Eos
A Universal Verifiable and Coercion Resistant Voting Protocol

anonymous

No Institute Given

Abstract. We present the voting protocol Eos that is based on a conditional linkable ring signatures scheme. Voters are organized in rings allowing them to sign votes anonymously. Voters may assume multiple pseudo identities, one of which is legitimate. We use the others to signal coercion to the Election Authority. Eos uses two mixing phases with the goal to break the connection between the voter and vote, not to preserve vote privacy (which is given already) but to guarantee coercion resistance by making it (nearly) impossible for a coercer to follow their vote through the bulletin board. Eos is universally verifiable and guarantees coercion resistance.

1 Introduction

All of the well-known voting protocols use a form of mixing to break the connection between vote and voter. Prêt-á-Voter [**?**], Helios [**?**], JCJ [**?**], Civitas [**?**], encrypt the votes at the time of casting, and then mix them before decrypting them for tabulation. Under the assumptions that at least one mixer is honest, so the argument, it is impossible to link a decrypted vote back to the identity of its voter. Selene [**?**] follows a slightly different approach. It uses tracker numbers and assigns them to voters, who eventually will be able to identify their votes on a bulletin board. This itself would not be novel, however, the trick is that a voter only gains access to their respective tracker after the election authority shares cryptographic information with the voter. This is supposed to happen only after voting has closed, and after the results have been published. In Selene, voters can fool a coercer into believing that *any* and not just *one* vote on the bulletin board was theirs. Both JCJ and Selene are receipt-free and coercion-resistant.

To our knowledge, not much work has been done, however, to leverage the power of ring signatures [**?**] to the design of voting protocols, besides perhaps the mention in [**?**,**?**,**?**]. Here, a ring refers to a group of participants who have the ability to sign messages anonymously by hiding their respective identities within the group. Assuming that the message was sent over an anonymous channel, the receiver of the message will not be able to trace the signature back to the signer. The idea of using ring signatures for voting brings some advantages. For example, an election authority will be able to publish all ballots and their signatures on a public bulletin board for public scrutiny without revealing the voters' identities.

Also, every voter can check their vote by accessing the bulletin board. But there are also challenges: First, the administration of the voter's identities, i.e. private keys, and second the protection of such identities from misuse, for example, for the purpose of vote-selling or voter coercion.

For the first challenge, we do not provide a solution in this paper, we merely assume that an effective, trusted identity infrastructure is in place that allows a voter to access a private key on a secure platform, for example, by means of a trusted Hardware Security Module (HSM). This may seem like a controversial assumption, but it really is not: Our experiments have shown that suitable HSMs exist. They may not be totally secure, but they are reasonable well designed to protect private keys against malicious agents and against malicious firmware. Hacking such an HSM is possible and requires fiddling with firmware and hardware, but we argue that this is difficult to do on a large scale in practice.

In this paper, we tackle a second challenge. We devise a ring signature scheme that allows voters to assume different pseudo identities, and that provides mechanisms for linking such identities from the point of view of the signature verifier. This scheme is a generalization of the so called linkable spontaneous anonymous group (LSAG) signatures [**?**], where all signatures from the same signer are linked. Correspondingly, we refer to this scheme as a *conditional-linking ring* (CLR) signature scheme. Using CLR signatures it is up to the voter to allowing linking or not. Linking does not break anonymity.

Based on CLR signatures, we develop then a universally verifiable, receipt-free, coercion-resistant, vote secrecy and integrity preserving voting protocol, named Eos, which is also described in this paper. This protocol assumes that each voter is in the possession of a private key, and that the voter must be authenticated to cast a valid vote. The authentication method yields additional entropy, such as, for example, a PIN number, a picture, or a fingerprint. When casting a vote, before submission, it is either put into a "green envelope" marking it as valid, or a "red envelope" marking it as invalid or possibly coerced, depending if the authentication succeeded or failed.

The entropy collected during the authentication procedure determines which pseudo-identity is used. Furthermore, we encrypt the color of the envelope, the electoral identity of the voter (which is unique), and the vote itself in order to guarantee coercion resistance and fairness, respectively. All envelopes together with their corresponding CLR signatures are recorded and shared on a public bulletin board. Pseudo identities are malleable, which means that from the point of view of the voter or the coercer, it will be "discrete logarithm hard" to link any two ballots from the bulletin board. Voters and coercers will be able to check whether the ballot was recorded as cast but neither the voter nor the coercer will be able to link their ballots.

The protocol proceeds and executes two verifiable decryption mix-nets in order. The first mix-net shuffles the ballots and decrypts color and the electoral identity, but leaves the votes encrypted. All this information is posted on a public bulletin board accompanied by zero-knowledge proofs of correct shuffle and correct decryption. As the color of the envelopes has become visible, red envelopes are discarded, and green envelopes are opened and the encrypted votes contained within (and only those) are passed to the second and last mixing stage. Note that a coercer cannot follow the coerced vote through mixing unless all mixing servers collude and are under the coercer's control. If two green envelopes are cast from two different pseudo-identities that correspond to the same electoral identity, this indicates a malicious attempt for a double vote. In this case we recommend to discard the votes of this electoral identity.

In the second and final mixing stage, we use another verifiable decryption mix-net to shuffle and decrypt votes. Also here, the resulting bulletin board including the zero-knowledge proofs of correct shuffle and decryption are published and are made available for scrutiny.

In order to afford decrypting the ballots without losing privacy or anonymity, the entire bulletin board has to be shuffled. For this phase, any mix-net that can generate a valid proof of correct shuffle can be used, for example, Sako Kilian [**?**], Fukuraua [**?**] or Neff [**?**,**?**]. In this paper we introduce a new mixer that we believe is more readable and easier to understand as it follows the classic Discrete Logarithm Equality Zero Knowledge proof, and it comes with efficient algorithms both in generating and verifying proofs of correct shuffle.

Contributions The contributions of this paper are (1) a Conditional Linkable Ring (CLR) signature scheme, (2) a mixer that produces a proof of correct shuffle, (3) the Eos voting protocol, and (4) proofs that the Eos is vote secrecy and integrity preserving, universally verifiable, receipt-free, and coercion resistant.

This paper is organized as follows. In Section **??**, we introduce basic notation that we will use throughout the paper. Next, we introduce the Conditional Linkable Ring (CLR) signature schemes in Section **??**. In Section **??**, we describe and analyze the mix-net that we propose to use in Eos including the proof of correct shuffle. Specifically, we present a version for the simple case, namely just atoms, and for the complex case, i.e. sequences of atoms. In Section **??** we the describe the Eos voting protocol in detail. In particular, we discuss the different phases, including setup, voting, two mixing and decryption phases. Each algorithm that we use is verifiable, and each input and out for each phase are published on a public bulletin board. In Section **??**, we establish that Eos has all desired properties. Finally, in Section **??** we present conclusions and give an outlook on future work.

2 Basic Notations

We define the following notations that we will use throughout our paper. We work with an ElGamal crypto systems that is defined over a cyclic group $\mathbb{G}$ of prime p order q generated by g. Please note that all mathematical operations presented in this paper are done modulo p.

We use standard notation and write $\{m\}_y^r = (g^r, y^r m)$ for the ElGamal tuple that one obtains by encrypting message m with randomness r and a public key y. We use r to denote randomness, and write $r \in_{\mathrm{R}} \mathbb{Z}_q$ to say that we choose r from $\mathbb{Z}$ modulo q at random using a uniform distribution. We will also use sequences of n elements over $\mathbb{Z}_q$, written as $\langle x_1, ..., x_n \rangle \in \mathbb{Z}_q^n$. We define $[n]$ to denote the index set of natural number up to n as $\{1, ..., n\}$. Furthermore, we use Greek letters σ for signatures and π to denote permutations and write $\mathbb{P}_n$ as the set of all permutation of n elements. Concatenation of two elements $a, b \in \{0,1\}^*$ is written as $a \| b$.

3 Conditional-Linkable Ring Signatures

We begin the technical exposition of this paper by defining the concept of Conditional Linkable Ring (CLR) signatures. In a linkable ring (LR) signature scheme [?] a verifier can learn which signatures originate from the same signer. Note that this does not mean that the verifier learns something about the identity of the signer — ring signatures always guarantee the anonymity of the signer. For our application, however linkability is overly restrictive – if we were to use LR signatures naively, we could not achieve coercion-resistance. Therefore, we relax the notion of linkability, and introduce *conditional linkability* given the signer the ability to link (revealing that the signatures originate from the same signer) or not to link (making it look like as if two signatures were produced by two different signers). We present the CLR signature scheme as a generalization of the LR signature scheme presented in [?] following the same phases:
Preparation phase: Every prospective member of the ring creates a secret key $x_i \in \mathbb{Z}_q$, and shares the public key $y_i = g^{x_i}$ with a designated election authority.
Set-up phase: The election authority produces the set of ring members $L = \langle y_1, ..., y_n \rangle$, which represents eligible voters.
Identity selection phase: Assume that the signer is the member of the ring at position α. The signer selects a pseudo-identity by choosing $\phi \in_R \mathbb{G}$ and by forming the pair (ϕ, θ) where $\theta = \phi^{x_\alpha}$. The pair (ϕ, θ) is called a pseudo identity. If the signer wishes his signature to be linkable, he will always choose the same value of ϕ, otherwise, he will choose a different value for ϕ for every signature.

CLR signatures give us quite a lot of freedom to choose ϕ. To see that LR signatures proposed in [?] are simply an instance of CLR signatures, choose ϕ to be the cryptographic hash value of L (written as $h = H_2(L)$ in their paper) and compute $\theta = h^{x_\alpha}$. Liu et al. denote this value as $\tilde{y}$. The security of linkability reduces therefore to a secure choice of ϕ, for which it is sufficient to require that finding $\log_g \phi$ is a hard problem.

The idea to consider other ways to compute ϕ is already present in [?], what is new in our work is to allow ϕ to be drawn at random from $\mathbb{G}$. We shall see in Section ?? how to choose ϕ while still guaranteeing that CLR signatures as used in Eos guarantee receipt-freeness and coercion resistance. The properties and proofs of the LR signatures scheme presented in [?] carry over to the CLR signature scheme with the exception of linkability that no longer holds. CLR signatures instead are conditionally linkable.

One other aspect of LR signature that our scheme preserves is *claimability.* This means that, once generated, the signer only can come forth and claim responsibility of a signature generated by him by providing in zero knowledge a discrete logarithm equality proof between $\log_\phi \theta = \log_g y_\alpha$. Note that, to do this, the signer has to break the anonymity aspect of LR signature scheme by showing his public key y_α, consequently disclosing his position α in the ring.

3.1 Signing Algorithm

We begin now with the presentation of the signing algorithm. Our algorithm follows closely the original LR signing algorithm described in [?], the most notable difference being that we use ϕ and θ instead of h and $\tilde{y}$, respectively. Given the message to be signed $m \in \{0,1\}^*$, for each element in the ring L, a cipher text is computed, starting from the position of the signer in the ring, α, to the end of the list and from the beginning of the list back to α.

The first computed cipher text is therefore

$$c_{\alpha+1} = \mathcal{H}\left(m \| g^u \| \phi^u\right)$$

where, $\mathcal{H}$ is a cryptographic hash function that returns a number from $\mathbb{Z}_q$ (referred to as H_1 in [?]) and $u \in_{\mathrm{R}} \mathbb{Z}_q$. Next, for each element in L from $i = \alpha + 1$ to n and from $i = 1$ to $\alpha - 1$, the signer computes:

$$c_{i+1} = \mathcal{H}\left(m \| g^{s_i} \cdot y_i^{c_i} \| \phi^{s_i} \cdot \theta^{c_i}\right)$$

where each $s_i \in_{\mathrm{R}} \mathbb{Z}_q$ is a random number assigned for each entity in L. Note that at step $i = n$, we generate $c_1 = c_{i+1}$. The signer computes:

$$s_\alpha = u - x_\alpha \cdot c_\alpha \bmod q$$

Finally, the output of the CLR signing algorithm is the signature σ on message m with the pseudo-identity (ϕ, θ) that has the following structure:

$$\sigma(m) = \left(c_1, \langle s_1, ..., s_n \rangle\right)$$

3.2 Verification Algorithm

After having received a message m and the corresponding signature, $\sigma(m) = (c_1, \langle s_1, .., s_n \rangle)$ from the pseudo-identity (ϕ, θ), anybody can now verify the signature by executing the following steps of the verification algorithm, which is computationally linear in terms of size of L, that should output either the signature is valid or not, i.e. it was generated by a ring member or not. For each element in L starting from $i = 1$ to n compute:

$$c_{i+1} = \mathcal{H}\left(m \| g^{s_i} \cdot y_i^{c_i} \| \phi^{s_i} \cdot \theta^{c_i}\right)$$

The algorithm validates the signature if and only if the last $c_{n+1} = c_1$, where c_1 is contained as the first argument in the signature.

4 Mix-net

Next, we describe the mix-net the we will be using. The goal of the mix-net is to shuffle ballots in an indistinguishable way, which means that it is impossible to correlate inputs and outputs of the mix-net. In essence, any reencrypting mix-net that also provides a proof of correct shuffle would fit the bill. While analyzing the different mix-net protocols, in particular [**?**,**?**,**?**,**?**], we observed simplifications to Neff's protocol that we describe next, but at the cost of losing universal soundness. We believe that our protocol is mathematically more elegant, more intuitive, and easier to understand than Neff's protocol as it follows closely the classic discrete logarithm equality zero knowledge proof. In addition, the algorithms to generate and verify proofs of correct shuffle are efficient.

4.1 Proof of Correct Shuffle

Our mix-net consists of several mixing servers, each of which receives as input a bulletin board of n ElGamal tuples (a_i, b_i) and produces an output bulletin board, where all tuples are reencrypted and shuffled:

$$(c_i, d_i) = \left(a_{\pi(i)} \cdot g^{s_{\pi(i)}}, b_{\pi(i)} \cdot y^{s_{\pi(i)}}\right) \text{ for } i \in [n]$$

where y is the encryption key, $\langle s_1, ..., s_n \rangle \in_{\mathrm{R}} \mathbb{Z}_q^n$ the randomness used for reencryption, and $\pi \in_{\mathrm{R}} \mathbb{P}_n$ the permutation underlying the shuffle.

The challenge when defining a mix-net is how each mixing server can prove the correctness of the shuffle to a public verifier, without revealing information about the permutation π or the randomness $\langle s_1, ...s_n \rangle$ used for reencryption.

The following proof of correct shuffle is inspired by the protocol developed by Sako and Kilian [**?**], where they say that the proof should show that the output of the mixer could be generated in some manner from the input. Finally, the aggregation of the entire set of ElGamal pairs, is inspired by Ramchen's paper [**?**]. Our proof follows the natural flow of a classic discrete logarithm equality zero knowledge proof depicted in Figure **??**, i.e. the mix server publishes a commitment of the input, a verifier challenges the output of the mixer and then mixer generates a response, which convinces the verifier that the shuffle was correct.

Let (a_i, b_i) be the n ElGamal tuples that form the input for the mix server. Let (c_i, d_i) be n ElGamal tuples, computed as above, be the output of the mix server. To prove the correctness of the shuffle, the mix server $\mathcal{P}$ and a public verifier $\mathcal{V}$ have to follow the protocol that is described in Figure **??**.

$\mathcal{P}$ secretly generates: $k \in_{\mathrm{R}} \mathbb{Z}_q$ and $\langle m_1, ..., m_n \rangle \in_{\mathrm{R}} \mathbb{Z}_q^n$ and publishes commitment:

$$A = g^k \cdot \prod_{i \in [n]} {a_i}^{m_i} \qquad\qquad B = y^k \cdot \prod_{i \in [n]} {b_i}^{m_i}$$

$\mathcal{V}$ sends challenge: $\langle e_1, ..., e_n \rangle \in_{\mathrm{R}} \mathbb{Z}_q^n$
$\mathcal{P}$ publishes response:

$$r_i = m_i + e_{\pi^{-1}(i)} \bmod q \text{ for } i \in [n]$$

$$t = k + \sum_{i \in [n]} \left(e_i \cdot s_{\pi(i)}\right) \bmod q$$

$\mathcal{V}$ accepts the proof if the following verification calculations match:

$$g^t \cdot \prod_{i \in [n]} {a_i}^{r_i} = A \cdot \prod_{i \in [n]} {c_i}^{e_i} \qquad\qquad y^t \cdot \prod_{i \in [n]} {b_i}^{r_i} = B \cdot \prod_{i \in [n]} {d_i}^{e_i}$$

Fig. 1. Protocol: Proof of Correct Shuffle

Theorem 1. *The protocol described in Figure* **??** *is complete.*

Proof. To show that out protocol is correct, we have to prove that the equations that $\mathcal{V}$ verifies hold, when the response $(\langle r_1, ..., r_n \rangle, t)$ is computed correctly.

$$\begin{aligned}
g^t \cdot \prod_{i \in [n]} {a_i}^{r_i} &= A \cdot \prod_{i \in [n]} {c_i}^{e_i} \\
g^{k + \sum_{i \in [n]} e_i \cdot s_{\pi(i)}} \cdot \prod_{i \in [n]} {a_i}^{m_i + e_{\pi^{-1}(i)}} &= g^k \cdot \prod_{i \in [n]} {a_i}^{m_i} \cdot \prod_{i \in [n]} \left(a_{\pi(i)} \cdot g^{s_{\pi(i)}}\right)^{e_i} \\
g^k \cdot g^{\sum_{i \in [n]} e_i \cdot s_{\pi(i)}} \cdot \prod_{i \in [n]} {a_i}^{m_i} \cdot \prod_{i \in [n]} {a_i}^{e_{\pi^{-1}(i)}} &= g^k \cdot \prod_{i \in [n]} {a_i}^{m_i} \cdot \prod_{i \in [n]} \left({a_{\pi(i)}}^{e_i} \cdot g^{s_{\pi(i)} \cdot e_i}\right) \\
\prod_{i \in [n]} g^{e_i \cdot s_{\pi(i)}} \cdot \prod_{i \in [n]} {a_i}^{e_{\pi^{-1}(i)}} &= \prod_{i \in [n]} {a_{\pi(i)}}^{e_i} \cdot \prod_{i \in [n]} g^{s_{\pi(i)} \cdot e_i} \\
\prod_{i \in [n]} {a_i}^{e_{\pi^{-1}(i)}} &= \prod_{i \in [n]} {a_{\pi(i)}}^{e_i}
\end{aligned}$$

The last equation in the proof is true because the product aggregation happens through the entire set of n elements. This means we can compute the product aggregation of ${a_i}^{e_{\pi^{-1}(i)}}$ in a permuted way, namely ${a_{\pi(i)}}^{e_{\pi(\pi^{-1}(i))}} = {a_{\pi(i)}}^{e_i}$.

In the same way, the second equations that $\mathcal{V}$ has to verify can be proven to hold, if the response $(\langle r_1, ..., r_n \rangle, t)$ is computed correctly. □

Theorem 2. *The protocol described in Figure* **??** *satisfies special soundness.*

Proof. Each transcript of our protocol has the following form:

$$\text{View}\,[\mathcal{P} \leftrightarrow \mathcal{V}] = (A, B, \langle e_1, ...e_n \rangle, \langle r_1, ..., r_n \rangle, t)$$

where A and B represent the initial commitment, sequence $\langle e_1, ...e_n\rangle$ is the random challenge picked by the verifier and the sequence $\langle r_1, ..., r_n\rangle$ together with the value t represent the response to the challenge.

For any cheating prover $\mathcal{P}^*$ (that does not know the permutation $\pi(i)$ and the re-encryption coefficients $\langle s_1, ..., s_n\rangle$), given two valid conversations between $\mathcal{P}$ and the verifier $\mathcal{V}$, $(A, B, \langle e_1, ...e_n\rangle, \langle r_1, ..., r_n\rangle, t)$ and $(A, B, \langle e'_1, ...e'_n\rangle, \langle r'_1, ..., r'_n\rangle, t')$, that have the same commitment but different challenge $e_i \neq e'_i$, the permutation $\pi(i)$ used for shuffling the board can be computed in polynomial time in the following way:

$$\forall i \in [n] \text{ there } \exists p \text{ , such as } \pi(i) = p, \text{ where } r_p - r'_p = e_i - e'_i$$

Please, note that, the permutation $\pi(i)$ is the actual secret that the mixing server has to hide. The re-encryption mechanism is used exactly with this purpose of hiding, but extracting the re-encryption coefficients $\langle s_1, ..., s_n\rangle$ is out of our scope.

□

Theorem 3. *The protocol described in Figure* **??** *is honest verifier zero knowledge.*

Proof. We prove that for any cheating verifier $\mathcal{V}^*$, there exists a simulator $\mathcal{S}$ that can produce a computationally indistinguishable transcript of the protocol that would take place between $\mathcal{P}$ and $\mathcal{V}^*$ if it knew the challenge in advance.

Our simulator $\mathcal{S}$ gets as input: the initial set of n ElGamal tuples (a_i, b_i), the mixed set of ElGamal tuples (c_i, d_i) and a challenge in form of a random sequence $\langle e_1, ..., e_n\rangle$. $\mathcal{S}$ proceeds by picking a random response of the transcript:

$$\langle r_1, ..., r_n\rangle \in_{\mathrm{R}} \mathbb{Z}_q^n$$

$$t \in_{\mathrm{R}} \mathbb{Z}_q$$

$\mathcal{S}$ computes the initial commitment:

$$A = g^t \cdot \prod_{i\in[n]} \left({a_i}^{r_i} \cdot {c_i}^{-e_i}\right) \qquad\qquad B = y^t \cdot \prod_{i\in[n]} \left({b_i}^{r_i} \cdot {d_i}^{-e_i}\right)$$

$\mathcal{S}$ outputs the transcript: $(A, B, \langle e_1, ..., e_n\rangle, \langle r_1, ..., r_n\rangle, t)$.

It is obvious that the transcript $\mathcal{S}$ outputs will always pass the equations that $\mathcal{V}$ has to verify. Note that this transcript was generated independently of the permutation $\pi(i)$ and the re-encryption coefficients $\langle s_1, ...s_n\rangle$ used for mixing, thus is zero knowledge.

□

4.2 Proof of Correct Parallel Shuffle

The proof of shuffle for mixing individual ciphertexts can be extended to a proof of correct parallel shuffle for sequences of ElGamal tuples. Such a parallel mixer expects as input a matrix over ElGamal tuples with n rows and ℓ columns: $(a_{i,j}, b_{i,j})$, where $i \in [n]$ and $j \in [\ell]$, the Mixer then outputs a mixed and re-encrypted matrix where only the rows are shuffled. This matrix is defined as

$$(c_{i,j}, d_{i,j}) = \left(a_{\pi(i),j} \cdot g^{s_{\pi(i),j}}, b_{\pi(i),j} \cdot y^{s_{\pi(i),j}}\right)$$

where y is the encryption key, $\langle s_{1,1}, ..., s_{n,\ell}\rangle \in_{\mathrm{R}} \mathbb{Z}_q^{n\times\ell}$ are the re-encryption coefficients and $\pi \in_{\mathrm{R}} \mathbb{P}_n$ is a permutation.

The proof of correct parallel shuffle depicted in Figure **??** is designed to convince a public verifier that the same permutation $\pi(i)$ was applied to each column. The proof, inspired by [**?**], deviates slightly from the construction that we have presented for the simple case in the previous section. By applying the same challenge e_i to all columns in the matrix, the verifier will be assured that the same permutation $\pi(i)$ was applied consistently across all columns.

$\mathcal{P}$ secretly generates: $\langle k_1, ..., k_\ell\rangle \in_{\mathrm{R}} \mathbb{Z}_q^\ell$ and $\langle m_1, ..., m_n\rangle \in_{\mathrm{R}} \mathbb{Z}_q^n$ and publishes commitment:

$$A_j = g^{k_j} \cdot \prod_{i\in[n]} {a_{i,j}}^{m_i} \text{ for } j \in [\ell] \qquad\qquad B_j = y^{k_j} \cdot \prod_{i\in[n]} {b_{i,j}}^{m_i} \text{ for } j \in [\ell]$$

$\mathcal{V}$ sends challenge: $\langle e_1, ..., e_n\rangle \in_{\mathrm{R}} \mathbb{Z}_q^n$
$\mathcal{P}$ publishes response:

$$r_i = m_i + e_{\pi^{-1}(i)} \bmod q \text{ for } i \in [n]$$

$$t_j = k_j + \sum_{i\in[n]} \left(e_i \cdot s_{\pi(i),j}\right) \bmod q \text{ for } j \in [\ell]$$

$\mathcal{V}$ verifies for each $j \in [\ell]$ and accepts the proof if all calculations match:

$$g^{t_j} \cdot \prod_{i\in[n]} {a_{i,j}}^{r_i} = A_j \cdot \prod_{i\in[n]} {c_{i,j}}^{e_i} \qquad\qquad y^{t_j} \cdot \prod_{i\in[n]} {b_{i,j}}^{r_i} = B_j \cdot \prod_{i\in[n]} {d_{i,j}}^{e_i}$$

Fig. 2. Protocol: Proof of Correct Parallel Shuffle

Our proof has the same security properties as the simple proof presented in the previous section. Completeness holds as it follows a slightly more generalized version of the calculation done in the proof of Theorem **??**, as now we need to take into account index j for each ElGamal tuple in a sequence. Special soundness follows exactly the same arguments as in the proof of Theorem **??**. The proof of honest verifier zero knowledge is an elegant generalization of the proof of Theorem **??**: The simulator $\mathcal{S}$ is modified in such a way that it outputs a sequence of initial commitments for each $j \in [\ell]$:

$$A_j = g^{t_j} \cdot \prod_{i \in [n]} \left({a_{i,j}}^{r_i} \cdot {c_{i,j}}^{-e_i} \right) \qquad B_j = y^{t_j} \cdot \prod_{i \in [n]} \left({b_{i,j}}^{r_i} \cdot {d_{i,j}}^{-e_i} \right)$$

where $\langle r_1, ..., r_n \rangle \in_{\mathrm{R}} \mathbb{Z}_q^n$ and $\langle t_1, ..., t_\ell \rangle \in_{\mathrm{R}} \mathbb{Z}_q^\ell$ represent the response of the challenge $\langle e_1, ..., e_n \rangle$.

This proof might be seen as an ℓ-run of the simple protocol, to which we feed the same challenge sequence $\langle e_1, ..., e_n \rangle$. Note that our proof of correct parallel shuffle does not break the honest verifier zero knowledge property because in each run, the prover picks a different value k_j. Moreover, each run of the protocol is applied on a different partial board $(a_{i,j}, b_{i,j})$, for $i \in [n]$. We summarize these findings in form of a theorem.

Theorem 4. *The proof of correct shuffle satisfies completeness, special soundness, and honest verifier zero knowledge.*

As for complexity, the computation cost for the proof of correct parallel shuffle is summarized as follows. To generate a proof of correct shuffle of the entire matrix, a prover will require $2n\ell + 2\ell$ exponentiations and $3n\ell$ multiplications. In contrast, the verifier will be more expensive, because it requires $4n\ell + 2\ell$ exponentiations and $4n\ell$ multiplications.

5 Eos Protocol

CLR signatures and mix-nets are the building blocks of the Eos Voting protocol that we define next. The hallmark characteristics of the protocol is that voters are organized in rings and they can sign their ballots anonymously. Mix-nets are used to make it impossible for coercers to trace the coerced ballot. Each voter has the possibility to assume one out of many pseudo identities. If coerced, the voter simply picks a different identity and signals to the election authority that the ballot was submitted as coerced. As an analogy, we may imagine that voter has access to green and red envelopes. A green envelope means that the vote contained within reflects the voter's intention while a red envelope signals coercion. The color of the envelope will be encrypted. Note, that for this to work, Eos must make the assumption that there is a device that can be used to sign ballots.

There are different entities that participate in the overall process of the election, each having different roles and duties. The following stakeholders are part in the Eos protocol.

A *voter* is a person that can participate legitimately in the election process. All voters together generate a set of CLR signed ballots as input. Every ballot cast must be signed by an eligible voter, but not every eligible voter is required to cast a ballot. A voter may be under the influence of a *coercer*. The coercer may be colluding with the authorities, however, to achieve coercion-resistance, we must assume at least one of the mixing server is not under the coercer's control. The *election authority* administrates the election. Its role is to initialize the election, form the ring for CLR signing and collect the signed ballots cast by the voters.

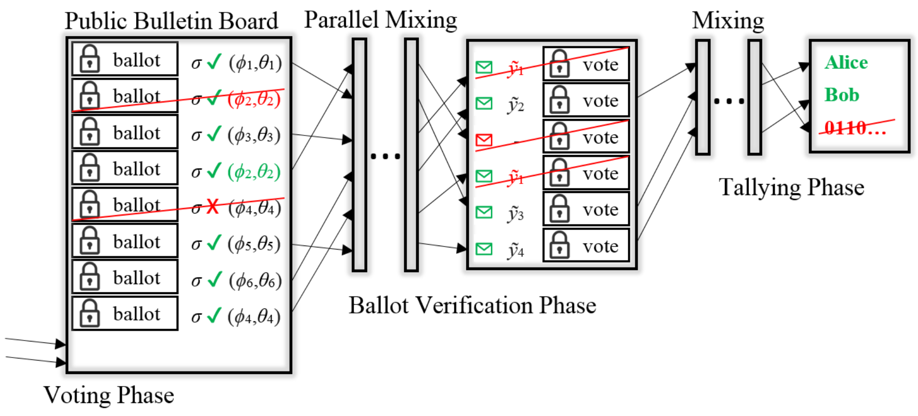

Fig. 3. Protocol Overview

Each ballot is recorded on a public bulletin board allowing the voter to check that it was recorded as cast. The election authority is responsible for starting and supervising the mixing phase. Eos assumes that all bulletin boards are append-only, but other variants are possible (although not discussed here). Votes are cast for one *candidate* only. The result of Eos is a public bulletin board with all votes recorded in clear text. Eos supports distributed and threshold decryption schemes, which entails that shares of the decryption key are distributed among different tellers.

5.1 Election Set-up

The election authority prepares $L = \langle y_1, ..., y_n \rangle$, the list of all eligible voter public keys that will form the ring. In addition, the election authority prepares the set of candidates as vote choices $\mathbb{V} \subset \mathbb{Z}_p^*$. We assume that there are several mixing servers maintained by different non-colluding stakeholders, each with access to a good source of entropy. We call a mixing server honest, if it is correct and not controlled neither by the adversary nor the coercer. An honest mixing server does not reveal the permutation used for mixing.

As it is common practice, we use a (t, e)-threshold cryptosystem, as described in [**?**], to encrypt and decrypt ballots. All ballots will be encrypted with a public key Y, while the corresponding private key is split and shared among e tellers. Recall that in threshold encryption, it takes the shares of at least t tellers in order to decrypt successfully. Decryption will fail, if less than t shares are available.

5.2 Voting Phase

The voter commits to the color of the envelope, using the respective private key $x_i \in \mathbb{Z}_q$ associated with a public key $y_i = g^{x_i} \in L$ and some entropy generated during the authentication process. We use both, private key and entropy to derive (deterministically) the randomness used in the ElGamal encryption using a secure hashing function. Once the ballot generated and signed, it is sent to the election authority that publishes it on the (append only) public bulletin board.

Ballot Generation A ballot consists of three ElGamal tuples, each representing the following: an encryption of the color of the envelope, an encryption of the electoral identity of the signer and an encryption of the vote. *Encryption of the color:* Recall from Section **??** the definition of h and $\tilde{y}$. A green envelope is formed as an encryption of h, whereas the red envelope is an encryption of 1. *Encryption of the electoral identity:* In the case of a green envelope, there will be an encryption of $\tilde{y}$, while in the case of a red envelope, there will be again an encryption of 1. *Encryption of the vote:* The vote, to be encrypted, will be represented as value $v \in \mathbb{V}$.

The entire ballot generation algorithm is depicted in Figure **??**. Formally, the ballot generation algorithm for voter α depends on the following inputs, the authentication entropy $\mathcal{E}$ (such as PIN, a picture, a fingerprint), the private key x_α, and an election specific generator h. The first two ElGamal tuples (F, ϕ) and (T, θ) of the generated ballot play an important role in forming the pseudo identity. Let ϕ be the second projection (trap door commitment) of the encryption of the color of the envelope, and θ is the second projection of the encryption of the electoral identity. Together, (ϕ, θ) form the pseudo identity of the signer.

The algorithm of Ballot Generation starts by the device computing:

$$f = \mathcal{H}\left(\mathcal{E} \| x_\alpha \| h\right)$$

$$t = f \cdot x_\alpha \bmod q$$

$$d \in_{\mathrm{R}} \mathbb{Z}_q$$

$$(D, \delta) = \{v\}_Y^d = \left(g^d, Y^d \cdot v\right)$$

If authentication was successful:

$$(F, \phi) = \{h\}_Y^f = \left(g^f, Y^f \cdot h\right) \qquad (T, \theta) = \{\tilde{y}\}_Y^t = \left(g^t, Y^t \cdot \tilde{y}\right)$$

If authentication was unsuccessful:

$$(F, \phi) = \{1\}_Y^f = \left(g^f, Y^f\right) \qquad (T, \theta) = \{1\}_Y^t = \left(g^t, Y^t\right)$$

The generated ballot is: $\left(\left(F, \phi\right), \left(T, \theta\right), \left(D, \delta\right)\right)$.

Fig. 4. Algorithm: Ballot Generation

Note, that due to the deterministic computation of the randomness used here for ElGamal encryption, a voter is able to generate the same pseudo identity deterministically multiple times. If an implementation of Eos uses, for example, a PIN code as entropy for authentication, the pseudo identity of the voter is uniquely defined by the choice of PIN. The valid pseudo identity is selected locally on the voting device by correct authentication, i.e. by using the correct PIN. If the same coercer forces the same voter to vote multiple times, Eos will do so, as it computes the same coerced pseudo identity.

In addition, to guarantee the internal consistency of an encrypted ballot the signer proves in zero knowledge, that the encryptions of the color of the envelope and of the electoral identity are correct by providing a proof of the discrete logarithm equality between $log_F T = log_\phi \theta$. This means that there will be only one pseudo identity per device, for each value of f. Note that $\log_\phi \theta = x_\alpha$ (i.e. private key of an eligible voter) is enforced by the CLR signature verification algorithm. Together with the encrypted vote, one must include also a proof of knowledge of the discrete logarithm of $\log_g D$. This will protect against vote copying.

A malicious user might also try to cast multiple countable votes by encrypting his electoral identity with different values of f. Obviously, this could happen mathematically, but practically this attack would require the malicious voter to tamper with software or hardware to trick the protocol. This however, would be noticed as we discuss later in the description of the Ballot Verification Phase **??** where the value of $\tilde{y}$ will be decrypted and duplicates will be visible. We suggest, in this case, to discard the multiple votes from the same electoral identity.

Note that only during the Tallying Phase (Section **??**), the vote v will be visible in plain text. There the public can scrutinize and validate each plain text and if it represents a valid candidate v, such that $v \in \mathbb{V}$. Otherwise, the vote should be disregarded.

Signing a ballot The CLR signature of a ballot is computed as described in Section **??**. Concretely, the pseudo-identity (ϕ, θ) is embedded in the ballot and the message to be signed is publicly computable.

$$m = \mathcal{H}\left(D \| \delta\right)$$

The CLR signature will then be computed as:

$$\sigma\left(m\right) = \left(c_1, \langle s_1, ..., s_n \rangle\right).$$

Beside the ballot and the signature, a voter has to send also the two zero knowledge proofs described above: one for proving the correct encryptions and the second for proving the knowledge of the vote.

Public Bulletin Board The public bulletin board is a public large file, to which only the election authority is allowed to append. Each entry on the board contains a ballot, its corresponding CLR signature and two zero knowledge proofs. Note that no ballots will ever be removed from the public bulletin board, only added. Each voter and coercer is able to check that their respective vote have been appended on the bulletin board after submission, hence individually verifiable. Ballots from the same pseudo identity can be related on the board as they have pseudo identity (ϕ, θ). Assuming that voter and coercer use different pseudo identities, their votes can only be related with a negligible probability.

5.3 Ballot Verification Phase

Once the Voting Phase finished and all votes have been collected, the election authority no longer accepts signed ballots and seals the public bulletin board. The election authority performs a cleansing operation on the public bulletin board and only copies those ballots (without signatures and zero knowledge proofs) to a new board, for which both zero knowledge proofs are checked and the CLR signature is validated. In the case multiple ballots were cast from the same pseudo identity, only the most recent ballot is copied. The cleansing operation is publically verifiable. This procedure is visible in Figure **??**, as some of the ballots get crossed out and disregarded.

Parallel Mixing Before the election authority commences with decrypting the ballots, it first uses the parallel shuffle described in Section **??** to shuffle the entire bulletin board by reencrypting all three ElGamal tuples of each entry. We assume that there are multiple mixing servers, at least one of which is honest. Each mixing server performs a mixing operation on the output of the previous mix server and constructs a mixed board together with a Proof of Correct Parallel Shuffle, which is subsequently validated by the election authority. In case a proof fails, an error has occured, and the output of this particular server is disregarded and another mixing server is used. After the shuffle is complete neither voters nor coercers will be able to identify their ballots on the the mixed board, unless all mixing servers collude.

Pseudo Identity Decryption To decrypt the pseudo identities for each entry, t tellers out of e must come together and decrypt the contents of the mixed board using the threshold decryption scheme. At this stage, only the first two ElGamal tuples will be decrypted, i.e. the color of the envelope and the electoral identity of each entry. The vote itself will not be decrypted and this is guaranteed assuming that strictly more than $e - t$ tellers are honest. The closer t and e, the fewer honest tellers are needed. All ballots whose color of envelope do not decrypt to the value of h or 1 will be disregarded as they are not well-formed. All ballots whose color decrypts to 1 will be discarded because they are coerced. The remaining ballots should all have unique values for the electoral identity. In case there are multiple ballots whose electoral identity decrypts to the same value $\tilde{y}$, these ballots should be disregarded as they represent an attempt to cast multiple votes. This scenario might happen in case of a malicious voter misusing the Eos protocol. These examples can be seen in Figure **??** as some ballots are crossed out in the Ballot Verification Phase.

5.4 Tallying Phase

Only the remaining ballots encrypt votes that should be counted. To extract the votes from these ballots, we drop the encryptions of the color and electoral identity and create a new bulletin board to undergo another round of mixing before decryption. This bulletin board now only contains encryption of the votes, i.e. (D, δ). This way, we assure that the link between an electoral identity and the vote is broken.

Mixing Recall that CLR signatures are claimable as the voter can prove in zero knowledge the discrete logarithm equality between $log_g y_i = log_h \tilde{y}$. By mixing the list of encrypted votes once more, a voter might only prove to a potential coercer, that he voted but not who he voted for. For this phase, the simple mixing protocol described in Section **??** is used.

Vote Decryption Finally, the tellers get together once more and perform a threshold decryption protocol of the new board of encrypted votes, producing a proof of correct decryption. After decryption, each value of v should be counted as a valid vote for the respective candidate if $v \in \mathbb{V}$.

6 Analysis

In this secion we analyze Eos for its properties. Eos is individually verifiable, because a voter can check if his ballot was correctly recorded by checking the public bulletin board. Moreover, all zero knowledge proofs generated by the mixing servers and the tellers are public and thus universally verifiable. Eos is designed to assure that every ballot published on the public bulletin board will be handled and counted correctly.

In order to argue for the integrity of the Eos protocol, we have to prove the correctness of the final bulletin board of decrypted ballots. First, the the mixing operations applied on the entire bulletin board can be challenged by any public verifier. Secondly, the decyryption operations can also be challenged by any public verifier. Finally, the disposal of coerced votes is a transparent process as the red and green envelopes will be visible in the Ballot Verification Phase without compromising any other properties of our voting protocol.

The secrecy of the vote is guaranteed by the ElGamal crypto system and the use of a cryptographically secure hashing function. The anonymity of the voter is guaranteed by the CLR signature scheme, which protects the voter's true identity. At the same time, we have to assume that there will be at least one honest mixing server that will not disclose its choice of permutation. This assures that a coercer is not able to trace his ballot all the way to the decrypted board and learn if the coerced vote was cast in a green or red envelope. Last but not least, we assume that there will be at least $e - t + 1$ honest tellers to participate in the threshold decryption. This means that we assume that t dishonest talliers will never collude to decrypt the ballots from the bulletin board before the the final step of the protocol as this will represent an attack to the fairness of the election.

On terms of receipt-freeness, Eos guarantees that neither a voter nor a coercer can identify his ballot on the decrypted board. This is achieved through two mixing phases which break the connection between the ballot on the public bulletin board and the one on the decrypted board. In addition, a coercer may force a voter to cast a particular vote. In this case, the voter will use one of the alternate pseudo identities to sign the ballot, which will subsequently be discarded during the Ballot Verification Phase. Note that the pseudo identity used for a coerced vote is indistinguishable from the real pseudo identity of the voter.

One bit of power that the coercer has over the voter is that of an abstention attack, to force a vote for a particular candidate for which he knows will receive only this one vote, something like an invalid write-in vote. All the coercer has to do is to check that this vote appears on the final decrypted board of votes. If it does, this would mean that the coercer forced the voter to cast an invalid vote, spoiling the ballot. This situation can be mitigated by the voter proving that his vote is part of the valid set of votes without revealing what the vote is, for example using a disjunctive zero knowledge proof protocol as described in [**?**]. These votes could be cleansed earlier, and would therefore never appear on the final board.

7 Conclusion and Future Work

We have described in this paper a verifiable, privacy preserving coercion-resistant voting protocol that was inspired by Conditional-Linkable Ring (CLR) signatures. Furthermore, we argued for why this protocol protects the integrity of the election, how it guarantees the secrecy of the vote, receipt freeness and is coercion resistant as long as one of the mixing servers is honest. In future work, we plan to reduce the size of CLR signatures from linear to constant size, for example using "accumulators" such as described in [**?**]. These constant sized signatures can also be made linkable [**?**].

Our protocol is different from other coercion mitigating protocols, such as Selene [**?**] or JCJ [**?**]. In Selene tracker numbers are generated prior to the election, and once a vote is cast, only the trap-door commitment is shared with the voter. After the election is over, the randomness necessary to decrypt the tracker number is shared, allowing each voter to gain confidence in that his or her vote was recorded correctly. Moreover, this protocol allows every voter to trick a potential coercer into believing that he or she voted for the coercer's choice. In JCJ, every voter has access to different kinds of credentials. One credential is there to be used to cast a valid vote, whereas as the other credentials are there to cast a vote that from the outset looks like a valid vote, but really is not. The election authority will be able to weed out coerced votes. A detailed to Selene and JCJ is left to future work.