [HPP20]). *A mixnet 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 certification and encryption keys.*
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 secret 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*
– *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 message*
– *generates the message tuples* $(\mathcal{T}_i)_{i\in\mathcal{I}^*}$ *for corrupted users but provides the messages* $(M_i)_{i\in\mathcal{I}\setminus\mathcal{I}^*}$ *for the honest ones*

3. *The challenger generates the keys of the honest mix-servers* $j \in \mathcal{J} \setminus \mathcal{J}^*$ *and the keys of the honest users* $(\mathsf{sk}_i, \mathsf{vk}_i)_{i\in\mathcal{I}\setminus\mathcal{I}^*}$ *and their message tuples* $(\mathcal{T}_i)_{i\in\mathcal{I}^*}$.

The initial shuffle set is thus defined by $\mathcal{SS}\mathsf{et} = (\mathcal{T}_i)_{i\in\mathcal{I}}$. *The challenger randomly chooses a bit* $b \leftarrow\!\!\$\ \{0,1\}$ *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 tuples of the corrupted users in the input shuffle set* $\mathcal{SS}\mathsf{et}^{(j-1)}$
– *if* $j \in \mathcal{J}^*$, $\mathcal{A}$ *builds itself the new shuffle set* $\mathcal{SS}\mathsf{et}^{(j)}$ *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* $(\mathcal{SS}\mathsf{et}^{(j)}, \mathsf{proof}^{(j)})$

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

Security against malicious users. Security for *honest users* is not sufficient for voting applications. To see why, we consider the following example that is possible in their model. Assume the adversary controls four out of ten voters in an election of three candidates (C1, C2 and C3). Let us also assume that the six votes from honest users are distributed so that C1 gets four, C2 gets one and so does C3. Initially, the adversary mandates the coerced users to vote such that two votes are given to C1, one to C2 and one to C3. Once that all votes are casted an exit poll reveals that C1 is the favourite. Knowing this, the adversary colludes

with the first mix server to change the votes of coerced users such that only the vote for C3 is counted (the others are replaced by randomizations of that vote). None of the votes from honest users is discarded nor modified yet the election outcome changes. While such an action is not a flaw in the security model, it is clearly a violation of voting schemes known as *fairness*. The essential problem is that the universal verifiability is lost under the collusion of the first mix server and some users. The authors consider a partial fix to this issue, adding another Groth-Sahai proof as discussed in Sec. 6.1 from HPP20. However, such fix still allows *replay attacks* [CS11] that should also be avoided in voting applications.

C Couteau & Hartmann's Proof System

Below we give the NIZK proof system for $\mathcal{L}_{\mathbf{A}}$ in the framework of CH20 (Sec. 7.1). Security has been proven under the kerMDH assumption [MRV16] in [CH20].

- $\mathsf{NIZK.Setup}(1^\kappa)$: $\mathsf{pp} \leftarrow\!\$\ \mathsf{BGGen}(1^\kappa)$; $z \leftarrow\!\$\ \mathbb{Z}_p; \tau \leftarrow z$;
  $Z \leftarrow zG; \mathsf{crs} \leftarrow (\mathsf{pp}, Z)$; **return** $((\mathsf{pp}, \mathsf{crs}), \tau)$
- $\mathsf{NIZK.Prove}(\mathsf{crs}, \mathbf{A}, \mathsf{x}, w)$: $r \leftarrow\!\$\ \mathbb{Z}_p$;
  $\mathbf{a} \leftarrow r\mathbf{A}$; $\mathsf{d} \leftarrow wZ + rG$; $\pi \leftarrow (\mathbf{a}, \mathsf{d})$; **return** π
- $\mathsf{NIZK.Verify}(\mathsf{crs}, \mathbf{A}, \mathbf{x}, (\mathbf{a}, \mathsf{d}))$:
  return $e(\mathsf{d}, \mathsf{A}_0) = e(Z, \mathsf{x}_0) + e(G, \mathsf{a}_0) \wedge e(\mathsf{d}, \mathsf{A}_1)$
  $= e(Z, \mathsf{x}_1) + e(G, \mathsf{a}_1) \wedge\ e(\mathsf{d}, \mathsf{A}_2) = e(Z, \mathsf{x}_2) + e(G, \mathsf{a}_2)$

C.1 Batch Verification

The proof system from [CH20] is compatible with the batch verification technique from [FGHP09] that ports the small exponents test [BGR98] to the pairing setting. Given two valid proofs (a, d) and $(\mathsf{a}', \mathsf{d}')$ for A and A' respectively, a naive verification would have to check six pairing equations:

$$\begin{aligned} &e(\mathsf{d}, \mathsf{A}_0) = e(Z, \mathsf{x}_0) + e(G, \mathsf{a}_0) \wedge e(\mathsf{d}, \mathsf{A}_1) = e(Z, \mathsf{x}_1) + e(G, \mathsf{a}_1) \\ &\wedge\ e(\mathsf{d}, \mathsf{A}_2) = e(Z, \mathsf{x}_2) + e(G, \mathsf{a}_2) \wedge e(\mathsf{d}', \mathsf{A}'_0) = e(Z, \mathsf{x}'_0) + e(G, \mathsf{a}'_0) \\ &\wedge\ e(\mathsf{d}', \mathsf{A}'_1) = e(Z, \mathsf{x}'_1) + e(G, \mathsf{a}'_1) \wedge\ e(\mathsf{d}', \mathsf{A}'_2) = e(Z, \mathsf{x}'_2) + e(G, \mathsf{a}'_2) \end{aligned}$$

With [FGHP09], a verifier can instead sample $(\delta_i)_{i\in[6]}$ where δ_i is an ℓ-bit element of $\mathbb{Z}_p$ and check a single equation given by:

$$\begin{aligned} &e(\mathsf{d}, \mathsf{A}_0^{\delta_1}\mathsf{A}_1^{\delta_2}\mathsf{A}_2^{\delta_3}) + e(\mathsf{d}', \mathsf{A}_0'^{\delta_4}\mathsf{A}_1'^{\delta_5}\mathsf{A}_2'^{\delta_6}) \\ = \\ &e(Z, \mathsf{x}_0^{\delta_1}\mathsf{x}_1^{\delta_2}\mathsf{x}_2^{\delta_3}\mathsf{x}_0'^{\delta_4}\mathsf{x}_1'^{\delta_5}\mathsf{x}_2'^{\delta_6}) + e(G, \mathsf{a}_0^{\delta_1}\mathsf{a}_1^{\delta_2}\mathsf{a}_2^{\delta_3}\mathsf{a}_0'^{\delta_4}\mathsf{a}_1'^{\delta_5}\mathsf{a}_2'^{\delta_6}) \end{aligned}$$

There is an efficiency trade-off: the larger ℓ is (in general $\ell = 80$), the better are the soundness guarantees.

C.2 Updatability

Updatable NIZK proofs are non-interactive (one move) *malleable* zero-knowledge proofs [BCC+09, CKLM12] that allow to publicly update a valid proof to a different witness. Updated proofs should be indistinguishable from fresh proofs that are computed on the updated witness from scratch. This property is known as derivation privacy. In the following, we recall the syntax and main security properties of this primitive, and refer the reader to [DEF+25] for a more detailed exposition.

Syntax. An updatable NIZK proof system for a language $\mathcal{L}$ and a set of transformations $\mathcal{T}$ is defined by the following algorithms of which all except Setup are implicitly parametrized by crs.

$\mathsf{Setup}(1^\kappa) \to (\mathsf{crs}, \tau)$: Generates a common reference string crs and a trapdoor τ, which is used for security definitions.
$\mathsf{Prove}(\mathsf{x}, \mathsf{w}) \to \pi$: Produces a proof for $(\mathsf{x}, \mathsf{w}) \in \mathcal{R}_\mathcal{L}$.
$\mathsf{Verify}(\pi, \mathsf{x}) \to \{0,1\}$: Verifies π w.r.t. the public instance x.
$\mathsf{Update}(\pi, \mathsf{x}, T) \to \pi'$: Updates the proof π for x into π' for $T(\mathsf{x})$.

Definition 14 (Update Completeness [DEF+25]). *An updatable* NIZK *proof system for $\mathcal{L}$ satisfies update completeness w.r.t. a set of transformations $\mathcal{T}$, if given $(\mathsf{crs}, \cdot) \leftarrow\!\$\ \mathsf{Setup}(1^\kappa)$, for all x, π such that $\mathsf{Verify}(\pi, \mathsf{x}) = 1$, and all $T = (T_\mathsf{x}, \cdot) \in \mathcal{T}$ it holds that:*

$$\Pr[\mathsf{Verify}(\mathsf{crs}, \mathsf{Update}(\mathsf{crs}, \pi, \mathsf{x}, T), T_\mathsf{x}(\mathsf{x})) = 1] = 1$$

Definition 15 (Derivation Privacy [DEF+25]). *An updatable* NIZK *proof system for $\mathcal{L}$ satisfies derivation privacy w.r.t. a set of transformations $\mathcal{T}$, if given $(\mathsf{crs}, \cdot) \leftarrow\!\$\ \mathsf{Setup}(1^\kappa)$, for all $(\mathsf{x}, \mathsf{w}) \in \mathcal{R}_\mathcal{L}$, all π such that $\mathsf{Verify}(\pi, \mathsf{x}) = 1$, and all $T = (T_\mathsf{x}, T_\mathsf{w}) \in \mathcal{T}$ it holds that: $\{\mathsf{Update}(\pi, \mathsf{x}, T)\} \stackrel{c}{=} \{\mathsf{Prove}(T_\mathsf{x}(\mathsf{x}), T_\mathsf{w}(\mathsf{w}))\}$.*

Malleabiltiy of CH20 was used in [CLPK22] to build EQS and very recently further formalized by David *et al.* [DEF+25] to build updatable privacy-preserving blueprints [KLN23]. In brief, for linear languages like the on used in this work $\mathcal{L}_\mathbf{A}$, we can update a witness w into ww' and obtain a proof for it by simply multiplying the proof $\pi = (\mathbf{a}, \mathsf{d})$ for w by w'.

D Proof of Theorem 3

Proof. We consider an adversary $\mathcal{A}$ similar to that one against the unforgeability game from Def. 3. The difference is that we let the challenger generate the encryption keys and give the adversary access to ek only. To prove unforgeability we follow a similar strategy (in parts verbatim) to that of [BF20]. The main difference is that now, the *generic* adversary no longer controls the secret key $\mathsf{dk} = x$. Consequently, group elements output by the adversary can be a linear combination of previously seen elements, which includes the representation of x in the GGM. To prove that our modified scheme is also unforgeable w.r.t. the

interactive signing protocol (Def. 8), we need to modify the simulator from Fig. 8 to drop S and simulate it in the first ZKPoK, which can easily be done under DDH.

We begin observing that the challenger picks $(\mathsf{sk}, \mathsf{vk}) = ((x_0, x_1, x_2), (\hat{X}_0^* = x_0\hat{G}, \hat{X}_1^* = x_1\hat{G}, \hat{X}_2^* = x_2\hat{G}))$, $(\mathsf{dk}, \mathsf{ek}) = (x, X = xG)$, and randomness s_i for each of the adversary's signing queries.

After seeing vk and signatures $(Z_i, \hat{S}_i, T_i)_{i=1}^k$ (computed with randomness s_i) on queries $(C_0^{(i)}, C_1^{(i)})_{i=1}^k$, $\mathcal{A}$ outputs $(C_0^{(k+1)}, C_1^{(k+1)})$, a signature $(Z^*, \hat{S}^*, T^*)$ and verification key $\mathsf{vk}^* = (\hat{X}_0^*, \hat{X}_1^*, \hat{X}_2^*)$. Since $\mathcal{A}$ is a generic forger, all computed elements must be a linear combination of previously seen elements. Consequently, the following equations should hold for a suitable set of coefficients chosen by $\mathcal{A}$:

$$C_0^{(i)} = \gamma^{(i)}G + \gamma_x^{(i)}X + \sum_{j=1}^{i-1}(\gamma_{z,j}^{(i)}Z_j + \gamma_{t,j}^{(i)}T_j)$$

$$C_1^{(i)} = \kappa^{(i)}G + \kappa_x^{(i)}X + \sum_{j=1}^{i-1}(\kappa_{z,j}^{(i)}Z_j + \kappa_{t,j}^{(i)}T_j)$$

$$Z^* = \zeta G + \zeta_x^{(i)}X + \sum_{j=1}^{k}(\zeta_{z,j}Z_j + \zeta_{t,j}T_j)$$

$$\hat{S}^* = \phi\hat{G} + \phi_0\hat{X}_0 + \phi_1\hat{X}_1 + \phi_2\hat{X}_2 + \sum_{j=1}^{k}\phi_{s,j}\hat{S}_j$$

$$T^* = \tau G + \tau_x^{(i)}X + \sum_{j=1}^{k}(\tau_{z,j}Z_j + \tau_{t,j}T_j)$$

$$\hat{X}_0^* = \chi^0\hat{G} + \chi_0^0\hat{X}_0 + \chi_1^0\hat{X}_1 + \chi_2^0\hat{X}_2 + \sum_{j=1}^{k}\chi_{s,j}^0\hat{S}_j$$

$$\hat{X}_1^* = \chi^1\hat{G} + \chi_0^1\hat{X}_0 + \chi_1^1\hat{X}_1 + \chi_2^0\hat{X}_2 + \sum_{j=1}^{k}\chi_{s,j}^1\hat{S}_j$$

$$\hat{X}_2^* = \chi^2\hat{G} + \chi_0^2\hat{X}_0 + \chi_1^2\hat{X}_1 + \chi_2^2\hat{X}_2 + \sum_{j=1}^{k}\chi_{s,j}^2\hat{S}_j$$

Moreover, for all $1 \leq i \leq k$, we can write the discrete logarithms z_i and t_i in basis G of the elements $Z_i = \frac{1}{s_i}(x_0C_0^{(i)} + x_1C_1^{(i)} + x_2G)$ and $T_i = \frac{1}{s_i}(x_0G + x_1X)$ from the oracle answers. We have:

$$z_i = \frac{1}{s_i}(x_0(\gamma^{(i)} + \gamma_x^{(i)}x + \sum_{j=1}^{i-1}(\gamma_{z,j}^{(i)}z_j + \gamma_{t,j}^{(i)}t_j))$$
$$+ x_1(\kappa^{(i)} + \kappa_x^{(i)}x + \sum_{j=1}^{i-1}(\kappa_{z,j}^{(i)}z_j + \kappa_{t,j}^{(i)}t_j)) + x_2)$$
$$t_i = \frac{1}{s_i}(x_0 + x_1 x)$$

A successful forgery $(Z^*, \hat{S}^*, T^*)$ on $(C_0^{(k+1)}, C_1^{(k+1)})$ satisfies the verification equations, and we can take the discrete logarithms in base $e(G, \hat{G})$ for each equation as shown below:

$$(\zeta + \zeta_x x + \sum_{j=1}^{k}(\zeta_{z,j}z_j + \zeta_{t,j}t_j))(\phi + \phi_0 x_0 + \phi_1 x_1$$
$$+\phi_2 x_2 + \sum_{j=1}^{k}\phi_{s,j}s_j) = \alpha x_0 c_0^{(k+1)} + \alpha x_1 c_1^{(k+1)} + \alpha x_2 \quad (1)$$

$$(\tau + \tau_x x + \sum_{j=1}^{k}(\tau_{z,j}z_j + \tau_{t,j}t_j))(\phi + \phi_0 x_0 + \phi_1 x_1$$
$$+\phi_2 x_2 + \sum_{j=1}^{k}\phi_{s,j}s_j) = \alpha x_0 + \alpha x_1 x \quad (2)$$

Equations (1) and (2) are valid with respect to the forged key $(\hat{X}_0^*, \hat{X}_1^*, \hat{X}_2^*)$. However, since verification pass, we have that $[\hat{X}_i^*]_{\mathsf{pk}} = [\hat{X}_i]_{\mathsf{pk}}$ and thus $\exists\ \alpha \in \mathbb{Z}_p^*$ s.t. $\hat{X}_i^* = \alpha \hat{X}_i, i \in \{0,1,2\}$.[3] Furthermore, we can interpret the previous verification equations as multivariate rational functions in variables $x_0, x_1, x_2, x, s_1, \ldots, s_k$, unknown to $\mathcal{A}$. We begin analyzing if α can be zero modulo any x_i, as this will prove useful later. We can take the discrete logarithms in base $\hat{G}$ for each equation defining $\hat{X}_i^*$ to obtain:

$$\alpha x_0 = \chi^0 + \chi_0^0 x_0 + \chi_1^0 x_1 + \chi_2^0 x_2 + \sum_{j=1}^{k}\chi_{s,j}^0 s_j$$
$$\alpha x_1 = \chi^1 + \chi_0^1 x_0 + \chi_1^1 x_1 + \chi_2^0 x_2 + \sum_{j=1}^{k}\chi_{s,j}^1 s_j$$
$$\alpha x_2 = \chi^2 + \chi_0^2 x_0 + \chi_1^2 x_1 + \chi_2^2 x_2 + \sum_{j=1}^{k}\chi_{s,j}^2 s_j$$

[3] Such relation is efficiently checkable by the challenger (it knowns sk).

From the above, it follows that for α to be zero modulo any x_i, all the of coefficients must be zero, which is a contradiction.

In the following, we assume without loss of generality that $(\phi+\phi_0 x_0+\phi_1 x_1+\phi_2 x_2+\sum_{j=1}^{k}\phi_{s,j}s_j) \neq 0$ because $\hat{S}^* \neq 0$.

As in [BF20], we now interpret the equalities over the ring $\mathbb{Z}_p(s_1,\ldots,s_k)[x_0,x_1,x_2,x]$ as well as over $\mathbb{Z}_p(s_1,\ldots,s_k)[x_0,x_1,x_2,x]/(x_0,x_1,x_2,x) \equiv \mathbb{Z}_p(s_1,\ldots,s_k)$.[4] Over such quotient $z_i = 0$ and $t_i = 0$, and thus, (1) and (2) become:

$$\zeta(\phi+\sum_{j=1}^{k}\phi_{s,j}s_j) = 0 \tag{3}$$

$$\tau(\phi+\sum_{j=1}^{k}\phi_{s,j}s_j) = 0 \tag{4}$$

Case 1: If $(\phi+\sum_{j=1}^{k}\phi_{s,j}s_j) = 0$ then $\phi = \phi_{s,j} = 0$. However, this would imply that S^* is a linear combination of the public key. But this can only hold if it's the trivial one, leading to a contradiction.

Case 2: $(\phi+\sum_{j=1}^{k}\phi_{s,j}s_j) \neq 0$. We have $\forall i \in \{1,\ldots,k\} : \tau = \zeta = 0$. Hence, (1) and (2) turn into:

$$\begin{aligned}(\zeta_x x+\sum_{j=1}^{k}(\zeta_{z,j}z_j+\zeta_{t,j}t_j))(\phi+\phi_0 x_0+\phi_1 x_1\\ +\phi_2 x_2+\sum_{j=1}^{k}\phi_{s,j}s_j) = \alpha x_0 c_0^{(k+1)}+\alpha x_1 c_1^{(k+1)}+\alpha x_2\end{aligned} \tag{5}$$

$$\begin{aligned}(\tau_x x+\sum_{j=1}^{k}(\tau_{z,j}z_j+\tau_{t,j}t_j))(\phi+\phi_0 x_0+\phi_1 x_1\\ +\phi_2 x_2+\sum_{j=1}^{k}\phi_{s,j}s_j) = \alpha x_0+\alpha x_1 x\end{aligned} \tag{6}$$

Computing the above modulo (x_0,x_1,x_2) we get $\zeta_x = \tau_x = 0$. Putting back x_2 and looking modulo (x_0,x_1), we get:

$$(\sum_{j=1}^{k}\zeta_{z,j}\frac{1}{s_j})(\phi+\phi_2 x_2+\sum_{j=1}^{k}\phi_{s,j}s_j) = \alpha \tag{7}$$

[4] This interpretation is possible because x_0, x_1 and x_2 never appear in the denominators of any expression.

$$(\sum_{j=1}^{k} \tau_{z,j} \frac{x_2}{s_j})(\phi + \phi_2 x_2 + \sum_{j=1}^{k} \phi_{s,j} s_j) = 0 \tag{8}$$

We deduce $\tau_{z,j} = 0 \ \forall j \in \{1, \ldots, k\}$. Now, equation (6) modulo (x, x_1) becomes:

$$(\sum_{j=1}^{k} \tau_{t,j} \frac{1}{s_j})(\phi + \phi_0 x_0 + \sum_{j=1}^{k} \phi_{s,j} s_j) = \alpha \tag{9}$$

We first observe that there exists j_0 such that $\tau_{t,j_0} \neq 0$ as otherwise T^* would be zero and thus a contradiction. Then, looking at the degrees in s_{j_0}, the left hand size of the equation has $\deg_{s_{j_0}} = -1$, which means that $(\phi + \phi_0 x_0 + \sum_{j=1}^{k} \phi_{s,j} s_j)$ should have degree one in s_{j_0}. Hence, there is also at least one $\phi_{s,j_0} \neq 0$. Suppose there exist $j_1 \neq j_2 \in \{1, \ldots, k\}$ such that $\phi_{s,j_1} \neq 0$ and $\phi_{s,j_2} \neq 0$. As in [BF20], that leads to a contradiction. So there is only one non-zero coefficient. Similarly, we conclude $\forall i \in \{1, \ldots, k\} \setminus \{j_0\} : \zeta_{z,j} = \tau_{t,j} = 0$.

Now, equations (5) and (6) become:

$$\begin{aligned}(\zeta_{z,j_0} z_{j_0} + \sum_{j=1}^{k} (\zeta_{t,j} \frac{x_0 + x_1 x}{s_j}))(\phi + \phi_0 x_0 + \phi_1 x_1 \\ + \phi_2 x_2 + \phi_{s,j_0} s_{j_0}) = \alpha x_0 c_0^{(k+1)} + \alpha x_1 c_1^{(k+1)} + \alpha x_2\end{aligned} \tag{10}$$

$$(\tau_{t,j_0} \frac{x_0 + x_1 x}{s_{j_0}})(\phi + \phi_0 x_0 + \phi_1 x_1 + \phi_2 x_2 + \phi_{s,j_0} s_{j_0}) = \alpha x_0 + \alpha x_1 x \tag{11}$$

equating coefficients for x_0 we get $\tau_{t,j_0} \phi_{s,j_0} = \alpha$, which means that $\tau_{t,j_0} \neq 0$. Moreover, we deduce, $\phi = \phi_0 = \phi_1 = \phi_2 = 0$. Besides, $\zeta_{z,j_0} \phi_{s,j_0} = \alpha$ (taking modulo x_0, x_1). This means that $\zeta_{z,j_0} = \tau_{t,j_0}$. Now, we have:

$$\begin{aligned}x_0 c_0^{(j_0)} \zeta_{z,j_0} \phi_{s,j_0} + x_1 c_1^{(j_0)} \zeta_{z,j_0} \phi_{s,j_0} \\ + \phi_{s,j_0} s_{j_0} \sum_{j=1}^{k} (\zeta_{t,j} \frac{x_0 + x_1 x}{s_j}) = \alpha x_0 c_0^{(k+1)} + \alpha x_1 c_1^{(k+1)}\end{aligned} \tag{12}$$

Equating coefficients for x_0 and x_1 we get that $c_0^{(j_0)} = c_0^{(k+1)}$ and $c_1^{(j_0)} = c_1^{(k+1)}$, meaning that it's a ciphertext that has already been queried.

The above means that the adversary cannot win the unforgeability game in the ideal world (because the first winning condition cannot be met if the other two hold). It remains to see that the statistical distance from the adversary's point of view when interacting in the real game (for concrete choices of $x_0, x_1, x_2, x, s_1, \ldots, s_k$) with the ideal one is negligible. This follows from the analysis in [BF20], which applies the Schwartz-Zippel lemma [Sch80].

E Zero-knowledge Proofs

We instantiate the ZKPoK of our interactive signing protocol in the ROM using known techniques [FS87, Sch91, CP93]. $\pi_0 := (\mathsf{ZKPoK}[s_0 : S_0 = s_0 G \ \wedge \ \hat{S}_0 = s_0 \hat{G}])$ is shown in Fig.13. $\pi_1 := (\mathsf{ZKPoK}[(r, x_0^1, x_1^1, x_2^1) : T_1 = rS_0 + x_0^1 G + x_1^1 X \wedge Z_1 = rS_0 + x_0^1 C_0 + x_1^1 C_1 + x_2^1 G \wedge \hat{X}_0^1 = x_0^1 \hat{G} \wedge X_1^1 = x_1^1 \hat{G} \wedge X_2^1 = x_2^1 \hat{G}])$ is shown in Fig.15. They are a simple application of standard ZKPoK. However, $\widetilde{\pi}_0 := (\mathsf{ZKPoK}[(s_0, x_0^0, x_1^0, x_2^0) : T_0 = \frac{1}{s_0}(T_1 + x_0^0 G + x_1^0 X) \wedge Z_0 = \frac{1}{s_0}(Z_1 + x_0^0 C_0 + x_1^0 C_1 + x_2^0 G) \wedge S_0 = s_0 G \wedge \hat{X}_0^0 = x_0^0 \hat{G} \wedge \hat{X}_1^0 = x_1^0 \hat{G} \wedge \hat{X}_2^0 = x_2^0 \hat{G}])$ and $\widetilde{\pi}_1 := (\mathsf{ZKPoK}[(r, s_1) : T = \frac{1}{s_1}(T_0 - rG) \wedge Z = \frac{1}{s_1}(Z_0 - rG) \wedge \hat{S} = s_1 \hat{S}_0])$ include multiplication of witness variables in some clauses. Hence, we need to re-arrange the statements. We change $\widetilde{\pi}_0$ into

$$\begin{aligned}\mathsf{ZKPoK}[(s_0, x_0^0, x_1^0, x_2^0) : \ & T_1 = s_0 T_0 - x_0^0 G - x_1^0 X \wedge \\ & Z_1 = s_0 Z_0 - x_0^0 C_0 - x_1^0 C_1 - x_2^0 G \wedge \\ & S_0 = s_0 G \ \wedge \hat{X}_0^0 = x_0^0 \hat{G} \wedge \hat{X}_1^0 = x_1^0 \hat{G} \wedge \hat{X}_2^0 = x_2^0 \hat{G}]\end{aligned}$$

and turn $\widetilde{\pi}_1$ into $\mathsf{ZKPoK}[(r, s_1) : T_0 = rG + s_1 T \wedge Z_0 = rG + s_1 Z \wedge \hat{S} = s_1 \hat{S}_0]$. These statements are equivalent to the original ones that were shown in Fig. 16 and Fig. 14.

Prover: $S_0, \hat{S}_0, s_0$		Verifier: $S_0, \hat{S}_0$
$a_1 \leftarrow\$ \mathbb{Z}_p$		
$A_1 = a_1 G; \hat{A}_1 = a_1 \hat{G}$	$\xrightarrow{A_1, \hat{A}_1}$	
	$\xleftarrow{c}$	$c \leftarrow\$ \mathbb{Z}_p$
$q_1 = a_1 - c s_0$	$\xrightarrow{q_1}$	**return** $A_1 = q_1 G + c S_0$ $\wedge \ \hat{A}_1 = q_1 \hat{G} + c \hat{S}_0$

Fig. 13. ZKPoK protocol for π_0.

Prover: $Z, T, \hat{S}, Z_0, T_0, \hat{S}_0, r, s_1$		Verifier: $Z, T, \hat{S}, Z_0, T_0, \hat{S}_0$
$a_1, a_2 \leftarrow\$ (\mathbb{Z}_p)^2; A_1 = a_1 G + a_2 T$		
$A_2 = a_1 G + a_2 Z; \hat{A}_4 = a_2 \hat{S}_0$	$\xrightarrow{A_1, A_2, \hat{A}_4}$	
	$\xleftarrow{c}$	$c \leftarrow\$ \mathbb{Z}_p$
$q_1 = a_1 - cr; q_2 = a_2 - c s_1$	$\xrightarrow{q_1, q_2}$	**return** $A_1 = q_1 G + q_2 T + c T_0$ $\wedge \ A_2 = q_1 G + q_2 Z + c Z_0 \wedge \hat{A}_4 = q_2 \hat{S}_0 + c \hat{S}$

Fig. 14. ZKPoK protocol for $\widetilde{\pi}_1$.

F Aggregate and Multi-signatures

We recall the sequential aggregate signature from [PS16].

Prover: $T_1, Z_1, \{\hat{X}_i^1, x_i^1\}_{i\in\{0..2\}}, S_0, X, C_0, C_1, r$		Verifier: $T_1, Z_1, \{\hat{X}_i^1\}_{i\in\{0..2\}}, S_0, X, C_0, C_1$
$a_1, a_2, a_3, a_4 \leftarrow\$ (\mathbb{Z}_p)^4$ $A_1 = a_1 S_0 + a_3 G + a_4 X$ $A_2 = a_1 S_0 + a_3 C_0 + a_4 C_1 + a_2 G$		
$\hat{A}_3 = a_3\hat{G}; \hat{A}_4 = a_4\hat{G}; \hat{A}_5 = a_2\hat{G}$	$\xrightarrow{A_1, A_2, \hat{A}_3, \hat{A}_4, \hat{A}_5}$	
$q_1 = a_1 - cr; q_2 = a_2 - cx_2^1$	$\xleftarrow{c}$	$c \leftarrow\$ \mathbb{Z}_p$
$q_3 = a_3 - cx_0^1; q_4 = a_4 - cx_1^1$	$\xrightarrow{q_1, q_2, q_3, q_4}$	**return** $A_1 = q_1 S_0 + q_3 G + q_4 X + cT_1$ $\wedge\ A_2 = q_1 S_0 + q_3 C_0 + q_4 C_1 + q_2 G + cZ_1$ $\wedge\ \hat{A}_3 = q_3\hat{G} + c\hat{X}_0^1 \wedge \hat{A}_4 = q_4\hat{G} + c\hat{X}_1^1$ $\wedge \hat{A}_5 = q_2\hat{G} + c\hat{X}_2^1$

Fig. 15. ZKPoK protocol for π_1.

Prover: $\{T_i, Z_i\}_{i\in\{0,1\}}, S_0, s_0, \{\hat{X}_i^0, x_i^0\}_{i\in\{0..2\}}, X, C_0, C_1$		Verifier: $\{T_i, Z_i\}_{i\in\{0,1\}}, S_0, \{\hat{X}_i^0\}_{i\in\{0..2\}}, X, C_0, C_1$
$a_1, a_2, a_3, a_4, a_6 \leftarrow\$ (\mathbb{Z}_p)^5$ $A_1 = a_1 T_0 - a_2 G - a_3 X; A_2 = a_1 T - a_4 G$ $A_2 = a - 1 Z_0 - a_2 C_0 - a_3 C_1 - a_4 G$		
$A_3 = a_1 G; \hat{A}_4 = a_2\hat{G}; \hat{A}_5 = a_3\hat{G}; \hat{A}_6 = a_6\hat{G}$	$\xrightarrow{A_1 \ldots \hat{A}_6}$	
$q_1 = a_1 - cs_0; q_2 = a_2 - cx_0^0$	$\xleftarrow{c}$	$c \leftarrow\$ \mathbb{Z}_p$
$q_3 = a_3 - cx_1^0; q_4 = a_4 - cx_2^0$	$\xrightarrow{q_1, q_2, q_3, q_4}$	**return** $A_1 = q_1 T_0 - q_2 G - q_3 X + cT_1$ $\wedge\ A_2 = q_1 Z_0 - q_2 C_0 - q_3 C_1 - q_4 G + cZ_1$ $\wedge\ A_3 = q_1 G + cS_0 \wedge \hat{A}_4 = q_2\hat{G} + c\hat{X}_0^0$ $\wedge\ \hat{A}_5 = q_3\hat{G} + c\hat{X}_1^0 \wedge \hat{A}_6 = q_5\hat{G} + c\hat{X}_2^0$

Fig. 16. ZKPoK protocol for $\widetilde{\pi}_0$.

$\underline{\mathsf{SAS.Setup}(1^\kappa)}$: $\mathsf{pp} \leftarrow\$ \mathsf{BGGen}(1^\kappa)$; $w \leftarrow\$ \mathbb{Z}_p$;
$W \leftarrow wG$; $\hat{W} \leftarrow w\hat{G}$; **return** $(\mathsf{pp}, W, \hat{W})$.
$\underline{\mathsf{SAS.SKG}(\mathsf{pp})}$: $\mathsf{sk} \leftarrow\$ \mathbb{Z}_p^*$; $\mathsf{pk} \leftarrow \mathsf{sk}\hat{G}$; **return** $(\mathsf{sk}, \mathsf{pk})$.
$\underline{\mathsf{SAS.Sign}(\mathsf{sk}, \sigma, (m_1, \ldots, m_r), (\mathsf{pk}_1, \ldots, \mathsf{pk}_r), m)}$:
if $r = 0$ **then** $\sigma \leftarrow (G, W)$ **elseif** $(r > 0$
$\wedge\ \mathsf{SAS.Verify}(\sigma, (m_1, \ldots, m_r), (\mathsf{pk}_1, \ldots, \mathsf{pk}_r)) = 0) \vee m = 0 \vee \exists\ \mathsf{pk}_j \in \{\mathsf{pk}_1, \ldots, \mathsf{pk}_r\}$:
$\mathsf{pk}_j = \mathsf{pk}$ **return** $\perp$.
else $t \leftarrow\$ \mathbb{Z}_p^*; \sigma' \leftarrow (t\sigma_1, t(\sigma_2 + (\mathsf{sk} \cdot m)\sigma_1))$ **return** σ'.
$\underline{\mathsf{SAS.Verify}(\sigma, (m_1, \ldots, m_r), (\mathsf{pk}_1, \ldots, \mathsf{pk}_r))}$:
return $\sigma_1 \neq 1_{\mathbb{G}} \wedge e(\sigma_1, \hat{W} + \sum_i m_i \mathsf{pk}_i) = e(\sigma_2, \hat{G})$.

Its security considers the certified keys setting from [LOS+06] (*i.e.,* users must prove knowledge of their secret key if they want to produce a signature) and is proven in the generic group model for type-III pairings, under the Pointcheval-Sanders assumption given in Definition 16. Alternatively, as shown by the same authors [PS18], it's also possible to prove security under a non-interactive assumption (the q-MSDH-1 assumption, which is itself a variant of the q-SDH assumption) in the random oracle model with a small modification to the scheme that doesn't incur any efficiency overhead.

Definition 16 (PS Assumption). *Let* BGGen *be a type-III bilinear group generator and* $\mathcal{A}$ *a* PPT *algorithm. The Pointcheval-Sanders (PS) assumption over* BGGen *states that the following probability is negligible in* κ*:*

$$Pr\left[\begin{array}{l} Q := \emptyset; \mathsf{pp} \leftarrow\!\$\ \mathsf{BGGen}(1^\kappa) \\ x, y \leftarrow\!\$\ \mathbb{Z}_p^*; \hat{X} \leftarrow x\hat{G}; \hat{Y} \leftarrow y\hat{G} \\ (A^*, B^*, m^*) \leftarrow \mathcal{A}^{\mathcal{O}_{x,y}(\cdot)}(\mathsf{pp}, \hat{X}, \hat{Y}) \end{array} : \begin{array}{l} m^* \notin Q \ \wedge\ A^* \neq 1_{\mathbb{G}} \\ \wedge\ B^* = (A^*)^{x+m\cdot y} \end{array}\right],$$

where Q *is the set of queries that* $\mathcal{A}$ *has issued to the oracle* $\mathcal{O}_{x,y}(m) := Q \leftarrow Q \cup \{m\}; A \leftarrow G^*;$ **return** $(A, A^{x+m\cdot y})$.

We also recall the (aggregatable) multisignature signature of Boneh-Drijvers-Neven [BDN18], which uses two full-domain hash functions $\mathcal{H}_0 : \{0,1\}^* \to \mathbb{G}_2$ and $\mathcal{H}_1 : \{0,1\}^* \to \mathbb{Z}_p$.

$\underline{\mathsf{MSig.Setup}(1^\kappa)}$: $\mathsf{pp} \leftarrow\!\$\ \mathsf{BGGen}(1^\kappa)$; **return** pp.
$\underline{\mathsf{MSig.SKG}(\mathsf{pp})}$: $\mathsf{sk} \leftarrow\!\$\ \mathbb{Z}_p^*; \mathsf{pk} \leftarrow \mathsf{sk}\hat{G}$; **return** $(\mathsf{sk}, \mathsf{pk})$.
$\underline{\mathsf{MSig.KeyAgg}(\{\mathsf{pk}_1, \ldots, \mathsf{pk}_N\})}$:
$\mathsf{avk} \leftarrow \sum \mathcal{H}_1(\mathsf{pk}_i, \{\mathsf{pk}_1, \ldots, \mathsf{pk}_N\})\mathsf{pk}_i$; **return** avk.
$\underline{\mathsf{MSig.Sign}(\mathsf{sk}_i, \{\mathsf{pk}_1, \ldots, \mathsf{pk}_N\}, m)}$: **return** $\sigma_i = \mathsf{sk}_i \cdot \mathcal{H}_0(m)$.
//From all the individual signatures any combiner
//computes $\mathsf{msig} = \sum \mathcal{H}_1(\mathsf{pk}_i, \{\mathsf{pk}_1, \ldots, \mathsf{pk}_N\})\sigma_i$
$\underline{\mathsf{MSig.Verify}(\mathsf{avk}, m, \mathsf{msig})}$: **return** $e(G, \mathsf{msig}) = e(\mathcal{H}_0(m), \mathsf{avk})$.

G Detailed Performance Comparison With Related Work

We present the concrete costs for our work as well as the works from HPP20 and Rand-RCCA in Table 3 and Table 4 in descending order with respect to their asymptotic complexity. Scalar multiplications in groups $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ are denoted as E_1, E_2 and E_T, respectively. Pairing operations are denoted by P. We recall that n represents the number of users and N the number of mixers.

Table 3. Comparison of computational costs with prior work.

Scheme	Mixing
Rand-RCCA [FR22]	$(7n+6)E_1 + (7n+8)E_2 + 2nE_T + (9n+8)P$
HPP20 [HPP20]	$(10n+12N+11)E_1 + (7n+12N+10)E_2 + (8N-2)P$
Ours	$(6n+5)E_1 + (2n+N+2)E_2 + 2P$
	Verification
Rand-RCCA [FR22]	$(6N(n+1)-6n)E_1 + (6N+4nN)E_2 - 4nE_2 + 4NE_T + 4n(N-1)P$
Ours	$(14n+N+3)P$
HPP20 [HPP20]	$(8n+14)P$

Table 4. Comparison of communication costs with prior work.

Scheme	**Mixing**	
	Comm. (in)	Comm. (out)
Rand-RCCA [FR22]	$(7n + 2N)\mathbb{G}_1 + 8n\mathbb{G}_2 + n\mathbb{G}_T$	$(16n + 4)\mathbb{G}_1 + 12n\mathbb{G}_2 + 2n\mathbb{G}_T$
HPP20 [HPP20]	$(8n + 10N + 7)\mathbb{G}_1 + (6n + 8N + 8)\mathbb{G}_2$	$(8n + 17)\mathbb{G}_1 + (6n + 16)\mathbb{G}_2$
Ours	$(4n + N + 2)\mathbb{G}_1 + (4n + 5N)\mathbb{G}_2$	$(4n + 3)\mathbb{G}_1 + (4n + 2)\mathbb{G}_2$
	Verification	
Rand-RCCA [FR22]	$(16n + 4)\mathbb{G}_1 + 12n\mathbb{G}_2 + 2n\mathbb{G}_T$	
Ours	$(10n + 2N + 1)\mathbb{G}_1 + (8n + 7N + 1)\mathbb{G}_2$	
HPP20 [HPP20]	$(12n + 4)\mathbb{G}_1 + (14n + 7)\mathbb{G}_2$	