Scalable Mixnets from 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
{msyk.abe,octavio.perezkempner,mehdi.tibouchi}@ntt.com
[2] Kyoto University, Kyoto, Japan
nanri.masaya.26n@st.kyoto-u.ac.jp
[3] Security Fundamentals Laboratory, CSR, NICT
m.ohkubo@nict.go.jp
[4] Research Institute CODE, Universität der Bundeswehr München
daniel.slamanig@unibw.de

Abstract. A mix network, or *mixnet*, is a cryptographic tool for anonymous routing, taking messages from multiple (identifiable) senders and delivering them in a randomly permuted order. Traditional mixnets employ encryption and proofs of correct shuffle to cut the link between each sender and their input.

Hébant *et al.* (PKC '20) introduced a novel approach to scalable mixnets based on linearly homomorphic signatures. Unfortunately, their security model is too weak to support voting applications. Building upon their work, we leverage recent advances in equivalence class signatures, replacing linearly homomorphic signatures to obtain more efficient mixnets with security in a more robust model. More concretely, we introduce the notion of *mercurial signatures on randomizable ciphertexts* along with an efficient construction, which we use to build a scalable mixnet protocol suitable for voting. We compare our approach to other (scalable) mixnet approaches, implement our protocols, and provide concrete performance benchmarks. Our findings show our mixnet significantly outperforms existing alternatives in efficiency and scalability. Verifying the mixing process for 50k ciphertexts takes 135 seconds on a commodity laptop (without parallelization) when employing ten mixers.

Keywords: Equivalence Class Signatures, Mercurial Signatures, Mixnets, Voting, Anonymity

1 Introduction

The notion of a mixnet originates with the work on untraceable email by Chaum [Cha81], who proposed the use of multiple servers to shuffle a set of messages (*i.e.,* permute and perform cryptographic operations) in cascade to hide the relation between the initial input and resulting output. Since their introduction, mixnets have found numerous applications ranging from anonymous messaging [AKTZ17] and routing [CAB+15, KCGDF17, PHE+17] to voting (see *e.g.,* [CRS05, HMMP23a]) and even oblivious RAM [TDE17]. In general, mixnets are required to provide *verifiability* (*i.e.,* misbehavior during the mixing phase can be detected), which usually includes *accountability* (*i.e.,* misbehaving parties can be identified). Verifiable mixnets, *e.g.,* [SK95, Abe98, Abe99, AH01, FS01, Nef01, JJR02, Gro03, KMW12], require proof of correct shuffling, which adds a considerable overhead to the non-verifiable, *honest-but-curious* variant. Recent constructions such as [FLSZ17, AFK+20, KL21, ABGS23] assume various structures in the input ciphertext that make malicious behaviours by mix-servers harder, and hence, the proof and verification work easier. Unfortunately, this requires complex cryptographic building blocks, which are often error-prone and challenging for implementors. This complexity not only hampers widespread adoption but also raises concerns about the robustness of the implemented systems.

When mixnets are used in e-voting, one of its prime applications, they typically require users to create input ciphertexts to the mixnet. Such ciphertexts (the encrypted ballots) must be authenticated by an authority. This is crucial to enforcing the one-voter-one-vote principle. Therefore, the soundness of the election result requires trust placed in the authority. This need for authentication adds another layer of complexity.

Hébant, Phan and Pointcheval [HPP20] (HPP20 hereinafter) merged authentication into the mixnet structure itself; a certification authority (CA) authorizes each input with a signature, whose unforgeability is key to achieve an *easier* approach to verifiable shuffle. This enables a highly scalable mixnet architecture. The new paradigm requires the signatures to be malleable to carry out shuffling. Concretely, their mixnet requires three kinds of signature schemes (standard, linearly homomorphic [LPJY13], and multi-signatures [BDN18]), and the Groth-Sahai non-interactive zero-knowledge proof system [GS12] (GS proofs). Unfortunately, this results in a rather complex setup for the whole system and a new ad-hoc "unlinkability assumption" (Def. 4 from [HPP20]). More importantly, HPP20 guarantee soundness only for *honest users*, which is unacceptable for voting applications (see Appendix A for a detailed presentation of their model and related discussion). In this work, we address the following question:

Can we design a mixnet architecture that reduces implementation complexity and enhances scalability while providing strong security guarantees?

To answer the above question affirmatively, we follow HPP20's certified input paradigm leveraging recent results in Equivalence Class Signatures (EQS) [FHS19], allowing us to propose simpler and more efficient mixnets. EQS 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. Furthermore, an EQS allows anyone to jointly adapt a message-signature pair to obtain a new (and unlinkable) pair based on the concept of equivalence classes, originally defined as $[\boldsymbol{M}]_{\mathcal{R}} := \{\boldsymbol{N} \in (\mathbb{G}^*)^\ell \mid \exists\, s \in \mathbb{Z}_p^* \;:\; \boldsymbol{N} = s \cdot \boldsymbol{M}\}$ for any message vector $\boldsymbol{M} \in (\mathbb{G}^*)^\ell$ with $\ell \geq 2$. A forgery of EQS is considered as successful only if the forged message is not representative of any equivalence classes defined by the messages queried to the signing oracle. Besides, EQS provide a notion of *class-hiding* (it should be hard to distinguish whether two messages are in the same class or not) and signature adaptation (adapted signatures should be uniformly random in the space of valid signatures).

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, Bauer and Fuchsbauer [BF20] considered a different equivalence relation for the message space and gave the first construction of *Signatures on Randomizable Ciphertexts* (SoRC) [BFPV11] using an EQS. In brief, it signs ElGamal ciphertexts and all randomizations of a ciphertext define an equivalence class.

Our Contribution. In search of more efficient constructions of scalable mixnets with stronger security properties, we take the SoRC from [BF20] (which is an EQS) as a starting point and extend it to get a *Mercurial Signature on Randomizable Ciphertexts* (MSoRC). However, this is not yet a game changer as 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). To overcome this limitation, we incorporate a very recent idea used to construct interactive *threshold mercurial signatures* (TMS) [ANPT24] to obtain a construction that provides a stronger class-hiding notion for the public key. In brief, we present an interactive signing protocol for our base MSoRC scheme, allowing 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 do not share their secret keys. We believe that our new primitive is interesting on its own and might find various other applications.

Then we go on to show how we can build a scalable mixnet from MSoRC, obtaining a similar key-randomizability property when compared to HPP20, but without the hurdle of combining linearly homomorphic signatures and GS proofs. Moreover, we tailor our MSoRC to the mixnet setting, further optimizing it. In addition, we rely on a simpler proof system from Couteau and Hartmann (CH20) [CH20] instead of GS proofs, and use batch verification to further improve efficiency. Overall, this allows us to reduce computation and verification costs compared to HPP20, while achieving stronger security guarantees under minimal trust assumptions, *i.e.,* a single honest mixer.

To show the practicality of our approach, we discuss the application of our mixnet for voting. We provide a comparison with state-of-the-art alternatives and a Rust implementation of our protocols alongside corresponding benchmarks demonstrating the efficiency and scalability of our approach.

We emphasize that all the cryptographic building blocks used in this work can be easily implemented with existing cryptographic libraries. Additionally, no complex arithmetic operations or algorithms are needed.

2 Related Work

Signatures on Randomizable Ciphertexts. Originally introduced by Blazy *et al.* [BFPV11], SoRC can be seen as a predecessor of EQS [HS14, FHS19] for a specific equivalence relation. The motivation was to build signatures on ciphertexts that could be adapted to randomizations of them. The work by Hanser and Slamanig [HS14] together with Fuchsbauer [FHS19] that introduced EQS allows a controlled form of malleability on message-signature pairs where messages are not ciphertexts but broadly, can be any tuple defined over $(\mathbb{G}^*)^\ell$ with $\ell \geq 2$. To do so, the authors put forth the notion of equivalence classes, adapting the usual unforgeability notion, as previously mentioned. The SoRC construction from [BF20] (which is based on [FHS19]) provides a strong notion of *class-hiding*, *i.e.,* an adapted message-signature pair looks like a completely random message-signature pair even when knowing the original message-signature pair.

Our work extends the construction from [BF20] so that signatures can be adapted to a new verification key representative. Hence, we obtain the first key-randomizable SoRC (*i.e.,* a MSoRC). We use the technique from [ANPT24] to define an interactive signing protocol to compute a MSoRC jointly. Consequently, the corresponding secret key is distributed between two parties and we achieve a strong class-hiding property for the verification keys (public key unlinkability). Looking ahead, users of our mixnet protocol engage with the authority to produce an MSoRC signature for a message they choose, allowing them to adapt the signature thus hiding their identity while still being able to prove that it is valid.

To summarize, our work combines the notion of TMS and SoRC to present a new primitive (MSoRC) that provides enhanced privacy-preserving features.

Mixnets. The closest work to ours is HPP20, which inspires this work. It is based on linearly homomorphic signatures and GS proofs. We propose to replace both building blocks, introducing a new signature scheme that can be jointly computed between each user and the certificate authority (CA). Our scheme is an interactive MSoRC, and we use it alongside a different proof system to further improve the efficiency of the shuffle approach from HPP20. On the one hand, the use of MSoRC allows us to remove the need to combine two different linearly homomorphic signatures with GS proofs to manage the key-randomizability property required for privacy. On the other hand, the use of CH20 allows us to reduce the computational cost per mix-server. We also improve the security model. While HPP20 is unsuitable for voting (it only provides guarantees for *honest* users), our mixnet can be applied in voting applications as discussed in Section F.

Another recent line of work based on Re-randomizable Replayable CCA encryption (Rand-RCCA) [CKN03] shares similarities with the above, making it appropriate to discuss. Faonio *et al.* [FFHR19, FR22] proposed using Rand-RCCA PKE to circumvent the need for a proof of shuffle, carefully replacing it with individual NIZK proofs of plaintext knowledge for each ciphertext and NIZK proofs of membership for each mixing stage. At the end, servers run a multi-party computation protocol (called *verify-then-decrypt*) to obtain the final output. The similarity with HPP20 is that ciphertexts can be independently randomized, and thus, their solution also scales very well. Unfortunately, proving security in the universal composability (UC) framework comes at a much higher computational cost for their solution and they also require a rather complex setup.

As we discuss in Section 7, from a communication as well as computational perspective our approach outperforms both HPP20 as well as the Rand-RCCA PKE based approach [FR22]. In particular, compared to HPP20, which is more efficient than [FR22], we improve computation by a factor of $3.5x$ and communication up to a factor of around $3x$. Moreover, also concretely our approach is highly efficient and for instance for $n = 50\mathsf{k}$ ciphertexts and $N = 10$ mixers, the worst-case time for mixing takes around 40 seconds and the verification of the final mixing result takes around 135 seconds.

We do not consider *decryption mixnets* (*e.g.,*[DHK21, PHE+17]) in this work as we focus on mixnets suitable for applications like e-voting, which are require public verifiability of the mixing.

3 Our Approach

We present a high-level overview of our approach, focusing on voting, its primary application. We assume that a Bulletin Board (BB) is available to all parties.

The idea is that users encrypt their ballots with ElGamal and engage in an interactive signing protocol with the CA to produce a MSoRC signature for it, which verifies under a jointly computed public key. For security, the user will contribute to one share of the corresponding secret key, and the authority with two shares (an ephemeral secret key and its long-term secret key). The ephemeral public key is given to the user. This way, users can publish their ciphertexts and signatures alongside the ephemeral public key to prove they have a valid vote. Anyone can compute a verification key vk_i as the product of user i's public key with the posted ephemeral public key and the authority's public key ($\mathsf{vk}_i = U_{i,\mathsf{pk}} + E_{i,\mathsf{pk}} + A_{\mathsf{pk}}$) to get a (valid) key under which the signature verifies. Unforgeability of MSoRC signatures guarantees that if a signature verifies under vk_i, it must have been produced with the authority's participation (*i.e.,* the user is allowed to post a vote).

The above signing protocol includes randomizing the user's ciphertext, ensuring *receipt-freeness* (see Sec. F). In brief, users get a valid signature that can be adapted to a randomization of the user ciphertext but they cannot prove to a third party that the ciphertext encrypts a particular message.

With the above in mind, a user tuple posted to the BB consists of a ciphertext C_i, a signature σ_i, the user's public key $U_{i,\mathsf{pk}}$, and the ephemeral public key $E_{i,\mathsf{pk}}$. Since $U_{i,\mathsf{pk}} + E_{i,\mathsf{pk}}$ determines a unique public key, users could submit $U_{i,\mathsf{pk}} + E_{i,\mathsf{pk}}$ instead of $U_{i,\mathsf{pk}}$ and (separately) $E_{i,\mathsf{pk}}$. This would allow them to remain anonymous to other users in the system while the CA can still identify them. In any case, replay attacks (re-voting) at this stage are easily detectable as it suffice to verify that all the submitted tuples are different (see Sec. 5 for the details of our signature construction).

The first mix server gets all the tuples, randomizes each ciphertext and verification key, adapts the signatures, and shuffles everything. 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.

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

$$\mathsf{NIZK}\{(\sum_{i=1}^{i=n} \mathsf{vk}'^{(j-1)}_{\Pi(i)}, \rho) : \sum_{i=1}^{i=n} \mathsf{vk}'^{(j)}_{\Pi(i)} = \rho \cdot \sum_{i=1}^{i=n} \mathsf{vk}'^{(j-1)}_{\Pi(i)}\}$$

Servers sign their NIZK proof using an aggregate signature, and we use batch verification for all proofs. Everyone can publicly verify the aggregate signature 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 aggregate signature and 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 the final proof, making the final verification independent of N.

4 Preliminaries

Notation. The set of integers from 1 to n is denoted as $[n]$. $\mathbb{Z}_p$ represents the ring of integers modulus 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. We denote by $\mathcal{PP}$ the set of public parameters, and for $\mathsf{pp} \in \mathcal{PP}$ we let $\mathcal{M}_{\mathsf{pp}}$ be the set of messages, $\mathcal{DK}_{\mathsf{pp}}$ the set of decryption keys, $\mathcal{EK}_{\mathsf{pp}}$, the set of encryption keys, $\mathcal{C}_{\mathsf{pp}}$ the set of ciphertexts, $\mathcal{R}_{\mathsf{pp}}$ the set of ciphertext randomness, $\mathcal{SK}_{\mathsf{pp}}$ the set of signature keys, $\mathcal{VK}_{\mathsf{pp}}$ the set of verification keys and $\mathcal{S}_{\mathsf{pp}}$ the set of signatures. Let BGGen be a PPT algorithm that on input 1^κ, returns public parameters $\mathsf{pp} \in \mathcal{PP}$ s.t. $\mathsf{pp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, G, \hat{G}, e)$ describes 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. Elements in $\mathbb{G}_2$ are written with a hat (*e.g.,* $\hat{X} \in \mathbb{G}_2$).

ElGamal PKE [ElG86]. Let $\mathsf{pp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, G, \hat{G}, e)$ and (KeyGen, Enc, Dec). 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 $M := C_1 - xC_0$. The ElGamal PKE is IND-CPA in $\mathbb{G}_1$ as long as the DDH assumption holds in $\mathbb{G}_1$.

Zero-Knowledge Proofs. We consider languages in NP defined in terms of a relation $\mathcal{L}_{\mathcal{R}} = \{\mathsf{x} | \exists \mathsf{w} \text{ st. } (\mathsf{x}, \mathsf{w}) \in \mathcal{R}_{\mathcal{L}}\}$, where $\mathsf{x} \in X$ is referred to as the *instance* and $\mathsf{w} \in W$ as the *witness* with $\mathcal{R}_{\mathcal{L}}$ being a subset of $X \times W$. A zero-knowledge proof allows a prover to convince a verifier that $(\mathsf{x}, \mathsf{w}) \in \mathcal{R}_{\mathcal{L}}$ without disclosing any information about w. This work uses Zero-Knowledge Proofs of Knowledge (ZKPoK) and non-interactive arguments (*i.e.,* Non-Interactive Zero-Knowledge arguments or NIZK). The former are three-round public coin, honest verifier zero-knowledge proofs that satisfy *knowledge soundness* (see [Gol01] for further details). The latter are single-round protocols in the common reference string (crs) model whose syntax we recall next (we refer the reader to [DEF+23] and [CH20] for formal definitions). A NIZK proof system for a language $\mathcal{L}$ is defined by three algorithms: (1) CRSGen generates a common reference string 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 B, as required by our mixnet scheme.

Mercurial Signatures on Randomizable Ciphertexts. Our definition of mercurial signatures on randomizable ciphertexts adapts the presentation from [BF20] to signatures on randomizable ciphertexts (similar to what [CL19] does for mercurial signatures when generalizing the ideas from [FHS19]). Thus, the definitions below can be seen as a merge between the original syntax and security properties of SoRC and MS schemes. With this in mind, a MSoRC scheme consists of the following polynomial-time algorithms of which all except Setup are implicitly parametrized by an element $\mathsf{pp} \in \mathcal{PP}$.

$\underline{\mathsf{Setup}(1^\kappa) \to \mathsf{pp}}$: Generates public parameters.

$\underline{\mathsf{KeyGen}() \to (\mathsf{ek}, \mathsf{dk})}$: Generates an encryption key pair.

$\underline{\mathsf{Enc}(\mathsf{ek}, m; r) \to c}$: Produces a ciphertext c under ek for a message m using randomness r.

$\underline{\mathsf{Dec}(\mathsf{dk}, c) \to m}$: Outputs a message m.

$\underline{\mathsf{Rndmz}(\mathsf{ek}, c; \mu) \to c'}$: Randomizes a ciphertext c into c' using randomness μ.

$\underline{\mathsf{SKG}() \to (\mathsf{vk}, \mathsf{sk})}$: Generates a signature key pair.

$\underline{\mathsf{Sign}(\mathsf{sk}, \mathsf{ek}, c; s) \to \sigma}$: Produces a signature σ for c under sk using randomness s.

$\underline{\mathsf{Verify}(\mathsf{vk}, \mathsf{ek}, c, \sigma) \to 0/1}$: Verifies (c, σ) w.r.t. vk and ek.

$\underline{\mathsf{Adapt}(\sigma; \mu, \rho) \to \sigma'}$: Randomizes a signature σ into σ' using randomness μ and ρ.

$\underline{\mathsf{ConvertSK}(\mathsf{sk}, \rho) \to \mathsf{sk}'}$: Randomizes a secret key sk into sk' using randomness ρ.

$\underline{\mathsf{ConvertVK}(\mathsf{vk}, \rho) \to \mathsf{vk}'}$: Randomizes a verification key vk into vk' using randomness ρ.

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 (Correctness [BF20]). *A* SoRC *scheme is correct if for all* $\mathsf{pp} \in \mathcal{PP}$*, for all pairs* $(\mathsf{ek}, \mathsf{dk})$ *and* $(\mathsf{sk}, \mathsf{vk})$ *in the range of* $\mathsf{KeyGen}(\mathsf{pp})$ *and* $\mathsf{SKG}(\mathsf{pp})$*, respectively, and all* $m \in \mathcal{M}_{\mathsf{pp}}$, $r \in \mathcal{R}_{\mathsf{pp}}$ *and* $c \in \mathcal{C}_{\mathsf{pp}}$*:* $\mathsf{Dec}(\mathsf{dk}, \mathsf{Enc}(\mathsf{ek}, m; r)) = m$ *and* $\Pr[\mathsf{Verify}(\mathsf{vk}, \mathsf{ek}, c, \mathsf{Sign}(\mathsf{sk}, \mathsf{ek}, c)) = 1] = 1$.

Similar to mercurial signatures, unforgeability of MSoRC 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 with them and how they are managed in the unforgeability game. In this regard, the unforgeability notion from BF20 considers a forgery the case in which the adversary can produce a signature on an encryption of a message for

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

Fig. 1. Unforgeability experiment (UNF-I).

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. We capture this setting with the following definition.

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

While the previous unforgeability notion enables applications such as blind signatures [BFPV11], for our concrete application of mixnet, the encryption keys 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 from Definition 2 so that it's the challenger the one that picks the encryption key pair instead of the adversary[5]. We reflect this with Definition 3.

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

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

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

An MSoRC should also provide an encryption scheme with IND-CPA security and full class-hiding. These properties were defined in [BF20] and are recalled below.

Definition 4 (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* $\boldsymbol{Adv}^{\mathsf{IND-CPA}}_{\mathsf{MSoRC},\mathcal{A}}(\kappa) := 2\cdot Pr\left[\boldsymbol{Exp}^{\mathsf{IND-CPA}}_{\mathsf{MSoRC},\mathcal{A}}(\kappa) \Rightarrow \mathsf{true}\right] - 1 = \epsilon(\kappa)$.
Full class-hiding: *the advantage of any* PPT *adversary* $\mathcal{A}$ *defined by* $\boldsymbol{Adv}^{\mathsf{Full-CH}}_{\mathsf{MSoRC},\mathcal{A}}(\kappa) := 2\cdot Pr\left[\boldsymbol{Exp}^{\mathsf{Full-CH}}_{\mathsf{MSoRC},\mathcal{A}}(\kappa) \Rightarrow \mathsf{true}\right] - 1 = \epsilon(\kappa)$.

where $\boldsymbol{Exp}^{\mathsf{IND-CPA}}_{\mathsf{MSoRC},\mathcal{A}}(\kappa)$ *and* $\boldsymbol{Exp}^{\mathsf{Full-CH}}_{\mathsf{MSoRC},\mathcal{A}}(\kappa)$ *are the experiments shown below.*

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

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

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 $c^* \notin Q \wedge [\mathsf{vk}^*]_{\mathsf{vk}} = [\mathsf{vk}]_{\mathsf{vk}} \wedge \mathsf{Verify}(\mathsf{vk}^*, \mathsf{ek}, c^*, \sigma^*) = 1$

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. 3. Unforgeability w.r.t an interactive signing protocol.

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

Definition 5 (Signature adaption). *A* MSoRC *scheme is adaptable (under malicious keys) if for all* $\mathsf{pp} \in \mathcal{PP}$*, all* $(\mathsf{vk}, \mathsf{ek}, c, \sigma) \in \mathcal{VK}_{\mathsf{pp}} \times \mathcal{EK}_{\mathsf{pp}} \times \mathcal{C}_{\mathsf{pp}} \times \mathcal{S}_{\mathsf{pp}}$ *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* $\{\sigma' \in \mathcal{S}_{\mathsf{pp}} \mid \mathsf{Verify}(\mathsf{ConvertVK}(\mathsf{vk}, \rho), \mathsf{ek}, \mathsf{Rndmz}(\mathsf{ek}, c, \mu), \sigma') = 1\}$.

Besides standard definitions, we also consider an interactive signing protocol for MSoRC schemes as defined below.

$\mathsf{ISign}_{\mathsf{P}_0}(\mathsf{sk}_0, \mathsf{ek}, c) \leftrightarrow \mathsf{ISign}_{\mathsf{P}_1}(\mathsf{sk}_1, \mathsf{ek}, c) \to \sigma$: This algorithm is run interactively between parties P_0 and P_1. It produces a signature σ for c under sk, implicitly defined as $\mathsf{sk}_0 + \mathsf{sk}_1$.

We now define unforgeability and public-key class-hiding, assuming at least one honest signer. To prove security, we introduce a key generation 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 [ANPT24].

Definition 6 (Security of key generation). TKGen *is secure if it outputs* vk *with the same distribution as* SKG*, and there exists a simulator,* 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 signing parties and leak its corresponding keys. As in Definition 3, the challenger picks the encryption key pair.

Definition 7 (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. 3.*

For public key class-hiding, we adapt the original definition from [CL19] in the vein of [ANPT24], that is, 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, we get a full public key class hiding notion when there is no collusion between the two parties. Following the naming convention from [ANPT24], we formalize this notion as *public key unlinkability*. As we shall see, this notion suffices for the discussed applications.

Definition 8 (Public Key Unlinkability). *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* $\boldsymbol{Exp}^{t-\mathsf{PK-UNL}}_{\mathsf{MSoRC}}(1^\kappa, \mathcal{A})$ *is shown in Fig. 4.*

5 Our Signature Scheme

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 $Z = \frac{1}{s}(x_0C_0 + x_1C_1 + G), S = sG, \hat{S} = s\hat{G}$ and $T = \frac{1}{s}(x_0G + x_1X)$, where (C_0, C_1) is the ciphertext, X it's public key, and (x_0, x_1)

Experiment $\mathbf{Exp}^{\mathsf{PK\text{-}UNL}}_{\mathsf{MSoRC}}(1^\kappa, \mathcal{A})$

$\mathsf{pp} \leftarrow\!\$\ \mathsf{Setup}(1^\kappa); \rho \leftarrow\!\$\ \mathcal{R}_{\mathsf{pp}}; b \leftarrow\!\$\ \{0,1\}; (\widetilde{\mathsf{sk}}, \widetilde{\mathsf{vk}}) \leftarrow\!\$\ \mathsf{TKGen}(\mathsf{pp}); (\mathsf{sk}_i, \mathsf{vk}_i)_{i\in\{0,1\}} \leftarrow\!\$\ \mathsf{TKGen}(\mathsf{pp})$

$\mathsf{vk}' \leftarrow \mathsf{ConvertVK}(\widetilde{\mathsf{vk}} + \mathsf{vk}_b, \rho); b' \leftarrow\!\$\ \mathcal{A}^{\mathsf{ISign}(\mathsf{sk}_b,\cdot,\cdot)}(\widetilde{\mathsf{sk}}, \widetilde{\mathsf{vk}}, \mathsf{vk}', \mathsf{vk}_0, \mathsf{vk}_1)$

return $b = b'$

Oracle $\mathsf{ISign}(\mathsf{sk}_b, \mathsf{ek}, c, \mathsf{vk})$

if $\mathsf{vk} = \mathsf{vk}'$ **then** $\sigma \leftarrow\!\$\ \mathsf{ISign}_b(\mathsf{sk}_b, \mathsf{ek}, c)$ **return** $\mathsf{Adapt}(\sigma; \rho)$

elseif $\mathsf{vk} = \mathsf{vk}_i$ **return** $\mathsf{ISign}(\mathsf{sk}_i, \mathsf{ek}, c)$

Fig. 4. Public key unlinkability experiment.

the scheme's secret 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. We recall that SPS cannot have less than 3 group elements [AGHO11]. This base construction has almost optimal size and is given below.

$\mathsf{MSoRC.KeyGen}()$: $\mathsf{dk} := x \leftarrow\!\$\ \mathbb{Z}_p^*$; $\mathsf{ek} := X \leftarrow xG$; **return** $(\mathsf{dk}, \mathsf{ek})$.

$\mathsf{MSoRC.Enc}(X, M; r)$: **return** $(rG, M + rX)$.

$\mathsf{MSoRC.Dec}(x, (C_0, C_1))$: **return** $M := C_1 - xC_0$.

$\mathsf{MSoRC.Rndmz}(X, (C_0, C_1); \mu)$: **return** $(C_0 + \mu G, C_1 + \mu X)$.

$\mathsf{MSoRC.SKG}()$: $\mathsf{sk} := (x_0, x_1, x_2) \leftarrow Z_p^*$; $\mathsf{vk} := (x_0\hat{G},\ x_1\hat{G}, x_2\hat{G})$; **return** $(\mathsf{sk}, \mathsf{vk})$.

$\mathsf{MSoRC.Sign}((x_0, x_1, x_2), X, (C_0, C_1))$:

$s \leftarrow\!\$\ Z_p^*$; $Z := \frac{1}{s}(x_0C_0 + x_1C_1 + x_2G)$; $S := sG$; $\hat{S} := s\hat{G}$; $T := \frac{1}{s}(x_0G + x_1X)$; **return** $(Z, S, \hat{S}, T)$.

$\mathsf{MSoRC.Verify}((\hat{X}_0, \hat{X}_1), X, (C_0, C_1), (Z, S, \hat{S}, T))$: **return** 1 if and only if

$$\begin{aligned} &e(Z, \hat{S}) = e(C_0, \hat{X}_0)e(C_1, \hat{X}_1)e(G, \hat{X}_2) \wedge \\ &e(T, \hat{S}) = e(G, \hat{X}_0)e(X, \hat{X}_1) \ \wedge\ e(S, \hat{G}) = e(G, \hat{S}) \end{aligned}$$

$\mathsf{MSoRC.Adapt}((Z, S, \hat{S}, T); \mu, \rho)$: $s' \leftarrow\!\$\ Z_p^*$;

$Z' := \frac{\rho}{s'}(Z + \mu T)$; $S' := s'S$; $\hat{S}' := s'\hat{S}$; $T' := \frac{\rho}{s'}T$; **return** $(Z', S', \hat{S}', T')$.

$\mathsf{MSoRC.ConvertSK}((x_0, x_1, x_2), \rho)$: **return** $(\rho x_0, \rho x_1, \rho x_2)$.

$\mathsf{MSoRC.ConvertVK}((\hat{X}_0, \hat{X}_1, \hat{X}_2), \rho)$: **return** $(\rho\hat{X}_0, \rho\hat{X}_1, \rho\hat{X}_2)$.

We can further optimize the above construction tailoring it to our mixnet application: we drop S to obtain a shorter signature (with optimal size), reducing the number of pairings used in verification by two. Moreover, we can also extend our construction to support a two-party *interactive* signing protocol as shown in Fig. 5. We do so using the techniques from [ANPT24] 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). We stress that the four ZKPoK involved are as simple to implement as a Schnorr proof.

Security. As in related work ([ANPT24, BF20]), we consider the stand-alone model and adversaries in the Generic Group Model (GGM) for asymmetric bilinear groups to prove the security. Correctness of our base scheme follows by inspection. In Appendix C, we argue that the interactive variant produces signatures under the same distribution. Next, we discuss an outline of the unforgeability proof for the base construction (which is the more general one) and defer the full details to Appendix C where we also prove unforgeability of the optimized version.

Theorem 9 (Unforgeability). *Our base* MSoRC *is unforgeable in the GGM w.r.t. Definition 2 if all* ZKPoK*'s are secure.*

$$
\begin{array}{lcl}
\underline{\mathsf{P}_0\colon C_0, C_1, X, \{\hat{X}_i^0 = x_i^0\hat{G}, x_i^0, \hat{X}_i^1\}_{i\in\{0,1,2\}}} & & \underline{\mathsf{P}_1\colon C_0, C_1, X, \{\hat{X}_i^1 = x_i^1\hat{G}, x_i^1, \hat{X}_i^1\}_{i\in\{0,1,2\}}} \\
s_0 \leftarrow\!\!\$\ \mathbb{Z}_p^*;\ S_0 \leftarrow s_0 G;\ \hat{S}_0 \leftarrow s_0\hat{G} & & r \leftarrow\!\!\$\ \mathbb{Z}_p;\ s_1 \leftarrow\!\!\$\ \mathbb{Z}_p^* \\
\pi_0 \leftarrow \mathsf{ZKPoK}[s_0] & \xrightarrow{S_0, \hat{S}_0, \pi_0} & \hat{S} \leftarrow s_1\hat{S}_0; Z_1 \leftarrow rS_0 + x_0^1 C_0 + x_1^1 C_1 + x_2^1 G \\
 & & T_1 \leftarrow rS_0 + x_0^1 G + x_1^1 X \\
T_0 \leftarrow \frac{1}{s_0}(T_1 + x_0^0 G + x_1^0 X) & \xleftarrow{T_1, Z_1, \pi_1} & \pi_1 \leftarrow \mathsf{ZKPoK}[r, x_0^1, x_1^1, x_2^1] \\
Z_0 \leftarrow \frac{1}{s_0}(Z_1 + x_0^0 C_0 + x_1^0 C_1 + x_2^0 G) & & \\
\widetilde{\pi}_0 \leftarrow \mathsf{ZKPoK}[s_0, x_0^0, x_1^0, x_2^0] & \xrightarrow{Z_0, T_0, \widetilde{\pi}_0} & T \leftarrow \frac{1}{s_1}(T_0 - rG); Z \leftarrow \frac{1}{s_1}(Z_0 - rG) \\
 & & \widetilde{\pi}_1 \leftarrow \mathsf{ZKPoK}[r, s_1] \\
\textbf{return } (\sigma, \widetilde{\pi}_1) & \xleftarrow{\sigma, \widetilde{\pi}_1} & \sigma \leftarrow (Z, \boxed{S}, \hat{S}, T);\ \textbf{return } (\sigma, \widetilde{\pi}_1)
\end{array}
$$

Fig. 5. Our two-party interactive signing algorithm. ZKPoK's are defined as: $\mathsf{ZKPoK}[s_0 : S_0 = s_0 G \wedge \hat{S}_0 = s_0\hat{G}]$, $\mathsf{ZKPoK}[(s_0, x_0^0, x_1^0, x_2^0) : T_0 = \frac{1}{s_0}(T_1 + x_0^0 G + x_1^0 X) \wedge S_0 = s_0 G \wedge Z_0 = \frac{1}{s_0}(Z_1 + x_0^0 C_0 + x_1^0 C_1 + x_2^0 G) \wedge \hat{X}_0^0 = x_0^0\hat{G} \wedge \hat{X}_1^0 = x_1^0\hat{G} \wedge \hat{X}_2^0 = x_2^0\hat{G}]$, $\mathsf{ZKPoK}[(r, x_0^1, x_1^1, x_2^1) : T_1 = rS_0 + x_0^1 G + x_1^1 X \wedge Z_1 = rS_0 + x_0^1 C_0 + x_1^1 C_1 + x_2^1 G \wedge \hat{X}_0^1 = x_0^1\hat{G} \wedge X_1^1 = x_1^1\hat{G} \wedge X_1^2 = x_2^1\hat{G}]$, $\mathsf{ZKPoK}[(r, s_1) : T = \frac{1}{s_1}(T_0 - rG) \wedge S = s_1 S_0 \wedge \hat{S} = s_1\hat{S}_0 \wedge Z = \frac{1}{s_1}(Z_0 - rG)]$. Enclosed in a box, the modification w.r.t the base scheme.

Proof (sketch): In the GGM, the adversary is given encodings of group elements from the bilinear group (*i.e.,* random strings), and it can query the respective group oracles to perform group operations (*e.g.,* test equality, sum, inversion, etc.). To prove unforgeability we first consider an adversary against our base construction (Def. 2). In brief, we give a reduction that uses an adversary against our MSoRC scheme to produce a forgery for SoRC scheme in [BF20] (denoted BF20). The reduction receives the public key $\mathsf{pk} = (\hat{X}_0, \hat{X}_1)$ of BF20, sets $\mathsf{pk}' = (\alpha\hat{X}_0, \alpha\hat{X}_1, \alpha\hat{G})$ for $\alpha \leftarrow\!\!\$\ Z_p^*$ and interacts with the adversary $\mathcal{A}$. Whenever $\mathcal{A}$ asks for a signature, the reduction forwards the request to the challenger of BF20. On response $\sigma = (Z, T, S, \hat{S})$, the reduction sets $\sigma' = (\alpha Z, \alpha T, S, \hat{S})$ and sends it back to $\mathcal{A}$. Whenever $\mathcal{A}$ outputs $(Z^*, T^*, S^*, \hat{S}^*)$ for public key $\mathsf{pk}^* = \beta\mathsf{pk}'$, $\mathcal{B}$ outputs $(\frac{1}{\alpha\beta}Z^*, \frac{1}{\alpha\beta}T^*, S^*, \hat{S}^*)$ as its forgery for BF20. We note that $\mathcal{B}$ is a generic forger and thus, it can obtain β.

To prove that our scheme is also unforgeable when considering an interactive two-party signing protocol we follow the approach from [ANPT24], which proved a similar result for their threshold mercurial signature scheme. In brief, we give a simulator whose advantage in breaking unforgeability for the interactive case is no greater than that of the original game. To do so, we have to consider the cases in which both parties are honest and when one is malicious.

Our MSoRC's provide IND-CPA and full class hiding. 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 and so we omit its proof.

Theorem 10 (Signature adaption). *Our* MSoRC *scheme is signature-adaptable under malicious keys.*

The proof follows directly from that of the original SoRC ([BF20], Proposition 2) in both cases (base and optimized) and thus, we also omit it.

Theorem 11 (Public Key Unlinkability). *Our* MSoRC *scheme is public key unlinkable under corruption of at most one party.*

Proof. It suffices to 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:

$$
\begin{aligned}
Z &= \frac{1}{s}(\rho(\widetilde{\mathsf{sk}}^0 + \mathsf{sk}_b^0)C_0 + \rho(\widetilde{\mathsf{sk}}^1 + \mathsf{sk}_b^1)C_1 + \rho(\widetilde{\mathsf{sk}}^1 + \mathsf{sk}_b^1)G) \\
T &= \frac{1}{s}(\rho(\widetilde{\mathsf{sk}}^0 + \mathsf{sk}_b^0)G + \rho(\widetilde{\mathsf{sk}}^1 + \mathsf{sk}_b^1)X), S = sG,\ \hat{S} = s\hat{G}
\end{aligned}
$$

Since ρ is uniformly random, it perfectly hides b and the adversary gains no information dependent on b. The same reasoning applies to the optimized variant.

$\underline{\mathsf{MixSetup}(1^\kappa):}$
$\mathsf{pp}_1 := (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, G, \hat{G}, e) \leftarrow\!\!\$\ \mathsf{BGGen}(1^\kappa); (\mathsf{pp}_2 = \hat{Z}, \cdot) \leftarrow\!\!\$\ \mathsf{NIZK.CRSGen}(1^\kappa)$
$(\mathsf{pp}_3 = (W, \hat{W}), \cdot) \leftarrow\!\!\$\ \mathsf{SAS.Setup}(1^\kappa); \mathsf{pp} \leftarrow (\mathsf{pp}_1, \mathsf{pp}_2, \mathsf{pp}_3); \textbf{return } \mathsf{pp}$

$\underline{\mathsf{MixElGamal}(1^\kappa)}$: $\mathsf{dk} := x \leftarrow\!\!\$\ \mathbb{Z}_p^*; \mathsf{ek} := X \leftarrow xG; \textbf{return } (\mathsf{ek}, \mathsf{dk})$

$\underline{\mathsf{MixKG}(\mathsf{pp}):}$
$\mathcal{S}_i\ : (\mathsf{ssk}_i, \mathsf{spk}_i) \leftarrow\!\!\$\ \mathsf{SAS.SKG}(\mathsf{pp}_3) \qquad u_i\ : (\mathsf{usk}_i, \mathsf{uvk}_i) \leftarrow\!\!\$\ \mathsf{MSoRC.SKG}(\mathsf{pp}_1)$
$\mathsf{CA} : (\mathsf{ask}, \mathsf{avk}) \leftarrow\!\!\$\ \mathsf{MSoRC.SKG}(\mathsf{pp}_1)$

$\underline{\mathsf{MixSign}_{u_i}(\mathsf{usk}_i, \mathsf{uvk}_i, \mathsf{avk}, M_i, \mathsf{ek}) \leftrightarrow \mathsf{MixSign}_{\mathsf{CA}}(\mathsf{ask}, \mathsf{avk}, \mathsf{uvk}_i, \mathsf{ek}):}$

$$\begin{array}{lcl}
u_i: & & \mathsf{CA}: \\
(C_0, C_1) \leftarrow \mathsf{ElGamal.Enc}(\mathsf{ek}, M_i; \gamma) & & (\mathsf{esk}_i, \mathsf{evk}_i) \leftarrow\!\!\$\ \mathsf{MSoRC.SKG}() \\
\pi \leftarrow \mathsf{ZKPoK}[(\gamma, \mathsf{usk}_i) : C_0 = \gamma G & & \\
\wedge\ \mathsf{uvk}_i = \mathsf{usk}_i \cdot \hat{G}] & \xrightarrow{(C_0, C_1), \pi} & (C_0', C_1') \leftarrow \mathsf{Rndmz}(\mathsf{ek}, (C_0, C_1); \mu) \\
 & \xleftarrow{(C_0', C_1'), \pi', \mathsf{evk}_i} & \pi' \leftarrow \mathsf{ZKPoK}[\mu : (C_0', C_1') \\
 & & = \mathsf{Rndmz}(\mathsf{ek}, (C_0, C_1); \mu)] \\
 & \xleftrightarrow{\mathsf{MSoRC.ISign}} & \\
\sigma_i \leftarrow \mathsf{MSoRC.ISign}_{\mathsf{P}_0}(\mathsf{usk}_i, (C_0', C_1')) & & \sigma_i \leftarrow \mathsf{MSoRC.ISign}_{\mathsf{P}_1}(\boxed{\mathsf{esk}_i + \mathsf{ask}}, (C_0, C_1)) \\
\mathsf{vk}_i \leftarrow \mathsf{uvk}_i + \mathsf{evk}_i + \mathsf{avk} & & \\
\textbf{return } (\sigma_i, (C_0', C_1'), \mathsf{evk}_i) & &
\end{array}$$

Fig. 6. Algorithms MixSetup, MixElGamal, MixKG and MixSign.

6 Our Mixnet Scheme

6.1 Building blocks

Our mixnet scheme requires three different building blocks: an MSoRC to sign ciphertexts, a NIZK proof system to prove the correct randomization of verification keys, and an aggregate signature (or multi signature to optimize verification).

As previously mentioned, the NIZK proof system from CH20 is suitable to prove discrete log relationships and it's setup is simpler than that of GS proofs. Besides, batch verification considerably reduces the costs. We use the sequential aggregate signature (SAS) from Pointcheval and Sanders [PS16] (see Appendix E for details) as it suits our setting. First of all, its setup is compatible with that of the MSoRC scheme as it also requires a bilinear group of type-III and the common random string can be generated in the same way as that of the CH20 NIZK (see Appendix. B). Furthermore, in terms of efficiency, signatures are only two elements in $\mathbb{G}_1$, signing requires only three exponentiations in $\mathbb{G}_1$ (beyond the verification of the aggregate signature up to that point) and the verification cost is two pairings and N exponentiations in $\mathbb{G}_2$ when verifying N messages.

6.2 Construction

This section provides a more detailed discussion of the technical decisions behind our mixnet scheme, shown in figures 6 and 7.

MixSetup. This algorithm samples the parameters for each building block. For ease of exposition we assume that a trusted party does the whole setup. However, we stress that all parameters can be produced via a multi-party protocol in a distributed way (see *e.g.,* [BCG+15]).

MixElGamal. This algorithm generates an ElGamal key pair. In a decentralized setting, multiple parties (*e.g.,* polling authorities) can run a DKG protocol [Ped91] to distribute trust.

MixKG. This algorithm is independently run by each entity. We assume the usual certified-key setting where parties register their public key with the CA and prove knowledge of their secret key (to avoid rogue key attacks). We note that for all users, this explicitly happens during $\mathsf{MixSign}_{u_i}$.

MixSign. This algorithm is run between a user and the CA to compute a signature as done in Fig. 5. The first two moves (the user's ZKPoK and a ZKPoK of correct re-randomization) can be interleaved with those of the interactive signing, resulting in a protocol with four moves in total. Moreover, the proof of re-randomization (useful for voting schemes and further discussed on Sec. F) is optional and can be removed. The user's ZKPoK serves two purposes: proves plaintext knowledge and authenticates her. Since ElGamal can be decrypted with knowledge of the encryption randomness, a proof of plaintext knowledge is just a Schnorr proof on C_0. Such a proof is needed to avoid replay attacks where a user B waits for another user A to submit her ciphertext, randomize it and get a signature for the same message. The ephemeral key pair used by the CA is introduced to protect the scheme against maliciously crafted user keys. Without it, malicious users could sample their keys correlatedly and collude with the first mix server to replace ciphertexts. The share of the ephemeral public key ensures that each verification key is independent on how users sample their keys.

MixInit. This algorithm corresponds to the initialization phase in which every user submits their ciphertext $(C_0, C_1)_i$ alongside the corresponding signature σ_i on it and public keys uvk_i and evk_i. The corresponding verification key for $((C_0, C_1)_i, \sigma_i)$ is computed as $\mathsf{vk}_i := \mathsf{uvk}_i + \mathsf{evk}_i + \mathsf{avk}_i$. Upon verification of each tuple, the initial shuffle set is defined as $\mathcal{SS}\mathsf{et}^{(0)} := \{(C_0, C_1)_i, \sigma_i, \mathsf{vk}_i\}^{(0)}_{i\in[n]}$.

Mix. This algorithm is run in cascade by N mixers. The first one takes the initial shuffle set $\mathcal{SS}\mathsf{et}^{(0)}$ and does the following:

1. Computes the addition of all verification keys to obtain $\mathsf{VK}^{(0)} := \sum_{i=1}^{i=n} \mathsf{vk}_i^{(0)}$. Multiplies it by ρ to obtain $\mathsf{VK}^{(1)} := \sum_{i=1}^{i=n} \mathsf{vk}_i^{(1)}$ and proves knowledge of ρ in zero-knowledge with respect to $\mathsf{VK}^{(0)}$. As a result, it obtains a proof $\pi^{(1)}$.
2. Generates an aggregate signature for the message $m^{(1)} := \pi^{(1)}||\mathsf{VK}^{(1)}$. The purpose of this signature is to bind the mixer's output to the previous one. For the first mixer, a valid proof ensures the relation between the permuted and randomized verification keys with those in $\mathcal{SS}\mathsf{et}^{(0)}$.
3. Runs $\mathsf{MSoRC.Rndmz}$ and $\mathsf{MSoRC.Adapt}$ using ciphertext and verification key randomizers μ and ρ to consistently randomize $(C_0, C_1)_i^{(0)}$ and $\sigma_i^{(0)}$, obtaining a tuple $((C_0, C_1)_i^{(1)}, \sigma_i^{(1)})$ that verifies under $\mathsf{vk}_i^{(1)} := \rho \cdot \mathsf{vk}_i^{(0)}$.[6]
4. Permutes message tuples and computes a partial aggregate signature for $\pi^{(1)}$.

Subsequent mixers apply the above procedure, taking the output from the previous mixer as input.

MixVerify. This algorithm can be run by any external party to verify the output of the whole mixing process. In the following, we discuss the scenario in which verification takes an input linear in the number of mixers as presented in Fig. 7. On input the initial and final shuffle sets ($\mathcal{SS}\mathsf{et}^{(0)}$ and $\mathcal{SS}\mathsf{et}^{(N)}$), the final aggregate signature and messages $m^{(k)}_{k\in[N]}$, all proofs are verified in batch as in Sec. 4. The batch verification implicitly validates all mixing steps, ensuring each mixer contributed to the randomization process. If it fails, each proof can be verified independently to identify the misbehaving mixer. Since all proofs are signed, a false one provides non-repudiable evidence on the mixer's misbehavior. For this reason, during the signing process, each mixing server needs to verify the partial aggregate signature up to that point and abort if it receives and invalid one.

Constant Verification. Using the multi-signature from [BDN18] (see Appendix E for details), we can modify the above approach to remove the linear dependency on N at the cost of introducing another round of interaction as done in HPP20. These modifications are shown with Mix^* and $\mathsf{MixVerify}^*$ in Fig. 7 (enclosed in boxes to emphasize that they are optional).

6.3 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

[6] Since the mixer chooses the randomizer ρ, collusion between the user and the CA would be able to break the anonymity of $\mathsf{vk}_i^{(1)}$ and, thus, its corresponding tuple. However, such collusion is not considered in our model, and even if it were, it would not help identify other users' votes.

$\underline{\mathsf{MixInit}(\{(C_0,C_1)_i,\sigma_i,\mathsf{uvk}_i,\mathsf{evk}_i\}_{i\in[n]}):}$ // Each server verifies the initial set

$\forall\ i,j : \mathsf{uvk}_i + \mathsf{evk}_i \neq \mathsf{uvk}_j + \mathsf{evk}_j \wedge$ **foreach** $i \in [n]$ **do**

  check $\mathsf{MSoRC.Verify}(\mathsf{uvk}_i + \mathsf{evk}_i + \mathsf{avk}, (C_0,C_1)_i, \sigma_i)$

$\underline{\mathsf{Mix}(\mathsf{ssk}_j, \{((C_0,C_1)_i,\sigma_i,\mathsf{vk}_i)\}^{(j-1)}_{i\in[n]}, \sigma^{(j-1)}, (\mathsf{spk}_k, m_k)_{k\in[j-1]}):}$

$\mu,\rho \leftarrow\!\!\$\ \mathbb{Z}_p^*; \pi^{(j)} \leftarrow \mathsf{NIZK.Prove}(\sum_{i=1}^{i=n}\mathsf{vk}_i^{(j-1)}, \sum_{i=1}^{i=n}\mathsf{vk}_i^{(j)}, \rho)$

$\mathsf{vk}_i^{(j)} \leftarrow \rho\cdot\mathsf{vk}_i^{(j-1)}; m_j := \pi^{(j)}\ ||\ \sum_{i=1}^{i=n}\mathsf{vk}_i^{(j)}$

if $j = 1$ **then** $\sigma^{(j)} \leftarrow \mathsf{SAS.Sign}(\mathsf{ssk}_j, \perp, \perp, m_j)$

// Each mixer verifies the aggregate signature from previous mixers

if $j > 1$ **then** $\sigma^{(j)} \leftarrow \mathsf{SAS.Sign}(\mathsf{ssk}_j, \sigma^{(j-1)}, (\mathsf{spk}_k, m_k)_{k\in[j-1]}, m_j)$

foreach $i \in [n]$ **do**

  $(C_0,C_1)_i^{(j)} \leftarrow \mathsf{MSoRC.Rndmz}(\mathsf{ek}, (C_0,C_1)_i^{(j-1)}; \mu)$

  $\sigma_i^{(j)} \leftarrow \mathsf{MSoRC.Adapt}(\sigma_i^{(j-1)}; \mu, \rho)$

$\{(C_0,C_1)_{\Pi(i)}, \sigma_{\Pi(i)}, \mathsf{vk}_{\Pi(i)}\}^{(j)}_{i\in[n]} \leftarrow\!\!\$\ \{(C_0,C_1)_i,\sigma_i,\mathsf{vk}_i\}^{(j)}_{i\in[n]}$

return $(\{(C_0,C_1)_{\Pi(i)}, \sigma_{\Pi(i)}, \mathsf{vk}_{\Pi(i)}\}^{(j)}_{i\in[n]}, \pi^{(j)}, \sigma^{(j)})$

$\underline{\mathsf{MixVerify}(\{\mathsf{spk}_k\}_{k\in[N]}, \{((C_0,C_1)_i,\sigma_i,\mathsf{vk}_i)\}^{(0)}_{i\in[n]}, \{((C_0,C_1)_i,\sigma_i,\mathsf{vk}_i)\}^{(N)}_{i\in[n]},}$

$\underline{\pi^{(k)}_{k\in[N]}, \{\sum_{i=1}^{i=n}\mathsf{vk}_i^{(k)}\}_{k\in[1..N-1]}, \sigma^{(N)}):}$

check $\mathsf{NIZK.Verify}(\pi^{(k)}_{k\in[N]}, \{\sum_{i=1}^{i=n}\mathsf{vk}_i^{(k)}\}_{k\in[0..N]})$

check $\mathsf{SAS.Verify}(\{\mathsf{spk}_k\}_{k\in[N]}, \{\pi^{(k)}\ ||\ \sum_{i=1}^{i=n}\mathsf{vk}_i^{(k)}\}_{k\in[N]}, \sigma^{(N)})$

foreach $i \in [n]$ **check** $\mathsf{MSoRC.Verify}(\mathsf{vk}_i^{(N)}, (C_0,C_1)_i^{(N)}, \sigma_i^{(N)})$

$\boxed{\mathsf{Mix}^*(\mathsf{msk}_i, \{\mathsf{mpk}_1,\ldots,\mathsf{mpk}_N\}, \pi^{(N)}\ ||\ \sum_{i=1}^{i=n}\mathsf{vk}_i^{(N)}):}$

return $\mathsf{MSig.Sign}(\mathsf{msk}_i, \{\mathsf{mpk}_1,\ldots,\mathsf{mpk}_N\}, \pi^{(N)}\ ||\ \sum_{i=1}^{i=n}\mathsf{vk}_i^{(N)})$

//Any combiner computes $\mathsf{msig} = \sum \mathcal{H}_1(\mathsf{pk}_i, \{\mathsf{pk}_1,\ldots,\mathsf{pk}_N\})\sigma_i$

$\boxed{\mathsf{MixVerify}^*(\mathsf{avk}, \{((C_0,C_1)_i,\sigma_i,\mathsf{vk}_i)\}^{(0)}_{i\in[n]}, \{((C_0,C_1)_i,\sigma_i,\mathsf{vk}_i)\}^{(N)}_{i\in[n]}, \pi^{(N)}, \mathsf{msig}):}$

check $\mathsf{NIZK.Verify}(\pi^{(N)}, \sum_{i=1}^{i=n}\mathsf{vk}_i^{(N)}) \wedge \mathsf{MSig.Verify}(\mathsf{avk}, \pi^{(N)}\ ||\ \sum_{i=1}^{i=n}\mathsf{vk}_i^{(N)}, \mathsf{msig})$

foreach $i \in [n]$ **check** $\mathsf{MSoRC.Verify}(\mathsf{vk}_i^{(N)}, (C_0,C_1)_i^{(N)}, \sigma_i^{(N)})$

Fig. 7. Algorithms MixInit, Mix, MixVerify, Mix^* and $\mathsf{MixVerify}^*$.

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 collude with some mixers. Both notions hold if at least one mix server is honest.

Definition 12 (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 the (honest and corrupted) users that will generate a message*
- *proves knowledge of the secrete 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 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 $\mathsf{MixVerify}(\mathcal{SS}\mathsf{et}, \mathcal{SS}\mathsf{et}', \mathsf{proof}') = 1$ *but* $\{\mathsf{Dec}^*(\mathcal{SS}\mathsf{et})\} \neq \{\mathsf{Dec}^*(\mathcal{SS}\mathsf{et}')\}$, *where* Dec^* *extracts the plaintexts using the decryption key.*

Theorem 13 (Soundness). *Our* Mixnet *scheme is sound in the certified key setting assuming the unforgeability of our* MSoRC *scheme and the* kerMDH *assumption.*

Proof Sketch. We first note that if the verification passes, soundness of the NIZK proof guarantees (under the kerMDH assumption) that $\forall\ \mathsf{vk}'_i \in \mathcal{SS}\mathsf{et}' \wedge \mathsf{vk}_i \in \mathcal{SS}\mathsf{et} : \sum \mathsf{vk}'_i = \sum \alpha \mathsf{vk}_i$. 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 $\mathsf{esk}_i + \mathsf{ask}$ "fixes" the corresponding (and unique) equivalence class, and that is outside the adversary's control. 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, which concludes the proof.

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.

Definition 14 (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 mix-servers $\mathcal{J}^*$ and generates itself their keys $(\mathsf{spk}_i)_{i\in\mathcal{J}^*}$*
- *decides on the set $\mathcal{I}$ of the (honest and corrupted) users that will generate a message*
- *decides on the set $\mathcal{J}$ of the (honest and corrupted) mix-servers that will make mixes*
- *proves its knowledge of the secrete 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 message tuples $(\mathcal{T}_i)_{i\in\mathcal{I}^*}$ for corrupted users*
3. *The challenger generates the keys of the honest mix-servers $(\mathsf{ssk}_j, \mathsf{spk}_j)_{j\in\mathcal{J}\setminus\mathcal{J}^*}$ 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 shuffle set is thus defined by $\mathcal{SS}\mathsf{et}^{(0)} = (\mathcal{T}_i)_{i\in\mathcal{I}}$. The challenger randomly chooses a bit $b \leftarrow\!\!\$\ \{0,1\}$ and then enters into a loop for $j \in \mathcal{J}$ with the attacker:

- *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 15 (Privacy). *Our* Mixnet *scheme 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 on 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 each $\mathsf{vk}_i \in \mathcal{SS}\mathsf{et}^{(j-1)}$ with ρ to get $\mathsf{vk}_i^{(j-1)}$. The public key unlinkability of MSoRC guarantees that $\mathsf{vk}_i^{(j-1)}$ is unlinkable to the adversary (even if it knows the user's secret key and any previous randomizer from a corrupted mixer).
2. randomizes each $(C_0, C_1)_i^{(j-1)}$ with μ and adapts $\Sigma_i^{(j-1)}$ with μ and ρ 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 $(C_0, C_1)_i^{(j)}$. 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 $\Sigma_i^{(j-1)}$.

6.4 Extensions

As discussed in Appendix. F (where we present a detailed discussion concerning the application of 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:

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

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.

7 Evaluation

In this section, we first compare the complexity our work with state-of-the-art mixnets constructions. Subsequently we present experimental results of our protocol's implementation.

Comparison. We compare our mixnet with the works by Hébant *et al.* [HPP20] and Faonio and Russo [FR22] in Table 1[7]. 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 specify which signature the servers use to sign their proofs and so 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 optimized one, which has constant costs. In the standard one, the mixers do not need to verify individual proofs, but they need to verify the partial aggregate signature. 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. Regarding the optimized case, we include the cost of verifying each individual proof and the final aggregate signature as part of the mixing process just as we do for HPP20. 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 $\mathbb{Z}_p$).

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, 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 Grot-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 approaches in Table 1, where we consider the popular 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 $\mathbb{G}_1$ (E_1) and $\mathbb{G}_2$ (E_2), the exponentiation in group $\mathbb{G}_T$ (E_T) as well as pairing computation (P), 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 $\mathbb{G}_1$. Firstly, we observe that HPP20 and our approach only linearly depend on the paramters n and N. In contrast [FR22] have a dependcency on $n \cdot N$ in the verification costs and generally higher computational and bandwidth costs overall. When taking a closer comparison

[7] 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 this property, achieving more efficient constructions.

Table 1. Comparison of mixnet approaches.* denotes the use of Mix^* and $\mathsf{MixVerify}^*$ in our scheme to optimize verification.

Scheme	**Mixing**	
	Comp.	
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*	$(6n+5)E_1 + (2n+N+2)E_2 + (N+6)P$	
Ours	$(6n+5)E_1 + (2n+N+2)E_2 + 2P$	
	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*	$(4n+N+2)\mathbb{G}_1 + (4n+6N)\mathbb{G}_2$	$(4n+3)\mathbb{G}_1 + (4n+3)\mathbb{G}_2$
Ours	$(4n+N+2)\mathbb{G}_1 + (4n+5N)\mathbb{G}_2$	$(4n+3)\mathbb{G}_1 + (4n+2)\mathbb{G}_2$
	Verification	
	Comp.	
Rand-RCCA [FR22]	$(6N(n+1)-6n)E_1 + (6N+4nN)E_2 - 4nE_2 + 4NE_T + 4n(N-1)P$	
HPP20 [HPP20]	$(8n+14)P$	
Ours*	$(14n+5)P$	
Ours	$(14n+N+3)P$	
	Comm.	
Rand-RCCA [FR22]	$(16n+4)\mathbb{G}_1 + 12n\mathbb{G}_2 + 2n\mathbb{G}_T$	
HPP20 [HPP20]	$(12n+4)\mathbb{G}_1 + (14n+7)\mathbb{G}_2$	
Ours*	$(10n+1)\mathbb{G}_1 + (8n+3)\mathbb{G}_2$	
Ours	$(10n+2N+1)\mathbb{G}_1 + (8n+7N+1)\mathbb{G}_2$	

Table 2. Running times of each protocol in seconds.

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

with the more scaleable solution due to 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, when setting $n = 1000$ and $N = 10$, mixing is around $3.5x$ more efficient with our approach and our 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 upon request. 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. The (interactive) signing protocol of our MSoRC scheme (Fig. 5) takes 6.4ms while MixSign (which includes the ZKPoK's) takes 8.1ms.

Running times of other protocols are summarized in Table 2, confirming the linear complexity of our mixnet scheme. In all cases, the standard deviation was below 1s. In this regard, we recall that the main difference in terms of computation between MixVerify and MixVerify* is on the number of N pairings and multi-exponentiation executed. A paring takes around 380 microseconds while a multi-exponentiation for $N = 10$ takes 737 microseconds. Thus, for a small N, their difference in the running times is less noticeable compared to others as shown in the table. We omit Mix* as it's a single signature computation but recall that in that case, each mixer runs MixVerify before Mix* and one gets higher overall running times for the mixing process.

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.

8 Conclusion

We developed the notion of MSoRC as a combination of threshold mercurial signatures and signatures on randomizable ciphertexts. We presented a concrete instantiation with an optimized variant that fits naturally as the core building block for the scalable mixnet framework of HPP20 [HPP20].

Our improvements over HPP20 are twofold. From the efficiency point of view, substituting GS proofs and incorporating aggregate signatures, we obtain an even more efficient, scalable mixnet protocol. This is demonstrated by our benchmarks on both verification strategies. In addition, public-key unlinkability of our MSoRC scheme is the cornerstone for our stronger security that withstands collusion between users and mix servers, or mix servers and the certificate authority while the users are assumed honest in previous work. As a result, our mixnet suits more practical e-voting where individual voters are not fully trustful. Scalability of our mixnet is supported by an implementation. In this regard, for 50k voters and 10 mix servers the worst-case time for mixing takes around 40 seconds and the verification of the whole mixing process (input validation plus out verification) takes less than 5 minutes on a commodity laptop, without any parallelization technique. We also stress the modular design of our approach that allows for a smoother integration of the required building blocks.

References

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

Abe99. M. Abe. Mix-networks on permutation networks. In *ASIACRYPT'99*, *LNCS* 1716, pages 258–273. Springer, Heidelberg, November 1999.

ABGS23. D. F. Aranha, C. Baum, K. Gjøsteen, and T. 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. D. Aranha, M. Battagliola, and L. 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. B. Adida. Helios: Web-based open-audit voting. In *USENIX Security 2008*, pages 335–348. USENIX Association, July / August 2008.

AF04. M. Abe and S. Fehr. Adaptively secure feldman VSS and applications to universally-composable threshold cryptography. In *Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, *Lecture Notes in Computer Science* 3152, pages 317–334. Springer, 2004.

AFG+10. M. Abe, G. Fuchsbauer, J. Groth, K. Haralambiev, and M. Ohkubo. Structure-preserving signatures and commitments to group elements. In *CRYPTO 2010*, *LNCS* 6223, pages 209–236. Springer, Heidelberg, August 2010.

AFK+20. A. Aggelakis, P. Fauzi, G. Korfiatis, P. Louridas, F. Mergoupis-Anagnou, J. Siim, and M. Zajac. A non-interactive shuffle argument with low trust assumptions. In *CT-RSA 2020*, *LNCS* 12006, pages 667–692. Springer, Heidelberg, February 2020.

AGHO11. M. Abe, J. Groth, K. Haralambiev, and M. Ohkubo. Optimal structure-preserving signatures in asymmetric bilinear groups. In *CRYPTO 2011*, *LNCS* 6841, pages 649–666. Springer, Heidelberg, August 2011.

AH01. M. Abe and F. Hoshino. Remarks on mix-network based on permutation networks. In *PKC 2001*, *LNCS* 1992, pages 317–324. Springer, Heidelberg, February 2001.

AKTZ17. N. Alexopoulos, A. Kiayias, R. Talviste, and T. Zacharias. MCMix: Anonymous messaging via secure multiparty computation. In *USENIX Security 2017*, pages 1217–1234. USENIX Association, August 2017.

ANPT24. M. Abe, M. Nanri, O. Perez Kempner, and M. Tibouchi. Interactive threshold mercurial signatures and applications. Cryptology ePrint Archive, Paper 2024/625, 2024.

BCG+15. E. Ben-Sasson, A. Chiesa, M. Green, E. Tromer, and M. Virza. Secure sampling of public parameters for succinct zero knowledge proofs. In *2015 IEEE Symposium on Security and Privacy*, pages 287–304. IEEE Computer Society Press, May 2015.

BDN18. D. Boneh, M. Drijvers, and G. Neven. Compact multi-signatures for smaller blockchains. In *ASIACRYPT 2018, Part II*, *LNCS* 11273, pages 435–464. Springer, Heidelberg, December 2018.

BF20. B. Bauer and G. Fuchsbauer. Efficient signatures on randomizable ciphertexts. In *SCN 20*, *LNCS* 12238, pages 359–381. Springer, Heidelberg, September 2020.

BFPV11. O. Blazy, G. Fuchsbauer, D. Pointcheval, and D. Vergnaud. Signatures on randomizable ciphertexts. In *PKC 2011*, *LNCS* 6571, pages 403–422. Springer, Heidelberg, March 2011.

BG02. D. Boneh and P. Golle. Almost entirely correct mixing with applications to voting. In *ACM CCS 2002*, pages 68–77. ACM Press, November 2002.

BGR98. M. Bellare, J. A. Garay, and T. Rabin. Fast batch verification for modular exponentiation and digital signatures. In *EUROCRYPT'98*, *LNCS* 1403, pages 236–250. Springer, Heidelberg, May / June 1998.

BGR12. S. Bursuc, G. S. Grewal, and M. D. Ryan. Trivitas: Voters directly verifying votes. In *E-Voting and Identity*, pages 190–207, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

BHKS18. M. Backes, L. Hanzlik, K. Kluczniak, and J. Schneider. Signatures with flexible public key: Introducing equivalence classes for public keys. In *ASIACRYPT 2018, Part II*, *LNCS* 11273, pages 405–434. Springer, Heidelberg, December 2018.

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

CAB+15. C. Chen, D. E. Asoni, D. Barrera, G. Danezis, and A. Perrig. HORNET: High-speed onion routing at the network layer. In *ACM CCS 2015*, pages 1441–1454. ACM Press, October 2015.

CCC+22a. D. Chaum, R. Carback, J. Clark, C. Liu, M. Nejadgholi, B. Preneel, A. T. Sherman, M. Yaksetig, Z. Yin, F. Zagórski, and B. Zhang. Votexx: A solution to improper influence in voter-verifiable elections. *E-Vote-ID 2022.*, 2022.

CCC+22b. D. Chaum, R. Carback, J. Clark, C. Liu, M. Nejadgholi, B. Preneel, A. T. Sherman, M. Yaksetig, Z. Yin, F. Zagórski, and B. Zhang. Votexx: A solution to improper influence in voter-verifiable elections. *IACR Cryptol. ePrint Arch.*, page 1212, 2022.

CCM08. M. R. Clarkson, S. Chong, and A. 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. R. Cramer, M. K. Franklin, B. Schoenmakers, and M. Yung. Multi-autority secret-ballot elections with linear work. In *EUROCRYPT'96*, *LNCS* 1070, pages 72–83. Springer, Heidelberg, May 1996.

CGGI13. V. Cortier, D. Galindo, S. Glondu, and M. 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. V. Cortier, P. Gaudry, and Q. 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.

CH11. J. Clark and U. Hengartner. Selections: Internet voting with over-the-shoulder coercion-resistance. In *Financial Cryptography and Data Security - 15th International Conference, FC 2011, Gros Islet, St. Lucia, February 28 - March 4, 2011, Revised Selected Papers*, *Lecture Notes in Computer Science* 7035, pages 47–61. Springer, 2011.

CH20. G. Couteau and D. Hartmann. Shorter non-interactive zero-knowledge arguments and ZAPs for algebraic languages. In *CRYPTO 2020, Part III*, *LNCS* 12172, pages 768–798. Springer, Heidelberg, August 2020.

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

CKLM13. M. Chase, M. Kohlweiss, A. Lysyanskaya, and S. Meiklejohn. Verifiable elections that scale for free. In *PKC 2013*, *LNCS* 7778, pages 479–496. Springer, Heidelberg, February / March 2013.

CKN03. R. Canetti, H. Krawczyk, and J. B. Nielsen. Relaxing chosen-ciphertext security. In *CRYPTO 2003*, *LNCS* 2729, pages 565–582. Springer, Heidelberg, August 2003.

CL19. E. C. Crites and A. Lysyanskaya. Delegatable anonymous credentials from mercurial signatures. In *CT-RSA 2019*, *LNCS* 11405, pages 535–555. Springer, Heidelberg, March 2019.

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

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

CLPK22. A. Connolly, P. Lafourcade, and O. Perez-Kempner. Improved constructions of anonymous credentials from structure-preserving signatures on equivalence classes. In *PKC 2022, Part I*, *LNCS* 13177, pages 409–438. Springer, Heidelberg, March 2022.

CP93. D. Chaum and T. P. Pedersen. Wallet databases with observers. In *Advances in Cryptology — CRYPTO' 92*, pages 89–105, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

CRS05. D. Chaum, P. Y. A. Ryan, and S. A. Schneider. A practical voter-verifiable election scheme. In *ESORICS 2005*, *LNCS* 3679, pages 118–139. Springer, Heidelberg, September 2005.

CS11. V. Cortier and B. Smyth. Attacking and fixing helios: An analysis of ballot secrecy. In *CSF 2011 Computer Security Foundations Symposium*, pages 297–311. IEEE Computer Society Press, 2011.

DEF+23. B. David, F. Engelmann, T. Frederiksen, M. Kohlweiss, E. Pagnin, and M. Volkhov. Updatable privacy-preserving blueprints. Cryptology ePrint Archive, Paper 2023/1787, 2023.

DHK21. C. Diaz, H. Halpin, and A. Kiayias. The nym network: The next generation of privacy infrastructure. Online, 2021.

ElG86. T. ElGamal. On computing logarithms over finite fields. In *CRYPTO'85*, *LNCS* 218, pages 396–402. Springer, Heidelberg, August 1986.

FFHR19. A. Faonio, D. Fiore, J. Herranz, and C. Ràfols. Structure-preserving and re-randomizable RCCA-secure public key encryption and its applications. In *ASIACRYPT 2019, Part III*, *LNCS* 11923, pages 159–190. Springer, Heidelberg, December 2019.

FGHP09. A. L. Ferrara, M. Green, S. Hohenberger, and M. Ø. Pedersen. Practical short signature batch verification. In *CT-RSA 2009*, *LNCS* 5473, pages 309–324. Springer, Heidelberg, April 2009.

FHR23. A. Faonio, D. Hofheinz, and L. Russo. Almost tightly-secure re-randomizable and replayable CCA-secure public key encryption. In *PKC 2023, Part II*, *LNCS* 13941, pages 275–305. Springer, Heidelberg, May 2023.

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

FLSZ17. P. Fauzi, H. Lipmaa, J. Siim, and M. Zajac. An efficient pairing-based shuffle argument. In *ASIACRYPT 2017, Part II*, *LNCS* 10625, pages 97–127. Springer, Heidelberg, December 2017.

FR22. A. Faonio and L. Russo. Mix-nets from re-randomizable and replayable cca-secure public-key encryption. In *Security and Cryptography for Networks*, pages 172–196, Cham, 2022. Springer International Publishing.

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

FS01. J. Furukawa and K. Sako. An efficient scheme for proving a shuffle. In *CRYPTO 2001*, *LNCS* 2139, pages 368–387. Springer, Heidelberg, August 2001.

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

Gro03. J. Groth. A verifiable secret shuffle of homomorphic encryptions. In *PKC 2003*, *LNCS* 2567, pages 145–160. Springer, Heidelberg, January 2003.

GS08. J. Groth and A. Sahai. Efficient non-interactive proof systems for bilinear groups. In *EUROCRYPT 2008*, *LNCS* 4965, pages 415–432. Springer, Heidelberg, April 2008.

GS12. J. Groth and A. Sahai. Efficient noninteractive proof systems for bilinear groups. *SIAM J. Comput.*, 41(5):1193–1232, 2012.

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

HMMP23a. T. Haines, R. Mosaheb, J. Müller, and I. Pryvalov. SoK: Secure E-voting with everlasting privacy. *PoPETs*, 2023(1):279–293, January 2023.

HMMP23b. T. Haines, R. Mosaheb, J. Müller, and I. Pryvalov. Sok: Secure e-voting with everlasting privacy. *Proc. Priv. Enhancing Technol.*, 2023(1):279–293, 2023.

HMQA23. T. Haines, J. Müller, and I. n. 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.

HPP20. C. Hébant, D. H. Phan, and D. Pointcheval. Linearly-homomorphic signatures and scalable mix-nets. In *PKC 2020, Part II*, *LNCS* 12111, pages 597–627. Springer, Heidelberg, May 2020.

HS14. C. Hanser and D. Slamanig. Structure-preserving signatures on equivalence classes and their application to anonymous credentials. In *ASIACRYPT 2014, Part I*, *LNCS* 8873, pages 491–511. Springer, Heidelberg, December 2014.

JCJ05. A. Juels, D. Catalano, and M. 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.

JJR02. M. Jakobsson, A. Juels, and R. L. Rivest. Making mix nets robust for electronic voting by randomized partial checking. In *USENIX Security 2002*, pages 339–353. USENIX Association, August 2002.

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

KCGDF17. A. Kwon, H. Corrigan-Gibbs, S. Devadas, and B. Ford. Atom: Horizontally scaling strong anonymity. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, page 406–422, New York, NY, USA, 2017. Association for Computing Machinery.

KER+22. C. Killer, M. Eck, B. Rodrigues, J. von der Assen, R. Staubli, and B. 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.

KL21. T. Krips and H. Lipmaa. More efficient shuffle argument from unique factorization. In *CT-RSA 2021*, *LNCS* 12704, pages 252–275. Springer, Heidelberg, May 2021.

KMW12. S. Khazaei, T. Moran, and D. Wikström. A mix-net from any CCA2 secure cryptosystem. In *ASIACRYPT 2012*, *LNCS* 7658, pages 607–625. Springer, Heidelberg, December 2012.

KW15. E. Kiltz and H. Wee. Quasi-adaptive NIZK for linear subspaces revisited. In *EUROCRYPT 2015, Part II*, *LNCS* 9057, pages 101–128. Springer, Heidelberg, April 2015.

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

LOS+06. S. Lu, R. Ostrovsky, A. Sahai, H. Shacham, and B. Waters. Sequential aggregate signatures and multisignatures without random oracles. In *EUROCRYPT 2006*, *LNCS* 4004, pages 465–485. Springer, Heidelberg, May / June 2006.

LPJY13. B. Libert, T. Peters, M. Joye, and M. Yung. Linearly homomorphic structure-preserving signatures and their applications. In *CRYPTO 2013, Part II*, *LNCS* 8043, pages 289–307. Springer, Heidelberg, August 2013.

LQT20. W. Lueks, I. Querejeta-Azurmendi, and C. Troncoso. VoteAgain: A scalable coercion-resistant voting system. In *USENIX Security 2020*, pages 1553–1570. USENIX Association, August 2020.

MMR22. D. Mestel, J. Müller, and P. 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.

MN07. T. Moran and M. Naor. Split-ballot voting: everlasting privacy with distributed trust. In *ACM CCS 2007*, pages 246–255. ACM Press, October 2007.

MRV16. P. Morillo, C. Ràfols, and J. L. Villar. The kernel matrix Diffie-Hellman assumption. In *ASIACRYPT 2016, Part I*, *LNCS* 10031, pages 729–758. Springer, Heidelberg, December 2016.

Nef01. C. A. Neff. A verifiable secret shuffle and its application to e-voting. In *ACM CCS 2001*, pages 116–125. ACM Press, November 2001.

OANWO20. J. O'Connor, J.-P. Aumasson, S. Neves, and Z. Wilcox-O'Hearn. Blake3. Online, 2020.

Oka97. T. Okamoto. Receipt-free electronic voting schemes for large scale elections. In *Security Protocols, 5th International Workshop*, *LNCS* 1361, pages 25–35, Paris, France, April 7–9 1997. Springer, Heidelberg.

PB09. K. Peng and F. Bao. A design of secure preferential e-voting. In *E-Voting and Identity, Second International Conference, VoteID 2009, Luxembourg, September 7-8, 2009. Proceedings*, *Lecture Notes in Computer Science* 5767, pages 141–156. Springer, 2009.

Ped91. T. P. Pedersen. A threshold cryptosystem without a trusted party (extended abstract) (rump session). In *EUROCRYPT'91*, *LNCS* 547, pages 522–526. Springer, Heidelberg, April 1991.

PHE+17. A. M. Piotrowska, J. Hayes, T. Elahi, S. Meiser, and G. Danezis. The loopix anonymity system. In *USENIX Security 2017*, pages 1199–1216. USENIX Association, August 2017.

PS16. D. Pointcheval and O. Sanders. Short randomizable signatures. In *CT-RSA 2016*, *LNCS* 9610, pages 111–126. Springer, Heidelberg, February / March 2016.

PS18. D. Pointcheval and O. Sanders. Reassessing security of randomizable signatures. In *CT-RSA 2018*, *LNCS* 10808, 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. K. Sako and J. Kilian. Receipt-free mix-type voting scheme - a practical solution to the implementation of a voting booth. In *EUROCRYPT'95*, *LNCS* 921, pages 393–403. Springer, Heidelberg, May 1995.

TDE17. R. R. Toledo, G. Danezis, and I. Echizen. Mix-oram: Using delegated shuffles. In *Proceedings of the 2017 on Workshop on Privacy in the Electronic Society*, WPES '17, page 51–61, New York, NY, USA, 2017. Association for Computing Machinery.

WB09. R. Wen and R. Buckland. Masked ballot voting for receipt-free online elections. In *E-Voting and Identity, Second International Conference, VoteID 2009, Luxembourg, September 7-8, 2009. Proceedings*, *Lecture Notes in Computer Science* 5767, pages 18–36. Springer, 2009.

Yan23. Q. 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 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 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 MixVerify.

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 16 (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 secrete keys to get the certifications Σ_i on vk_i for $i \in \mathcal{I}^*$*
   - *decides on the set $\mathcal{I}$ of the (honest and corrupted) users that will generate a 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 $\mathsf{MixVerify}(\mathcal{SS}\mathsf{et}, \mathcal{SS}\mathsf{et}', \mathsf{proof}') = 1$ *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 17 (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 secrete keys to get the certifications* Σ_i *on* vk_i *for* $i \in \mathcal{I}^*$
- *decides on the corrupted mix-servers* $\mathcal{J}^*$ *and generates itself their keys*
- *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 $\mathcal{S}\mathsf{Set} = (\mathcal{T}_i)_{i\in\mathcal{I}}$*. 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* $\mathcal{S}\mathsf{Set}^{(j-1)}$
- *if* $j \in \mathcal{J}^*$*,* $\mathcal{A}$ *builds itself the new shuffle set* $\mathcal{S}\mathsf{Set}^{(j)}$ *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* $(\mathcal{S}\mathsf{Set}^{(j)}, \mathsf{proof}^{(j)})$

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 Section 6.1 from HPP20. However, such fix still allows *replay attacks* [CS11] that should also be avoided in voting applications.

B Couteau & Hartmann's Proof System

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

- $\mathsf{NIZK.CRSGen}(1^\kappa)$: $\mathsf{pp} \leftarrow\!\!\$\ \mathsf{BGGen}(1^\kappa)$; $z \leftarrow\!\!\$\ \mathbb{Z}_p$; $\tau \leftarrow z$; $Z \leftarrow zG$; $\mathsf{crs} \leftarrow (\mathsf{pp}, Z)$; **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 $e(\mathsf{d}, \mathsf{A}_0) = e(Z, \mathsf{x}_0) + e(G, \mathsf{a}_0) \wedge e(\mathsf{d}, \mathsf{A}_1)$
  $= e(Z, \mathsf{x}_1) + e(G, \mathsf{a}_1) \wedge\ e(\mathsf{d}, \mathsf{A}_2) = e(Z, \mathsf{x}_2) + e(G, \mathsf{a}_2)$

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

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

C Security Proofs

Correctness. In the following, we argue that the interactive variant of our MSoRC scheme produces signatures under the same distribution. Looking closer at how Z and T are computed, we have:

$$\begin{aligned} Z &= \frac{1}{s_1}(Z_0 - rG) = \frac{1}{s_1}\left(\frac{1}{s_0}\Big(Z_1 + x_0^0 C_0 + x_1^0 C_1 + x_2^0 G\Big) - rG\right) \\ &= \frac{1}{s_1}\left(\frac{1}{s_0}\Big(\Big(rS_0 + x_0^1 C_0 + x_1^1 C_1 + x_2^1 G\Big) + x_0^0 C_0 + x_1^0 C_1 + x_2^0 G\Big) - rG\right) \\ &= \frac{1}{s_1}\left(\frac{1}{s_0}\Big(rs_0 G + \Big(x_0^0 + x_0^1\Big) C_0 + \Big(x_1^0 + x_1^1\Big) C_1 + \Big(x_2^0 + x_2^1\Big) G\Big) - rG\right) \\ &= \frac{1}{s_0 s_1}\Big(\Big(x_0^0 + x_0^1\Big) C_0 + \Big(x_1^0 + x_1^1\Big) C_1 + \Big(x_2^0 + x_2^1\Big) G\Big) \end{aligned}$$

Similarly, T is computed as:

$$\begin{aligned} T &= \frac{1}{s_1}(T_0 - rG) = \frac{1}{s_1}\left(\frac{1}{s_0}\Big(T_1 + x_0^0 G + x_1^0 X\Big) - rG\right) \\ &= \frac{1}{s_1}\left(\frac{1}{s_0}\Big(\Big(rS_0 + x_0^1 G + x_1^1 X\Big) + x_0^0 G + x_1^0 X - rG\Big)\right) \\ &= \frac{1}{s_0 s_1}\Big(\Big(x_0^0 + x_0^1\Big) G + \Big(x_1^0 + x_1^1\Big) X\Big) + \frac{1}{s_1}\Big(\frac{1}{s_0} r s_0 G - rG\Big) \\ &= \frac{1}{s_0 s_1}\Big(\Big(x_0^0 + x_0^1\Big) G + \Big(x_1^0 + x_1^1\Big) X\Big) \end{aligned}$$

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.

Theorem 1 (Unforgeability). *Our base* MSoRC *is unforgeable in the GGM w.r.t. Definition 2 if all* ZKPoK*'s are secure.*

Proof. We begin considering an adversary $\mathcal{A}$ against the unforgeability game from Def. 2, which makes use of the standard (single party) signing protocol. Subsequently, we construct a simulator that, given access to an adversary $\mathcal{A}'$ in the unforgeability game from Def. 3 (considering adversarial encryption keys), plays the role of an adversary against the unforgeability game from Def. 2. We then show that the simulator wins whenever $\mathcal{A}'$ wins. However, as we previously proved that no such adversary $\mathcal{A}'$ can successfully produce a forgery, we conclude that no adversary can exists against Def. 3, which is our goal.

We reduce the security of our base scheme (Def. 2) to that of [BF20]. Thus, 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

$$
\begin{array}{lcl}
\underline{\mathsf{P}_0\colon \mathsf{sk}_0, \mathsf{pk}_0, \mathsf{pk}_1, (C_0, C_1)} & & \underline{\mathsf{P}_1\colon \mathsf{pk}_0, \mathsf{pk}_1, (C_0, C_1)} \\
 & & (Z', S', \hat{S}', T') \leftarrow \mathsf{Sign}(\mathsf{sk}, (C_0, C_1)) \\
(S_0, \hat{S}_0, \pi_0) \leftarrow \mathcal{A}(\mathsf{st}) & \xrightarrow{S_0, \hat{S}_0, \pi_0} & Z_1 \leftarrow\!\!\$\; \mathbb{G}_1; S \leftarrow S'; \hat{S} \leftarrow \hat{S}'; T \leftarrow\!\!\$\; \mathbb{G}_1 \\
 & \xleftarrow{T_1, Z_1, \pi_1} & \pi_1 \leftarrow \mathsf{ZKPoK.Sim}(T_1, Z_1, S_0, (C_0, C_1)) \\
(T_0, Z_0, \widetilde{\pi}_0) \leftarrow \mathcal{A}(\mathsf{st}, T_1, Z_1, \pi_1) & \xrightarrow{Z_0, T_0, \widetilde{\pi}_0} & Z \leftarrow Z'; T \leftarrow T' \\
 & & \pi_1 \leftarrow \mathsf{ZKPoK.Sim}(Z, Z_0, S_0, \hat{S}_0) \\
\mathbf{return}\ (\sigma, \pi_1) & \xleftarrow{\sigma, \widetilde{\pi}_1} & \sigma \leftarrow (Z, S, \hat{S}, T);\ \mathbf{return}\ (\sigma, \widetilde{\pi}_1)
\end{array}
$$

$$
\begin{array}{lcl}
\underline{\mathsf{P}_0\colon \mathsf{pk}_0, \mathsf{pk}_1, (C_0, C_1)} & & \underline{\mathsf{P}_1\colon \mathsf{sk}_1, \mathsf{pk}_0, \mathsf{pk}_1, (C_0, C_1)} \\
(Z', S', \hat{S}', T') \leftarrow \mathsf{Sign}(\mathsf{sk}, (C_0, C_1)) & & \\
S_0 \leftarrow S'; \hat{S}_0 \leftarrow \hat{S}' & & \\
\pi_0 \leftarrow \mathsf{ZKPoK.Sim}(S_0, \hat{S}_0) & \xrightarrow{S_0, \hat{S}_0, \pi_0} & (Z_1, \pi_1) \leftarrow \mathcal{A}(\mathsf{st}, S_0, \hat{S}_0, \pi_0) \\
 & \xleftarrow{T_1, Z_1, \pi_1} & \\
r \leftarrow \mathsf{ZKPoK.Ext}(\pi_1) & & \\
Z_0 \leftarrow Z'P^r; T_0 \leftarrow T'P^r & & \\
\widetilde{\pi}_0 \leftarrow \mathsf{ZKPoK.Sim}(\{Z_i, T_i, C_i\}_{i \in \{0,1\}}, Y_0) & \xrightarrow{Z_0, T_0, \widetilde{\pi}_0} & (\sigma, \widetilde{\pi}_1) \leftarrow \mathcal{A}(\mathsf{st}, Z_0, T_0, \pi_0) \\
\mathbf{return}\ (\sigma, \widetilde{\pi}_1) & \xleftarrow{\sigma, \widetilde{\pi}_1} & \mathbf{return}\ (\sigma, \widetilde{\pi}_1)
\end{array}
$$

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

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 Schwartz-Zippel lemma, such probability is $O(\frac{k}{p})$, which is negligible.

In the following, we switch our attention to the security of the interactive signing protocol from Fig. 5 as used in Def. 3.

For an adversary $\mathcal{A}'$ against the unforgeability game of Def. 3, we construct a simulator that, given access to $\mathcal{A}'$, plays the role of the adversary in the unforgeability game of Def. 2. The simulator gets pp and vk from the challenger. Subsequently, it 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 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 cases, the simulator outputs whatever $\mathcal{A}'$ outputs. Hence, whenever $\mathcal{A}'$ wins, the simulator wins. □

Theorem 18 (Unforgeability of our optimized MSoRC). *Our optimized scheme is unforgeable in the GGM under corruption if at most one party is corrupted and if all* ZKPoK*'s are secure.*

Proof. We consider an adversary $\mathcal{A}$ similar to that one against the unforgeability game from Def. 2. 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, 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:

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

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

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. $\hat{X}_i^* = \alpha \hat{X}_i, i \in \{0, 1, 2\}$[8]. 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:

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

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 $\mathbb{Z}_p(s_1, \ldots, s_k)[x_0, x_1, x_2, x]/(x_0, x_1, x_2, x) \equiv \mathbb{Z}_p(s_1, \ldots, s_k)$[9]. Over such quotient $z_i = 0$ and $t_i = 0$, and thus, (1) and (2) become:

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

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

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

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

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 $\forall i \in \{1, \dots, k\} : \tau = \zeta = 0$. 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:

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

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

We deduce $\tau_{z,j} = 0 \ \forall j \in \{1, \dots, k\}$. 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 $j_1 \neq j_2 \in \{1, \dots, k\}$ 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 $\forall i \in \{1, \dots, k\} \setminus \{j_0\} : \zeta_{z,j} = \tau_{t,j} = 0$.

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:

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. 9. ZKPoK protocol for π_0.

Prover: $Z, T, \hat{S}, Z_0, T_0, \hat{S}_0, r, s_1$		Verifier: $Z, T, \hat{S}, Z_0, T_0, \hat{S}_0$
$a_1, a_2 \leftarrow\!\$\ (\mathbb{Z}_p)^2; A_1 = a_1 G + a_2 T$		
$A_2 = a_1 G + a_2 Z; \hat{A}_4 = a_2 \hat{S}_0$	$\xrightarrow{A_1, A_2, \hat{A}_4}$	
$q_1 = a_1 - cr$	$\xleftarrow{c}$	$c \leftarrow\!\$\ \mathbb{Z}_p$
$q_2 = a_2 - c s_1$	$\xrightarrow{q_1, q_2}$	**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. 10. ZKPoK protocol for $\widetilde{\pi}_1$.

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

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]. □

D Zero-knowledge Proofs

We instantiate the ZKPoK of our interactive signing protocol in the ROM using known techniques [FS87, Sch91, CP93]. $\pi_0 :=$ (ZKPoK$[s_0 :\ S_0 = s_0 G \wedge \hat{S}_0 = s_0 \hat{G}]$) is shown in Fig.9. $\pi_1 :=$ (ZKPoK$[(r, x_0^1, x_1^1, x_2^1) : T_1 = r S_0 + x_0^1 G + x_1^1 X\ \wedge\ Z_1 = r S_0 + x_0^1 C_0 + x_1^1 C_1 + x_2^1 G\ \wedge\ \hat{X}_0^1 = x_0^1 \hat{G}\ \wedge\ X_1^1 = x_1^1 \hat{G}\ \wedge\ X_2^1 = x_2^1 \hat{G}]$) is shown in Fig.11. They are a simple application of standard ZKPoK. However, $\widetilde{\pi}_0 :=$ (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 $\widetilde{\pi}_1 :=$ (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 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 shownm in Fig. 12 and Fig. 10.

E Aggregate and Multi-signatures

We recall the sequential aggregate signature from [PS16].

$$
\begin{array}{lcl}
\text{Prover: } T_1, Z_1, \{\hat{X}^1_i, x^1_i\}_{i\in\{0..2\}}, & & \text{Verifier: } T_1, Z_1, \{\hat{X}^1_i\}_{i\in\{0..2\}}, \\
S_0, X, C_0, C_1, r & & S_0, X, C_0, C_1 \\
\hline
a_1, a_2, a_3, a_4 \leftarrow\!\!\$ \, (\mathbb{Z}_p)^4 & & \\
A_1 = a_1 S_0 + a_3 G + a_4 X & & \\
A_2 = a_1 S_0 + a_3 C_0 + a_4 C_1 + a_2 G & & \\
\hat{A}_3 = a_3\hat{G}; \hat{A}_4 = a_4\hat{G}; \hat{A}_5 = a_2\hat{G} & \xrightarrow{A_1, A_2, \hat{A}_3, \hat{A}_4, \hat{A}_5} & \\
q_1 = a_1 - cr; q_2 = a_2 - cx^1_2 & \xleftarrow{\quad c \quad} & c \leftarrow\!\!\$ \, \mathbb{Z}_p \\
q_3 = a_3 - cx^1_0; q_4 = a_4 - cx^1_1 & \xrightarrow{q_1, q_2, q_3, q_4} & \textbf{return } A_1 = q_1 S_0 + q_3 G + q_4 X + cT_1 \\
 & & \wedge\, A_2 = q_1 S_0 + q_3 C_0 + q_4 C_1 + q_2 G + cZ_1 \\
 & & \wedge\, \hat{A}_3 = q_3\hat{G} + c\hat{X}^1_0 \wedge \hat{A}_4 = q_4\hat{G} + c\hat{X}^1_1 \\
 & & \wedge\, \hat{A}_5 = q_2\hat{G} + c\hat{X}^1_2
\end{array}
$$

Fig. 11. ZKPoK protocol for π_1.

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

Fig. 12. ZKPoK protocol for $\widetilde{\pi}_0$.

– $\mathsf{SAS.Setup}(1^\kappa)$: $\mathsf{pp} \leftarrow\!\!\$ \, \mathsf{BGGen}(1^\kappa)$; $w \leftarrow\!\!\$ \, \mathbb{Z}_p$;
  $W \leftarrow wG$; $\hat{W} \leftarrow w\hat{G}$; **return** $(\mathsf{pp}, W, \hat{W})$.
– $\mathsf{SAS.SKG}(\mathsf{pp})$: $\mathsf{sk} \leftarrow\!\!\$ \, \mathbb{Z}_p^*$; $\mathsf{pk} \leftarrow \mathsf{sk}\hat{G}$; **return** $(\mathsf{sk}, \mathsf{pk})$.
– $\mathsf{SAS.Sign}(\mathsf{sk}, \sigma, (m_1, \ldots, m_r), (\mathsf{pk}_1, \ldots, \mathsf{pk}_r), m)$:
  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'.
– $\mathsf{SAS.Verify}(\sigma, (m_1, \ldots, m_r), (\mathsf{pk}_1, \ldots, \mathsf{pk}_r))$:
  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 19. 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 19 (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 $\mathcal{H}_1 : \{0,1\}^* \to \mathbb{Z}_p$.

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

F Application to E-voting

As evidenced by the vast literature (see *e.g.,* [SK95, Abe98, Abe99, AH01, BG02, Adi08, CKLM13, LQT20, KER+22, ABGS23]), 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. $\mathsf{MixSetup}$ 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 $\mathsf{MixSign}$ 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 $\mathsf{MixSign}$, 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. 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 random oracle model that also suffices for the purpose.

Casting phase. Each voter casts their ballot $(\sigma_i, C_i', \mathsf{uvk}_i, \mathsf{evk}_i)$ on BB. Communication happens over a public channel, and the process is done only once.

Tallying phase. $\mathsf{MixInit}$ 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 $\mathsf{MixVerify}$ 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.

F.1 Security

Trust model. 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 $\mathsf{MixSign}$. The communication is done through an untappable channel or assumes the absolute absence of 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 $\mathsf{MixSign}$ concerning a forced vote $\widetilde{M}_i$ that is indistinguishable from the actual view. We recall $\mathsf{MixSign}$ to explain how u_i creates such a fake view.

1. Follow the first step of $\mathsf{MixSign}_{u_i}$ 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 $\mathsf{MSoRC.ISign}$.

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 $\mathsf{MixSign}$. If users are mandated to use a given vote $\widetilde{M}_i$, during $\mathsf{MixSign}$, users can use their real choice M_i during the signing process. The computationally bounded coercer will have no way to distinguish between 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, $\mathsf{MixInit}$ 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, HMMP23b], 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.

F.2 Comparison

Table 3. 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)	$\times$	✓	✓	✓
HPP20	**T**	U	$(1, N)$	(k, N)	**T**	**T**	U	(k, N)	$\times$	$\times$	$\times$	$\times$
Ours	U	U	$(1, N)$	(k, N)	U	**T**	U	(k, N)	✓	✓	✓	✓

Table 3 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 G, 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 A, their model only provides guarantees for honest users, and hence, they cannot achieve any of the properties required for e-voting.

G Extended Comparison of Voting Schemes

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.

Several countermeasures to coercion have been proposed in the literature, *e.g.,* Fake Credentials [JCJ05], Masking [WB09], Panic Password [CH11], Nullification [CCC+22a, CCC+22b]. Some of these approaches are tailored to homomorphic tallying where only the aggregated result is published. 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 A. 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.