Thwarting Last-Minute Voter Coercion

Rosario Giustolisi, Maryam Sheikhi Garjan and Carsten Schuermann
IT University of Copenhagen

***Abstract*—Counter-strategies are key components of coercion-resistant voting schemes, allowing voters to submit votes that represent their own intentions in an environment controlled by a coercer. By deploying a counter-strategy a voter can prevent the coercer from learning if the voter followed the coercer's instructions or not. Two effective counter-strategies have been proposed in the literature, one based on *fake credentials* and another on *revoting*. While fake-credential schemes assume that voters hide cryptographic keys away from the coercer, revoting schemes assume that voters can revote after being coerced.**

In this work, we present a new counter-strategy technique that enables *flexible vote updating*, that is, a revoting approach that provides protection against coercion even if the adversary is able to coerce a voter at the very last minute of the voting phase. We demonstrate that our technique is effective by implementing it in Loki, an Internet-based coercion-resistant voting scheme that allows revoting. We prove that Loki satisfies a game-based definition of coercion-resistance that accounts for flexible vote updating. To the best of our knowledge, we provide the first technique that enables deniable coercion-resistant voting and that can evade last-minute voter coercion.

1. Introduction

One of the central challenges of remote and Internet-based voting schemes is that the act of casting the vote no longer takes place in a protected, restricted, and safe environment, such as a polling station, but in an environment that might very well be under an adversary's control. The adversary may use this influence to perpetrate attacks with the goal to subvert an election.

The lack of physical limitations, exposure to blackmail, and other threats, compromise secrecy and integrity of the ballot and introduce risks of coercion. Voter coercion is generally understood as the undue influence of an adversary over a voter, allowing the adversary to force a voter to cast a ballot reflecting the adversary's rather than the voter's intention, or to force a voter to abstain from voting altogether. Hence, it implies that a voter must not receive any evidence that may be used to prove to a third-party, for example, a family patriarch, a party representative, or any other kind of coercer, how the voter voted.

Coercion-resistant Internet-based voting schemes protect against voter coercion in uncontrolled environments. Most schemes achieve coercion resistance by either *fake credentials* or by *deniable revoting*. Common to both approaches is the use of *counter-strategies*, which allow voters to follow a different procedure than coercer's instructions. A counter-strategy is successful if the coercer cannot tell whether the voter followed the coercer instructions or not.

In fake credentials, voters create and cast fake ballots when voting under coercion. Such ballots are verifiably removed from the tally. In deniable revoting, voters update or nullify previously cast votes while being under coercion.

These approaches need different assumptions. Fake-credential schemes assume that voters i) cannot be coerced during registration, ii) can store and hide cryptographic key material during the voting phase, and iii) need to lie convincingly while being coerced. Revoting-based voting schemes assume that i) voter authentication is inalienable during the voting phase, and ii) the adversary does not coerce the voter at the end of the voting phase, namely, the voter has always a chance to revote after being coerced. A way to remove the latter assumption is to use *flexible vote updating*, that is, a revoting approach that allows the adversary to coerce a voter at any point of the election, including the very last minute of the voting phase.

In this work, we propose the first counter-strategy technique that allows a voter to deniably revote while resisting coercion at any point of an election. Our technique relies on two key insights. First, we observe that a *Voting server* can efficiently generate noise ballots that obfuscate the encrypted ballots cast by voters without knowing the original vote (e.g., by re-randomizing them). Such noise ballots hide revoting patterns induced by voters or coercers.

Second, we observe that the Voting server and the voter are the only parties who know with certainty for each ballot associated with the voter whether it is a ballot cast by the voter or a noise ballot cast by the Voting server. A coercer cannot tell the difference between the two provided that votes are continuously being obfuscated and that a voter might (or might not) have cast other votes while not being coerced. Hence, when the voter casts a ballot, the Voting server can challenge the voter to identify the ballots previously cast by the voter. Based on whether the voter correctly identifies their previous ballots, the Voting server generates a noise ballot that either obfuscates the current voter ballot or obfuscates the most recent noise ballot cast by the Voting server.

The hallmark characteristic of our technique is therefore that it provides the voter with a counter-strategy to defend against a coercer: respond with a misidentification of previously cast ballots, which the coercer cannot know. We thus shift the last-vote-counts policy to a last-correct-vote-

counts policy. This allows us to implement our technique into a novel voting scheme, Loki, which evades last-minute coercion and supports flexible vote updating.

Contributions. The main contribution of this paper is a novel technique that enables flexible vote updating. To the best of our knowledge, no previously proposed counter-strategy that allows revoting can evade last-minute coercion. Another contribution of this work is Loki, a coercion-resistant Internet voting scheme that implements our counter-strategy to support flexible vote updating. We prove that Loki satisfies ballot privacy if the underlying ballot encryption scheme is Indistinguishable Relaxed Chosen Ciphertext Attack (IND-RCCA) [1] secure. We also prove that Loki satisfies coercion resistance under the DDH assumption and provides strong verifiability guarantees under the DLP assumption in the random oracle model. Loki is the first strongly verifiable revoting scheme providing protection against coercion even if the adversary is able to coerce a voter at the very last minute of the voting phase. Loki's tallying complexity is linear in the number of voters. Given the same trust assumptions, prior work can either provide revoting or last-minute coercion-resistance, but not both, and with more costly tallying based on the number of ballots. We provide a prototype implementation of Loki in Python showing that Loki key functions require only a few milliseconds to run on a MacBook Pro laptop.

2. Related work

Several voting schemes have provided strategies to achieve coercion-resistance and public verifiability. Civitas [2] implements Juels et al.'s (JCJ) coercion-resistant voting scheme [3]. Although the scheme is coercion-resistant, it does not provide a strategy for evading coercion in case of revoting, namely, a voter cannot deny revoting to a coercer. Efficiency improvements to Juels et al.'s scheme have been proposed [4], [5] but none of them supports deniable revoting. Achenbach et al. [6] extends Juels et al.'s scheme to achieve deniable revoting. More recently, *VoteAgain* [7] advanced an efficient way to achieve deniable revoting. However, both VoteAgain and Achenbach et al. strategies cannot address last-minute coercion as they assume that the coercer must give up control over the voter before the polls close to allow the voter to revote. In contrast, we show how one can remove such an assumption. Password-based voting schemes [8], [9] that use panic passwords as counter-strategies for fake credentials are coercion-resistant and allow revoting, but ballot overwriting or re-cast are exposed hence they are not deniable. Other approaches allow for revoting [10], [11], but the revoting strategy is either deniable or publicly verifiable.

Various definitions of coercion-resistance for voting schemes have been proposed [12]–[14], including epistemic [15] and game-based [7], [16]–[18] approaches, which have already been used to analyse several voting protocols [19]–[23]. *Caveat Coercitor* [24] introduces a relaxed version of coercion-resistance, namely, coercion attempts can be detected. In this approach, a voter changing their mind and overwriting their ballots would be signaled as a coercion attempt. Our work, in contrast, aims to the non-relaxed versions of coercion-resistance game-based notions. The notions of individual verifiability and universal verifiability have been extensively studied in voting [25]–[29]. Interestingly, Cortier and Lallemand [30] have shown that intuitively contrasting properties such as verifiability and privacy are somehow linked. They demonstrated that a voting scheme that does not meet individual verifiability fails to achieve vote privacy, when one considers the same trust assumptions. Schemes [7], [31] and definitions [18], [30], [32] that address weak trust assumptions, such as accounting for malicious bulletin boards and voters not checking their ballots, have been proposed. Our work considers verifiability notions by Cortier et al. [18] in which a voting scheme is *strongly* verifiable if registration authority and bulletin board are not simultaneously dishonest. However, Hirschi et al. [33] have recently shown that it is still necessary to assume that the bulletin board presents the same content to all readers. Therefore, we consider a voting scheme strongly verifiable if registration authority and tally servers are not simultaneously dishonest. Conversely, a voting scheme is *weakly* verifiable if both registration authority and tally servers are honest.

3. A new technique for flexible vote updating

3.1. In a nutshell

Our technique considers a *Voter*, who cast vote ballots and a *Voting server*, which is in charge of collection and publishing valid ballots in a trusted append-only bulletin board. For each voter, the bulletin board publishes a *cast ballot record* (CBR), which is a list of ballots associated with the voter. All the ballots included in a CBR are either generated by the voter or by the Voting server on behalf of the voter. In particular, the Voting server periodically casts *noise ballots* that are indistinguishable from the voter's ballots and serve to *obfuscate* the voter's casting behaviour and guarantee deniable vote updating to the voter.

The key idea behind our technique is that the voter can eventually signal to the Voting server which ballots are cast by the voter while being coerced. To do so, voter and Voting server keep track of a *list of indexes* that refers to the voter's CBR. The list contains the indexes of the ballots cast by the voter according to their CBR. Thus, when the voter casts a ballot, this also includes (an encryption of) the list of indexes. If a voter has already voted then, under coercion, they will need to provide an incorrect list of indexes, otherwise they will expose their "misbehaviour" to the coercer. This prompts the Voting server to add a noise ballot that "cancels" the ballot cast under coercion. This is possible because voter and the Voting server are the only two entities who exactly know which ballots in the voter CBR are cast under coercion and which ones are not.

Our technique can be seen as a mix of fake-credential and deniable revoting approaches. The fake-credential ap-

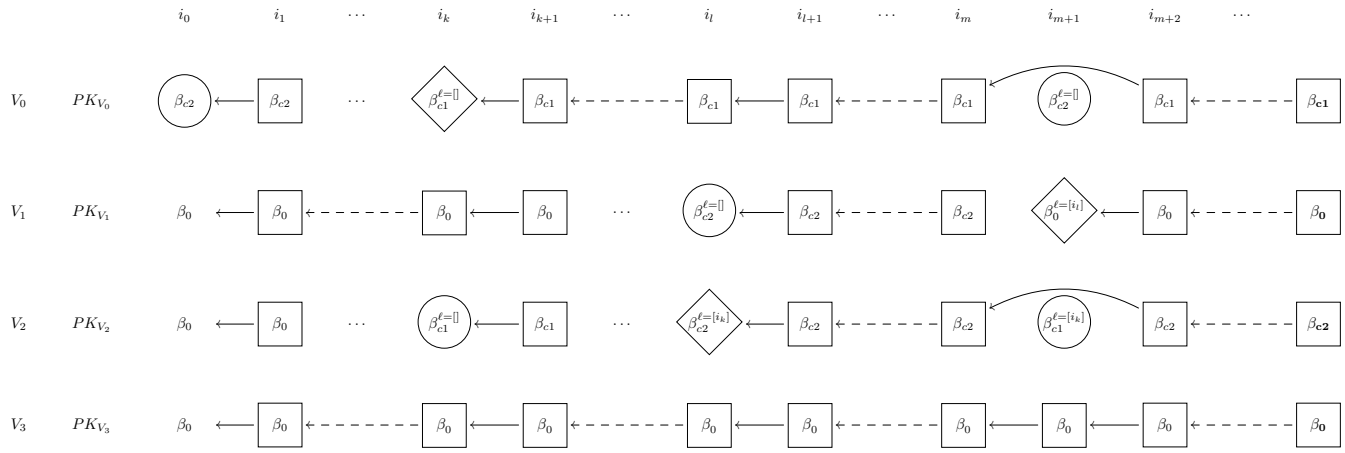

Figure 1: Our technique in practice. Each CBR is a row in this figure. Ballots cast by a voter under coercion are in circles, while coercion-free cast ballot are in diamond. Ballots cast by Voting server are in squares, and an arrow indicates which ballot is obfuscated. The CBR for V_0 exhibits an example of coercion during registration and at the last minute, in which the voter cannot revote. Both CBRs for for V_1 and V_2 exhibit examples of voters being able to revote after being coerced. The CBR for V_3 exhibits an example of voter abstention. The last column shows the ballots that are considered for tallying.

proach requires a coercion-free registration phase to set up real and fake credentials. This is because both real and fake credentials are fixed and do not change once they have been created. Our technique does not need such an assumption because the list of indexes is not fixed but dynamically changes according to the voter cast behavior. The deniable revoting approach instead requires that the voter revotes after coercion. Our technique does not rely on such an assumption because the voter can signal to the Voting server coercion as soon as this happens.

3.2. Description

Figure 1 deptics our technique in practice. At registration, a *registration* ballot, which includes the voter's public credentials, initialises the voter CBR. An authority or the voters themselves can cast such ballot. Note that the voter can even be coerced at registration and can reveal the content of the registration ballot to the coercer, without needing to lie. During the voting phase, the CBR is continuously populated with ballots generated either by the voter or by the Voting server. We require that the noise ballots generated by the Voting server are indistinguishable from the voter ballots. Let us consider β_v^ℓ as a ballot which contains a voter's vote for a candidate v. The voter generate this ballot including a list ℓ of indexes into the voter's CBR to signal if the ballot was coerced or not, and sends it to the Voting server. As a response, the Voting server appends the voter ballot to the voter's CBR and checks whether ℓ correctly identifies *all* ballots (if any) previously cast by the voter. If so, the Voting server appends to the voter's CBR an additional noise ballot which encrypts the same vote contained in the ballot just cast by the voter. Otherwise, an incorrect ℓ is signalling coercion, therefore the Voting server appends a noise ballot which encrypts the same vote contained in the *second-to-last* ballot stored in the CBR. Since i) a voter ballot is always followed by at least one noise ballot cast by the Voting server, and ii) the Voting server periodically obfuscates the last ballot in each CBR, the second-last ballot is a noise ballot and guarantees to always be an obfuscation of a correct voter ballot, that is, the last ballot that the voter sent identifying the list ℓ correctly. In fact, all noise ballots appearing in the CBR contain the vote of the correct voter ballot.

Our technique allows us to define a quantitative measure of coercion-resistance, which illustrates that it protects against a brute force attack perpetrated by the coercer. In such a scenario, the coercer may attempt to guess the correct list of indexes ℓ by asking the voter to submit several ballots with different lists of indexes. The number of required attempts exhibits exponential growth. Assuming that the CBR contains m ballots, the coercer needs to guess the k out of m ballots. With growing m, it becomes increasingly difficult for the coercer to guess the index list ℓ that identifies the complete set of uncoerced ballots. More precisely, the coercer needs to submit $\sum_{k=0}^{m} \binom{m-k}{k} = f_{m+1}$ ballots to be sure to guess the correct sequence of indexes ℓ at least once, with f_{m+1} being the $(m+1)^{th}$ Fibonacci number. Note that such brute force attack would allow the coercer to update any previously cast votes. It does not help the coercer to guess whether the voter followed the coercer instructions or not.

The design of our technique is based on the fact that Voting server and voter are the only parties who know exactly the voter's casting behaviour, thus it can be applied if the following criteria are satisfied.

1) The Voting server is trusted for coercion resistance.
2) The voter can cast at least one uncoerced vote.
3) Voter authentication to the Voting server is inalienable.
4) An append-only bulletin board guarantees that every-

one sees the same data.

5) Voter's ballots and Voting server's noise ballots are identifiable to the voter/voting server and indistinguishable to anyone else.

Note that all the criteria outlined above but the last one are necessary conditions for achieving coercion-resistance in elections that offer deniable revoting in general [6]. The last criterion might be achieved in different ways. For example, in KTV-Helios [34] and VoteAgain [7], the Voting server casts *dummy ballots*, which are encryptions of zeros that obfuscate voter ballots. Soroush [35] proposes the idea in which the Voting server periodically re-encrypts the last ballot associated to the voter as noise ballots to achieve deniable vote updating. In BeleniosRF [31], the Voting server re-randomizes the ballot cast by a voter to achieve strong receipt-freeness.

4. Loki

Loki is divided into the classic three phases of an election system: registration, voting, and tallying. At registration, the tallying servers jointly generate public and private keys for the election. For each voter, the registration authority generates the respective key material and creates a CBR in the bulletin board. The registration authority initialises the CBR with the voter's public credentials and with a registration ballot β_0. The ballot contains a "0", which represents a null vote.

Let us consider ℓ_{id} as the list of indexes into the CBR of voter id. When the voter sends to the Voting server a new ballot including a new list of indexes ℓ_v, the Voting server appends the encrypted voter ballot to the voter's CBR. If ℓ_v correctly extends ℓ_{id}, the Voting server appends to the CBR another ballot, i.e. a noise ballot, which re-randomizes the voter ballot with correspondingly updated encryptions and proofs. Otherwise the Voting server appends to the CBR a ballot re-randomization of the second-to-last ballot stored in the CBR. Differently from the fake crendential approach, Loki requires voters to remember only small integer, if needed at all. In fact, a voter needs to remember the indexes of their previous ballots only if they want to change their vote.

Verifiability of the Voting server is guaranteed by a disjunctive non-interactive zero-knowledge proofs (NIZKP) of knowledge that takes in encryptions of both ℓ_v and ℓ_{id}, which are publicly available in the bulletin board, ensuring the correct operation of the Voting server without revealing the content of the list of indexes. For tallying, Loki supports homomorphic tallying and other techniques that are commonly used in cryptographic voting protocols.

4.1. Threat model

In addition to Voter and Voting server, Loki involves a Registration authority[1] and the Tally servers. The latter are in charge of tallying and publishing the final result of the election in the bulletin board. The tally servers form a k-out-of-t threshold encryption system trusted for ballot privacy and coercion resistance. The registration authority is trusted for ballot privacy and verifiability as it generates the keys for the voters, while the Voting server is only trusted for coercion resistance, namely, it should not prove to a coercer that the Voting server casts a noise ballot. However, any process performed by the Voting server on a voter's CBR is publicly verifiable.

We consider a computationally bounded adversary whose efforts are towards coercing some voters into casting ballots for a particular candidate or to abstain. As any other coercion-resistant revoting based scheme [3], [6], [7], we require the inalienability of voter authentication until the polls have closed, meaning that the adversary can neither eliminate nor duplicate a voter's mean of authentication. Inalienability of authentication means that the voter authenticates by an inherence-based factor (e.g. biometric authentication) and it is required to prevent a coercer to cast ballots unbeknownst to the voter. This is also known as the over-the-shoulder adversary [8] and it is a necessary condition to achieve coercion-resistance in elections that offer deniable revoting in general [6]. Inalienable authentication can be achieved today by using existing biometric cards like the ones deployed by Samsung, Mastercard, and EVM [37]–[39]. These systems already provide an infrastructure for key management.

Differently from other coercion-resistant schemes based on revoting, we do not need to assume that the voter can cast one more vote after being coerced. We also do not require the classic assumptions in fake-credential based schemes [3], [6], such as the need for voters to hide cryptographic keys, no coercion at registration, or the need for voters to lie convincingly when they are under coercion.

A comparision of Loki with other coercion-resistant schemes is shown in Table 1. For ballot privacy, all schemes require the majority of tally servers and registrars to be honest. Loki, VoteAgain, and Achenbach et al. achieve verifiability with an honest registration authority and dishonest Voting server and tally servers. This is known as *strong* verifiability [18]. Conversely, JCJ achieves only *weak* verifiability since it needs to trust registration authority and the tally servers. For coercion-resistance, all the schemes in Table 1 requires the majority of tally servers to be honest. JCJ, VoteAgain, and Loki additionally require some trust on registrar, polling authority and Voting server, respectively.

4.2. Cryptographic primitives

Exponential ElGamal encryption [40] and NIZKP are the only two cryptographic primitives needed in Loki.

Let λ be a security parameter. Let $\mathbb{G}$ be a cyclic group of prime order p generated by generator g. We denote the integers modulo p with $\mathbb{Z}_p$ and write $r \xleftarrow{\$} \mathbb{Z}_p$ for r being chosen uniformly from $\mathbb{Z}_p$. Loki is built on the ElGamal encryption scheme for group $\mathbb{G}$, with generator g of order

1. Loki includes this party for simplicity. However, it is not strictly required by our technique as voters themselves can cast the registration ballots and can also be under coercion while doing so (e.g., V_0 in Fig. 1).

TABLE 1: Comparision of different coercion-resistant voting schemes, with trusted parties for ballot privacy and coercion-resistance. Strong verifiability assumes that the registrar or the talliers are trusted. Weak verifiability assumes that both registrar and talliers are trusted. The number of ballots is denoted by n_b while the number of voters is denoted by n_v.

	JCJ [3]	Achenbach et al. [6]	VoteAgain [7]	**Loki**
Deniable revoting	No	Yes	Yes	Yes
Last-minute coercion resistance	Yes	No	No	Yes
Ballot privacy	Talliers + Registrar	Talliers + Registrar	Talliers + Polling authority†	Talliers + Registrar
Verifiability	Weak	Strong	Strong	Strong
Coercion-resistance	Talliers + Registrar‡	Talliers	Talliers + Polling authority	Talliers + Voting server
Complexity	n_b^2	n_b^2	$n_b \log n_b$	n_v
Inalienable authentication	Registration phase	Voting phase	Voting phase	Voting phase

†The original paper claims only talliers, but a privacy attack is possible with an untrusted polling authority [36].

‡The original paper assumes a trusted registrar, but it can be distributed to several parties as long as one of them is trusted [2].

p and message space $\mathbb{M} = g^b$, where $b = \{0, 1\}$ consisting of the following algorithms.

$\mathsf{EKeyGen}(1^\lambda)$, which, on input of security parameter 1^λ, outputs a pair of ElGamal decryption and encryption keys (sk, pk) where $sk \xleftarrow{\$} \mathbb{Z}_p$ and $pk = g^{sk}$.

$\mathsf{Enc}(pk, m; r)$, which, given a public key pk, a message $m \in \mathbb{M}$, and some randomness $r \xleftarrow{\$} \mathbb{Z}_p$, outputs a ciphertext $(c_1, c_2) = (g^r, m \cdot pk^r)$.

$\mathsf{Dec}(sk, ct = (c_1, c_2))$, which outputs $m = c_1^{-sk} \cdot c_2$.

$\mathsf{ReEnc}(pk, ct = (c_1, c_2); r)$, which, using randomness $r \xleftarrow{\$} \mathbb{Z}_p$, outputs the reencryption of ct: $(c_1 \cdot g^r, c_2 \cdot pk^r)$.

For verifiability, we use a system of NIZKP. To define our relation for verifying the correct construction of a ballot, we compose the following relations.

- *Proof of plaintext knowledge* is a NIZKP of knowledge π_e proving that ct is the correct encryption of a message $m \in \mathbb{M}$ using the public encryption key pk and randomness r known to the prover. The corresponding relation R_e is defined as follows:

  $((ct, pk, \mathbb{M}), (r, m)) \in R_e$ iff $ct = \mathsf{Enc}(pk, m; r) \wedge m \in \mathbb{M}$

- *Proof of correct decryption* is a NIZKP of knowledge π_d for proving that ct is decrypted by applying the private encryption key sk on ciphertext ct. The corresponding relation R_d is defined as follows:

  $((pk, m, ct), sk) \in R_d$ iff $m = \mathsf{Dec}(sk, ct) \wedge pk = g^{sk}$

- *Proof of private key knowledge* is a NIZKP of knowledge π_s for proving that a verifying upk is generated by the following algorithm.

  $\mathsf{SKeyGen}(1^\lambda)$ which, on input security parameter 1^λ, outputs a pair of keys (usk, upk) such that $upk = g^{usk}$. The corresponding relation R_s is defined as follows:

  $$(upk, usk) \in R_s \text{ iff } upk = g^{usk}$$

- *Proof of correct re-encryption* is a NIZKP of knowledge π_{re} for proving that ct' is a re-encryption of a ciphertext ct w.r.t. public encryption key pk and the prover knows the randomness r. The corresponding relation R_{re} is defined as follows:

  $$((pk, ct, ct'), r) \in R_{re} \text{ iff } ct' = \mathsf{ReEnc}(pk, ct; r)$$

We use disjunctive NIZKP as introduced by Cramer et al. [41]. Let $R = R_1 \vee \cdots \vee R_n$ for $n > 1$ such that $((x_1, \ldots, x_n), \omega) \in R$ being defined as follows:

$$(x = (x_1, \ldots, x_n), \omega) \in R \text{ iff } (x_1, \omega) \in R_1 \vee \cdots \vee (x_n, \omega) \in R_n$$

This construction enables a prover to demonstrate the knowledge of ω, such that $(x_i, \omega) \in R$, but without revealing i. This is captured by the following algorithms.

$\mathsf{DisjProof}(x, \omega)$, which on input public statement x and private witness ω of a disjunctive relation $R = R_1 \vee \cdots \vee R_n$outputs a NIZKP π that proves the knowledge of ω_j w.r.t x_j of relation R_j and simulates the proof of knowledge other relations.

$\mathsf{Verify}(x, \pi)$ which, on input public statement x and NIZKP π checks the proof π w.r.t. x of relation R and outputs $\top$ if the checks succeeds, otherwise return $\bot$.

The last building blocks of Loki are the discrete probability distributions $D_\mathcal{R}$ and $D_\mathcal{T}$, as introduced in KTV-Helios [17], which are used respectively to sample the number of ballots, both re-randomized and voter's ones, and to determine the time to cast each of them in the voting phase. Note that both distributions should not map any value to 0 otherwise no voters' ballots would be re-randomized. Also, while $D_\mathcal{R}$ can be a uniform distribution, $D_\mathcal{T}$ is a distribution that represents typical vote casting behaviour[2]. To avoid race conditions on cast ballots, the distributions generate sorted timestamps. Voters can distinguish their ballots from the noise ballots independently of casting frequency since they

2. If $D_\mathcal{T}$ is a uniform distribution, an adversary might be able to distinguish revoting from re-randomized ballots due to vote casting behaviour.

know the ciphertexts of their ballots on the bulletin board. The Voting server generates the distributions for each CBR before the voting phase begins. Since the number of required attempts to brute force a CBR exhibits exponential growth, the Voting server can set the number of noise ballots, which is sampled by $D_{\mathcal{R}}$, to a relatively small value.

4.3. List of indexes encoding

Before we present Loki, we first describe how one can encode a list of indexes so that it can be consistently updated as soon as new ballots are included into a voter CBR. To do so, we first associate to each ballot in the CBR a '0' if the ballot is a re-randomization by the Voting server, and a '1' if the ballot is generated by the voter. The given (binary) value is then stored as decimal value in the list of indexes. For example, let us assume a CBR with 5 ballots, that is, $[\beta_0, \beta_1, \beta_2, \beta_3, \beta_4]$, where β_2 is the only ballot generated by the voter while all the other ballot are re-randomizations generated by the Voting server. The list of indexes associated to the last ballot, i.e. $\ell_{(\beta_4)}$ is then represented by $(0, 0, 1, 0, 0)$, that is, $\ell_{(\beta_4)} = 4$. If a new ballot re-randomization is added to the CBR, then the list of indexes is updated to $(0, 0, 1, 0, 0, 0)$, that is, $\ell_{(\beta_5)} = 8$. Otherwise $\ell_{(\beta_5)} = 9$ because the voter generates the last ballot. Thus, every new ballot re-randomization sets the value of the list of indexes $\ell_{(\beta_i)}$ to the twice of the previous value, i.e. $\ell_{(\beta_i)} = 2 \cdot (\ell_{(\beta_{i-1})})$ while a new ballot generated by the voter sets the new value to $\ell_{(\beta_i)} = 2 \cdot (\ell_{(\beta_{i-1})}) + 1$.

If a list of indexes ℓ is encrypted using ElGamal, i.e., $ct^{\ell} := (c_1, c_2) = (g^r, g^{\ell} \cdot pk^r)$ then its value can be updated as outlined above either it is a new ballot re-randomization or it is a ballot generated by the voter. In the first case, $ct_i^{\ell} := (ct_{i-1}^{\ell})^2 = ({c_1}^2, {c_2}^2)$; in the latter case, $ct_i^{\ell} := ({c_1}^2, g \cdot {c_2}^2)$.

4.4. Formal description of Loki

Figure 2 describes the algorithms defining Loki. The scheme is organized in the following three phases.

Registration phase: The algorithm $\mathsf{Setup}(1^{\lambda}, \mathbb{I}, \mathbb{V}) \rightarrow ((pk_T, sk_T), (pk_{vs}, sk_{vs}))$ allows, respectively, tallying servers and the Voting server to generate the tallying key pair (pk_T, sk_T)[3] and the Voting server key pair (pk_{vs}, sk_{vs}). The $\mathcal{BB}$ is initialised with the lists of candidates $\mathbb{V}$ and eligible voters $\mathbb{I}$. The registration authority registers the voter id to the election by running $\mathsf{Register}(id) \rightarrow (L_{id}, (usk, upk))$. It sets $\ell_v = \ell_{id} = 0$ and initialises the CBR list with β_0 so that $L_{id} = [id, upk, \overline{ct}_0]$, where $\overline{ct}_0 = (ct^0, ct^{\ell_0}, ct^{\ell_{id}}, r)$, $ct^0 = \mathsf{Enc}(pk_T, 0; r)$ and $ct^{\ell_0} = ct^{\ell_{id}} = \mathsf{Enc}(pk_{vs}, 0; r)$. The Voting server finally executes $\mathsf{Append}(\mathcal{BB}, \beta_0)$).

Voting phase: The voter encrypts and generates the proof for their ballots. The Voting server validates the ballots and appends them to the bulletin board. It also appends re-randomized ballots, i.e. noise ballots, based on the list of indexes provided in the voter ballot. We use disjunctive NIZKP to make the ballots generated by the voter indistinguishable from the noise ballots generated by the Voting server. Our disjunctive NIZKP π proves that $(x, \omega) \in R$, where $R = R^{\text{id}} \vee R^{\text{pred}} \vee R^{\text{pred2}}$ where R^{id}, R^{pred}, and R^{pred2} are defined as in Figure 2. This phase proceeds as follows.

- The Voting server does not receive a ballot from the voter id. Then, based on the probability distributions $D_{\mathcal{T}}$ and $D_{\mathcal{R}}$, the Voting server computes $\mathsf{Obfuscate}(\mathcal{BB}, sk_{vs}, id) \rightarrow \mathcal{BB}$, which appends a ballot that re-randomizes the last element in L_{id} at time t. In this case, the Voting server proves $(x, \omega) \in R^{\text{pred}}$ and simulates the other two relations.
- The voter generates a new ballot by $\mathsf{Vote}(id, usk, pk_T, pk_{vs}, v, \ell_v) \rightarrow \beta$, where $\beta = (id, upk, \overline{ct})$ and $\overline{ct} = (ct^v, ct^{\ell_v}, ct^{\ell_{id}}, \pi)$, and sends it to the Voting server through an authenticated channel. In this case, the voter's π proves that $(x, \omega) \in R^{\text{id}}$ and simulates the other two relations. The Voting server first appends $\overline{ct}$ to $\mathcal{BB}$ by $\mathsf{Append}(\mathcal{BB}, \beta) \rightarrow \mathcal{BB}$, and then it computes $\mathsf{Obfuscate}(\mathcal{BB}, sk_{vs}, id)$, which generates a ballot β'. If $\mathsf{Dec}(sk_{vs}, ct_{i-1}^{\ell_v}) = \mathsf{Dec}(sk_{vs}, ct_{i-1}^{\ell_{id}})$, then β' re-randomizes the vote in β. The Voting server proves $(x, \omega) \in R^{\text{pred}}$ and simulates the other two relations. Otherwise, it re-randomizes the vote encrypted in the ballot before β, namely, it re-encrypts the vote in the last re-randomization in L_{id}. In this case, the Voting server proves $(x, \omega) \in R^{\text{pred2}}$ and simulates the other two relations. The ballot β' contains updated and re-randomized values of both ℓ_v and ℓ_{id}.

The voter runs $\mathsf{VerifyVote}(\mathcal{BB}, upk, usk, \beta) \rightarrow \perp/\top$ to check that their ballot is included in the bulletin board.

Tallying phase: the tallying servers take the last ballot from each CBR, add the corresponding encryption for each candidate, and decrypt the results with proof of correctness. The tallying servers execute $\mathsf{Tally}(\mathcal{BB}, sk_T) \rightarrow (R, \Pi)$. Anyone can verify the result R of tallying by executing $\mathsf{VerifyTally}(\mathcal{BB}, R, \Pi)$, which checks Π w.r.t. $\mathcal{BB}$ and R.

5. Ballot privacy

Our ballot privacy definition is based on the game-based definition by Bernhard et al. [17], [42]. The indistinguishability game tracks on two bulletin boards, in which only one bulletin board is available to a probabilistic polynomial time (PPT) adversary $\mathcal{A}$. The goal of the adversary, who controls the Voting server and a subset of voters, is to distinguish whether the given result comes from the bulletin board accessible to $\mathcal{A}$ or not. Therefore, such definition ensures that one cannot learn anything about the individual voter's vote more than one can learn from the election result alone, which is derived from the public election information included on the bulletin board.

The ballot privacy experiment $\mathbf{Exp}_{ES,\mathcal{A}}^{BP,b}(\lambda, \mathbb{I}, \mathbb{V})$ is described in Algorithm 1. The adversary $\mathcal{A}$ has access to $\mathcal{BB}_b$ and can query the following oracles:

3. The secret key is generated in a distributed way, thus no single Tallying server learns the key.

- $\mathsf{Setup}(1^\lambda, \mathbb{I}, \mathbb{V}) \rightarrow ((pk_T, sk_T), (pk_{vs}, sk_{vs}))$: on input of the security parameter 1^λ, electoral roll $\mathbb{I}$, and candidate list $\mathbb{V}$ computes $(pk_T, sk_T) \xleftarrow{\$} \mathsf{EKeyGen}(1^\lambda)$ and $(pk_{vs}, sk_{vs}) \xleftarrow{\$} \mathsf{EKeyGen}(1^\lambda)$.
- $\mathsf{Register}(id) \rightarrow (L_{id}, (usk, upk))$: on implicit input $(pk_T, \mathbb{I}, \mathbb{V})$, and voter identity $id \in \mathbb{I}$, do the following.
  1) Compute $(usk, upk) \xleftarrow{\$} \mathsf{SKeyGen}(1^\lambda)$ to create a public and secret key pair (usk, upk) for the voter.
  2) Compute $ct^0 \xleftarrow{\$} \mathsf{Enc}(pk_T, 0; r)$ and $ct^{\ell_0} \xleftarrow{\$} \mathsf{Enc}(pk_{vs}, 0; r)$.
  3) Set $ct^{\ell_{id}} = ct^{\ell_0}$ to generate the voter initial ballot $\beta_0 = (id, upk, \overline{ct}_0)$ where $\overline{ct}_0 = (ct^0, ct^{\ell_0}, ct^{\ell_{id}}, r)$. Return $(L_{id}, (usk, upk))$, where $L_{id} = [id, upk, \overline{ct}_0]$ is the initial CBR list for the registered voter with identity id.
- $\mathsf{Vote}(id, usk, pk_T, pk_{vs}, v, \ell_v) \rightarrow \beta$: on input voter identity $id \in \mathbb{I}$, voter secret key usk, the tallier public key pk_T, the Voting server public key pk_{vs}, vote option $v \in \mathbb{V}$, list of indexes $\ell_v \in \mathbb{Z}_p$, implicit input $(\mathbb{V}, \mathbb{I})$ and L_{id} of length $i-1$, where $\overline{ct}_{i-1} = (ct^v_{i-1}, ct^{\ell_v}_{i-1}, ct^{\ell_{id}}_{i-1}, \pi_{i-1})$ is the last ballot on L_{id}, do the following.
  1) Set $ct_i = (ct^{\ell_{id}}_{i-1})^2$.
  2) Compute $ct^v \xleftarrow{\$} \mathsf{Enc}(pk_T, v; r^v)$ $ct^{\ell_v} \xleftarrow{\$} \mathsf{Enc}(pk_{vs}, \ell_v; r^{\ell_v})$ $ct^{\ell_{id}} \xleftarrow{\$} \mathsf{ReEnc}(pk_{vs}, g \cdot (ct_i); r^{\ell_{id}})$.
  3) Run $\pi \xleftarrow{\$} \mathsf{DisjProof}(x, \omega)$, where $x = (L_{id}, pk_T, pk_{vs}, \mathbb{V}, (ct^v, ct^{\ell_v}, ct^{\ell_{id}}))$ and $\omega = (usk, (r^v, v), (\ell_v, r^{\ell_v}), r^{\ell_{id}})$ s.t.
$$(x, \omega) \in R_i^{\text{id}} \text{ iff } ct^v = \mathsf{Enc}(pk_T, v; r^v) \wedge v \in \mathbb{V} \wedge ct^{\ell_v} = \mathsf{Enc}(pk_{vs}, \ell_v; r^{\ell_v}) \wedge$$
$$ct^{\ell_{id}} = \mathsf{ReEnc}(pk_{vs}, g \cdot ct_i; r^{\ell_{id}}) \wedge upk_{id} = g^{usk_{id}}$$
  where $R_i = R_i^{\text{id}} \vee R_i^{\text{pred}} \vee R_i^{\text{pred2}}$, $i \geq 1$ and $\overline{ct}_{-1} = \overline{ct}_0$.
  4) Set $\overline{ct} = (ct^v, ct^{\ell_v}, ct^{\ell_{id}}, \pi)$. Return the ballot $\beta = (id, upk, \overline{ct})$ as ith ballot of L_{id}.
- $\mathsf{Validate}(\mathcal{BB}, \beta) \rightarrow \top/\bot$: on input a ballot $\beta = (id, upk, \overline{ct})$, where $\overline{ct} = (ct^v, ct^{\ell_v}, ct^{\ell_{id}}, \pi)$, and implicit input $(L_{id}, pk_T, pk_{vs}, \mathbb{I}, \mathbb{V})$ checks that i) $id \in \mathbb{I}$, ii) β does not already appear in $\mathcal{BB}$, and iii) $\top \leftarrow \mathsf{Verify}(x, \pi)$. If any of the checks fail, it returns $\bot$ otherwise $\top$.
- $\mathsf{Append}(\mathcal{BB}, \beta) \rightarrow \mathcal{BB}$: on implicit input t from $D_{\mathcal{T}}$, and on input a ballot $\beta = (id, upk, \overline{ct})$ checks that $\mathsf{Validate}(\mathcal{BB}, \beta) = \top$. If so, it updates $\mathcal{BB}$ at time t by extending the voter CBR list L_{id} with $\overline{ct}$.
- $\mathsf{VerifyVote}(\mathcal{BB}, upk, usk, \beta) \rightarrow \bot/\top$: on input a ballot $\beta = (id, upk, \overline{ct})$, checks that $\overline{ct}$ is in L_{id} on $\mathcal{BB}$ and that $\mathsf{Validate}(\mathcal{BB}, \beta) = \top$. If any of the checks fail, it returns $\bot$ otherwise $\top$.
- $\mathsf{Obfuscate}(\mathcal{BB}, sk_{vs}, id) \rightarrow \beta$: on input $\mathcal{BB}$, id, and sk_{vs}, obtains L_{id} of length $i-1$, where $\beta_{i-1} = (id, upk, \overline{ct}_{i-1})$ is the last ballot on L_{id}, and $\overline{ct}_{i-1} = (ct^v_{i-1}, ct^{\ell_v}_{i-1}, ct^{\ell_{id}}_{i-1}, \pi_{i-1})$. Then, do the following.
  1) Set $ct_i = (ct^{\ell_{id}}_{i-1})^2$. Compute $ct^{\ell_v} \xleftarrow{\$} \mathsf{ReEnc}(pk_{vs}, ct_i; r^{\ell_v})$ and $ct^{\ell_{id}} \xleftarrow{\$} \mathsf{ReEnc}(pk_{vs}, ct_i; r^{\ell_{id}})$.
  2) If $\mathsf{Dec}(sk_{vs}, ct^{\ell_v}_{i-1}) = \mathsf{Dec}(sk_{vs}, ct^{\ell_{id}}_{i-1})$, then compute $ct^v \xleftarrow{\$} \mathsf{ReEnc}(pk_T, ct^v_{i-1}; r^v)$ and run $\pi \xleftarrow{\$} \mathsf{DisjProof}(x, \omega)$, where $x = (L_{id}, pk_T, pk_{vs}, \mathbb{V}, (ct^v, ct^{\ell_v}, ct^{\ell_{id}}))$ and $\omega = (sk_{vs}, r^v, r^{\ell_v}, r^{\ell_{id}})$ s.t.
$$(x, \omega) \in R_i^{\text{pred}} \text{ iff } ct^v = \mathsf{ReEnc}(pk_T, ct^v_{i-1}; r^v) \wedge ct^{\ell_v} = \mathsf{ReEnc}(pk_{vs}, ct_i; r^{\ell_v}) \wedge$$
$$ct^{\ell_{id}} = \mathsf{ReEnc}(pk_{vs}, ct_i; r^{\ell_{id}}) \wedge \mathsf{Dec}(sk_{vs}, ct^{\ell_v}_{i-1}) = \mathsf{Dec}(sk_{vs}, ct^{\ell_{id}}_{i-1})$$
  3) Else compute $ct^v \xleftarrow{\$} \mathsf{ReEnc}(pk_T, ct^v_{i-2}; r^v)$ and run $\pi \xleftarrow{\$} \mathsf{DisjProof}(x, \omega)$, where $x = (L_{id}, pk_T, pk_{vs}, \mathbb{V}, (ct^v, ct^{\ell_v}, ct^{\ell_{id}}))$ and $\omega = (sk_{vs}, r^v, r^{\ell_v}, r^{\ell_{id}})$ s.t.
$$(x, \omega) \in R_i^{\text{pred2}} \text{ iff } ct^v = \mathsf{ReEnc}(pk, ct^v_{i-2}; r^v) \wedge ct^{\ell_v} = \mathsf{ReEnc}(pk_{vs}, ct_i; r^{\ell_v}) \wedge$$
$$ct^{\ell_{id}} = \mathsf{ReEnc}(pk_{vs}, ct_i; r^{\ell_{id}}) \wedge \mathsf{Dec}(sk_{vs}, ct^{\ell_v}_{i-1}) \neq \mathsf{Dec}(sk_{vs}, ct^{\ell_{id}}_{i-1})$$
  where $R_i = R_i^{\text{id}} \vee R_i^{\text{pred}} \vee R_i^{\text{pred2}}$, $i \geq 1$ and $\overline{ct}_{-1} = \overline{ct}_0$.
  4) Set $\overline{ct} = (ct^v, ct^{\ell_v}, ct^{\ell_{id}}, \pi)$ and $\beta = (id, upk, \overline{ct})$ and run $\mathsf{Append}(\mathcal{BB}, \beta) \rightarrow \mathcal{BB}$.
- $\mathsf{Tally}(\mathcal{BB}, sk_T) \rightarrow (R, \Pi)$: on input $\mathcal{BB}$ and the decryption key sk_T computes the election result as follows. Let N be the number of voters, let $\{ct_x\}_{x=1}^N$ be the encrypted vote of the last ballots from each CBR in $\mathcal{BB}$, and let $ct_x = \{ct_i\}_{i=1}^{|\mathbb{V}|}$ where ct_i denotes the encrypted vote for the candidate $v_i \in \mathbb{V}$.
  1) Compute $T_i = \prod_{x=1}^N ct_x^i$. The tally t_i for candidate v_i is produced by decrypting T_i with the key sk_T.
  2) Compute the result $R = (t_1, \ldots, t_{|\mathbb{V}|})$ and Π, a Fiat-Shamir proof of correct decryption. Output (R, Π).
- $\mathsf{VerifyTally}(\mathcal{BB}, (R, \Pi)) \rightarrow \bot/\top$: on input $\mathcal{BB}$, the tally result (R, Π), verifies the correctness of (R, Π) on $\mathcal{BB}$. If any of the checks fails, it returns $\bot$ otherwise $\top$.

Figure 2: The algorithms defining Loki

$\mathbf{Exp}^{BP,b}_{ES,\mathcal{A}}(\lambda, \mathbb{I}, \mathbb{V})$:
$((pk_T, sk_T), (pk_{vs}, sk_{vs})) \leftarrow \mathsf{Setup}(\lambda, \mathbb{I}, \mathbb{V})$
Initialize $\mathcal{BB}_0$, $\mathcal{BB}_1$
$b' \leftarrow \mathcal{A}^{\mathcal{O}}(pk_T, \mathbb{I}, (pk_{vs}, sk_{vs}))$
return b'

$\mathcal{O}$voteLR(id, v_0, v_1) :
if $v_0 \notin \mathbb{V}$ or $v_1 \notin \mathbb{V}$ or $id \notin \mathbb{I}$ **then return** $\perp$
$\beta_{v_0} \leftarrow \mathsf{Vote}(id, pk, usk_{id}, v_0, \ell)$
$\beta_{v_1} \leftarrow \mathsf{Vote}(id, pk, usk_{id}, v_1, \ell)$
if $\mathsf{Validate}(\mathcal{BB}_b, \beta_{v_b}) = \perp$ **then return** $\perp$
$\mathcal{BB}_0 \leftarrow \mathsf{Append}(\mathcal{BB}_0, \beta_{v_0})$
$\mathcal{BB}_1 \leftarrow \mathsf{Append}(\mathcal{BB}_1, \beta_{v_1})$

$\mathcal{O}$cast$(\beta_{\mathcal{A}})$:
if $\mathsf{Validate}(\mathcal{BB}_b, \beta_{\mathcal{A}}) = \perp$ **then return** $\perp$
$\mathcal{BB}_0 \leftarrow \mathsf{Append}(\mathcal{BB}_0, \beta_{\mathcal{A}})$
$\mathcal{BB}_1 \leftarrow \mathsf{Append}(\mathcal{BB}_1, \beta_{\mathcal{A}})$

$\mathcal{O}$board() :
return $\mathcal{BB}_b$

$\mathcal{O}$tally() :
$(R, \Pi_0) \leftarrow \mathsf{Tally}(\mathcal{BB}_0, sk_T)$
$\Pi_1 \leftarrow \mathsf{SimTally}(\mathcal{BB}_1, R)$
return (R, Π_b)

Algorithm 1: The ballot privacy experiment $\mathbf{Exp}^{BP,b}_{ES,\mathcal{A}}(\lambda, \mathbb{I}, \mathbb{V})$, in which the adversary $\mathcal{A}$ has access to oracles $\mathcal{O} = \{\mathcal{O}\text{voteLR}, \mathcal{O}\text{cast}, \mathcal{O}\text{board}, \mathcal{O}\text{tally}\}$. The adversary can call $\mathcal{O}$tally only once.

- $\mathcal{O}$voteLR, which allows the adversary $\mathcal{A}$ to simulate honest voters. The voter id casts a ballot β_{v_0} for the candidate $v_0 \in \mathbb{V}$ in $\mathcal{BB}_0$ and a ballot β_{v_1} for the candidate $v_1 \in \mathbb{V}$ in $\mathcal{BB}_1$ provided that $\mathsf{Validate}(\mathcal{BB}_b, \beta_{v_b}) = \top$, which verifies the input of the oracle $\mathcal{O}$voteLR and β_{v_b} with respect to $\mathcal{BB}_b$. The ballot β_{v_b} is appended to $\mathcal{BB}_b$.
- $\mathcal{O}$cast, which allows the adversary to cast a ballot $\beta_{\mathcal{A}}$ on behalf of any upk_i. The ballot $\beta_{\mathcal{A}}$ is appended to both bulletin boards as long as $\mathsf{Validate}(\mathcal{BB}_b, \beta_{v_b}) = \top$.
- $\mathcal{O}$board, which allows $\mathcal{A}$ to see the public content of a bulletin board depending on the bit b in the experiment.
- $\mathcal{O}$tally, which allows the adversary to see the voting result R and the proof of correct tallying Π_b over the last votes at the end of the voting phase. The result R is returned by tallying $\mathcal{BB}_0$ in both experiments, and the proof is simulated when $b = 1$. The adversary can call the oracle $\mathcal{O}$tally only once.

The adversary $\mathcal{A}$ outputs a guess b' at the end of the game. We say that the adversary wins the game if $b' = b$.

Definition 1. *Ballot privacy. Let ES = (*Setup, Register, Vote, Validate, Append, VerifyVote, Obfuscate, Tally, VerifyTally*) be an election scheme for a candidate list $\mathbb{V}$, and security parameter λ. We say that ES meets ballot privacy if there exists a simulation algorithm* SimTally *such that for any PPT adversaries $\mathcal{A}$:*

$$|Pr[\mathbf{Exp}^{BP,0}_{ES,\mathcal{A}}(\lambda, \mathbb{I}, \mathbb{V}) = 1] - Pr[\mathbf{Exp}^{BP,1}_{ES,\mathcal{A}}(\lambda, \mathbb{I}, \mathbb{V}) = 1]|$$

is a negligible function in the security parameter λ.

5.1. Loki satisfies ballot privacy

In Loki, the (dishonest) Voting server generates noise ballots by re-randomizing the honest voter's ballots. Informally, a voting system is susceptible of replay attacks if it allows an adversary to copy a legitimate voter's ballot from the bulletin board and then can submit the ballot as the adversary's own. This results in a violation of ballot privacy. Loki is designed to prevent manipulation of the data beyond re-randomization of the voter's ballot by the adversary (Voting server). Since Loki's ballot encryption scheme consists of ElGamal encryption and NIZKP, which can only be malleable for randomization (by Voting server as an adversary), we can prove ballot privacy by showing that Loki is IND-RCCA secure.

Theorem 1. *Loki achieves ballot privacy if the underlying ballot encryption scheme is IND-RCCA secure.*

Proof: We define a sequence of games to show that our scheme provides ballot privacy. In these games, the adversary $\mathcal{A}$ interacts with a challenger in the ballot privacy experiment, starting with $b = 0$ and ending up with $b = 1$. Each game has a bulletin board in which the adversary can see the ballots. The ballots on the bulletin board are generated by $\mathcal{O}$voteLR and $\mathcal{O}$cast. The Voting server (i.e., the adversary) can generate noise ballots for the voter CBR list using $\mathcal{O}$board to get the voter ballots from $\mathcal{BB}$, then re-randomizing them using sk_{vs}, and finally casting the noise ballots via $\mathcal{O}$cast. The result is calculated on the last ballots by $\mathcal{O}$tally. The $\mathcal{O}$voteLR simulates the potential vote of an honest voter in our scheme. We show that the advantage of the adversary $\mathcal{A}$ in distinguishing the transition over the sequence of games is negligible in the security parameter.

Let $(upk_i, (v_i^0, \beta_i^0), (v_i^1, \beta_i^1))$ be the last query and output of the oracle $\mathcal{O}\text{voteLR}(i, v_i^0, v_i^1)$ for each voter with identity i, where $\beta_i^b \leftarrow \mathsf{Vote}(i, pk, usk_i, v_b, \ell)$ is the encrypted ballot (or the noise ballot of an encrypted ballot) corresponding to the vote v_i^b in the bulletin board $\mathcal{BB}_b$ $(b = 0, 1)$. For each $\mathcal{O}$cast query, we have $\beta_i^0 = \beta_i^1$ with respect to $v_i^0 = v_i^1$. Recall that in our scheme, we use NIZKP for the ballot and the tally proofs. These proofs can be simulated under the programmable random oracle.

Game G_1. Let G_1 be the game corresponding to $\mathbf{Exp}^{BP,b}_{ES,\mathcal{A}}(\lambda, \mathbb{I}, \mathbb{V})$ for $b = 0$. In this case, $\mathcal{A}$ sees $\mathcal{BB}_0$ where the output of the oracle $\mathcal{O}$tally() is computed on $\mathcal{BB}_0$.

Game G_2. Let G_2 be the same as G_1, except that all proofs made by honest parties are simulated. In this game, the output of tally proof Π_0 in G_1 is also simulated by $\mathsf{SimTally}(\mathcal{BB}_0, R)$. If the adversary $\mathcal{A}$ can distinguish this game from G_1 then we can use $\mathcal{A}$ to construct an adversary that breaks the zero-knowledge property of the proof system.

In the following games, we make incremental changes to replace ballots not cast by the adversary $\mathcal{A}$ in the bulletin board $\mathcal{BB}_0$ with the corresponding ballots in $\mathcal{BB}_1$.

Game G_3. Let $G_3 = \{G_2^i\}_{i=1}^n$ be a sequence of games, with n being the number of the last ballots cast by $\mathcal{O}$voteLR in $\mathcal{BB}_0$ and $\mathcal{BB}_1$. Let G_2^1 be the same game as the G_2 with the only difference that the last ballot β_1^1, instead of β_1^0, is placed in $\mathcal{BB}_0$ for upk_1. For $i > 1$, G_2^i is the same game as G_2^{i-1} except that β_i^1, instead of β_i^0, is now placed in $\mathcal{BB}_0$. We prove that the advantage of $\mathcal{A}$ to distinguish these sequence of the games, namely, $G_2^1, G_2^2, \ldots, G_2^n$ is negligible. We reduce the advantage of an adversary $\mathcal{B}$ against IND-RCCA to the advantage of the adversary $\mathcal{A}$ in distinguishing G_2^i and G_2^{i-1}. The IND-RCCA is essentially the same as IND-CCA2, with one key difference: the decryption oracle returns $\top$ whenever it is queried to decrypt any ciphertext that decrypts to $m \in \{m_0, m_1\}$, where $\{m_0, m_1\}$ is the challenge plaintext. In the IND-RCCA game, it is not advantageous for the adversary to generate new ciphertexts that can decrypt to the plaintext of the challenge ciphertext. Let $\mathcal{B}$ be the adversary who is given the challenge ciphertext ct^* corresponding to (v_i^0, v_i^1) in the IND-RCCA security reduction game. The adversary $\mathcal{B}$ in the reduction game returns $\beta_i = (i, upk, \overline{ct})$ as a last ballot for upk_i, such that $\overline{ct} = (ct^*, ct^\ell, ct^{\ell_{id}}, \pi)$. In our ballot privacy game, the challenger decrypts the ballots returned by $\mathcal{O}$cast, where the adversary can generate a ballot on behalf of a corrupted voter $upk_j \neq upk_i$ or can re-randomize the honest voter's ballot. The corrupted voters' ballots may contain the encryption of challenge (v_i^0, v_i^1). The decryption oracle in the security reduction game will return $\top$ for these ciphertexts. To solve this problem, the adversary $\mathcal{B}$ modifies the ciphertext of ballot with $upk_j \neq upk_i$ as follows: Let $ct = (c_1, c_2)$ be the adversary's ciphertext $ct \in (j, upk, \overline{ct}_j)$ where $j \neq i$. The adversary $\mathcal{B}$ modifies ct to the ciphertext $ct' = (c_1, c_2 \cdot pk_T^{H(upk_j)})$ and forwards ct' to the IND-RCCA decryption oracle, which returns $v_i \cdot pk_T^{H(upk_j)}$. The adversary $\mathcal{B}$ then extracts and returns v_i as a result of decryption in the ballot privacy game to the adversary $\mathcal{A}$. Note that in the reduction phase, the ciphertext of a ballot cast by $\mathcal{A}$ that contains upk_i is forwarded to the decryption oracle. If the decryption oracle returns $\perp$, the adversary $\mathcal{B}$ response with v_0.

The adversary $\mathcal{B}$ simulates a distinguishing game between G_2^{i-1} and G_2^i for the adversary $\mathcal{A}$ as follows:

- Answer the query of $\mathcal{O}$voteLR(j, v_j^0, v_j^1) by simulating the right vote of the query, namely, (j, v_j^1), for the voter upk_j where $j < i$, until the ballots of upk_{i-1} have been generated.
- Generate a ballot for upk_i which is an answer to the query to $\mathcal{O}$voteLR(i, v_0, v_1), using the IND-RCCA challenge oracle on (v_i^0, v_i^1). Let ct^* be the challenge ciphertext corresponding to (v_i^0, v_i^1) in the IND-RCCA security game. The adversary $\mathcal{B}$ returns $\beta_i = (i, upk, \overline{ct})$ as a last ballot for upk_i, such that $\overline{ct} = (ct^*, ct^\ell, ct^{\ell_{id}}, \pi)$
- Continue simulating the game for the voter upk_j where $j > i$ and use (j, v_j^0) as the vote of upk_j from now on.
- Tally all honest (simulated) ballots including the ballot with ct^* based on the left votes of $\mathcal{O}$voteLR(i, v_0, v_1). The votes of the ballots returned by $\mathcal{O}$cast (the adversary's ballot) in the tally phase are counted by querying the decryption oracle on the adversary's vote ciphertexts. All adversary's votes together with the honest left votes generate the tally result (R, Π), where the tally proof Π is simulated as in G_2.
- Send $\mathcal{A}$'s guess bit b to the IND-RCCA challenger.

In the IND-RCCA security game, if the challenger chose $b = 0$ (i.e., the challenge ciphertext $ct^* \in \beta_i$ corresponds to v_0), then $\mathcal{B}$ perfectly simulates G_2^{i-1} for $\mathcal{A}$. If $b = 1$, then it simulates G_2^i. The NIZKP protocol ensures that the adversary $\mathcal{A}$ cannot manipulate the ballot β_i beyond re-randomization due to its knowledge soundness. Furthermore, the soundness of the NIZKP does not allow $\mathcal{A}$ to create a valid proof for a ballot containing the re-randomized ct^* with $upk_j \neq upk_i$. Hence, the adversary $\mathcal{B}$ breaks the IND-RCCA security game with the same advantage as the adversary $\mathcal{A}$ distinguishes the games G_2^i and G_2^{i-1}.

Game G_4. Let G_4 be the last game in G_3, namely, G_2^n in which the view of the adversary $\mathcal{A}$ of the bulletin board $\mathcal{BB}_0$ is switched with $\mathcal{BB}_1$. The advantage of the adversary $\mathcal{A}$ in distinguishing the sequence of these games is equal to the advantage of the adversary $\mathcal{B}$ in the IND-RCCA security game. The game G_4 is equal to $\mathbf{Exp}^{BP,1}$ except that all ballot proofs of honest voters are generated by a zero-knowledge simulator. Thus, the game G_4 and $\mathbf{Exp}^{BP,1}$ are indistinguishable due to the zero-knowledge property. Since our ballot encryption scheme, consisting of ElGamal encryption plus disjunctive NIZKP construction, is only malleable for re-randomization by the adversary, then the scheme is IND-RCCA secure. Therefore,

$$|Pr[\mathbf{Exp}^{BP,0}_{ES,\mathcal{A}}(\lambda, \mathbb{I}, \mathbb{V}) = 1] - Pr[\mathbf{Exp}^{BP,1}_{ES,\mathcal{A}}(\lambda, \mathbb{I}, \mathbb{V}) = 1]|$$

is negligible in the security parameter λ.

In Appendix A we show that Loki satisfies strong-consistency and strong-correctness.

6. Verifiability

We use the verifiability definition introduced in Cortier et al. [18] to prove that Loki achieves verifiability with an honest registration authority and dishonest Voting server and tally servers. This is known as strong verifiability [18]. Therefore, we assume that the registration authority and the voting device are trusted, the honest voter secret key is not leaked to the adversary, and that the adversary can corrupt a subset of voters. According to [18], a voting scheme with $|C|$ corrupted voters is called strongly verifiable if the election result reflects the votes of 1) all honest voters who checked their votes, which appear on the bulletin board; 2) at most the $|C|$ voters who are controlled by the adversary; 3) the voters who have not checked their ballots.

Let $\mathbb{I}$ be a list of the identity of the voters and $\mathbb{U}$ be a list of the voter secret and public information. To model strong verifiability, we apply the experiment $\mathbf{Exp}^{ver}_{ES,\mathcal{A}}$ [18] as depicted in Algorithm 2. Let

$C = \{(id_1^c, v_1^c, *), (id_2^c, v_2^c, *), \ldots, (id_{n_c}^c, v_{n_c}^c, *)\}$ be the votes of the voters who did not cast a valid ballot after being corrupted. The votes of the honest voters who have checked their ballots on the bulletin board is denoted by $H = \{(upk_1^h, v_1^h, *), (upk_2^h, v_2^h, *), \ldots, (upk_n^h, v_n^h, *)\}$. The tuples $(id^h, v^h, *)$, denotes the last votes of the voter $id \in \mathbb{U}$. If the voter has submitted multiple votes and checked all of them, we only include in H the last vote checked by the voter. The honest voters who never have checked their ballots are denoted by $H' = \{(upk_1^{h'}, v_1^{h'}, *), (upk_2^{h'}, v_2^{h'}, *), \ldots, (upk_n^{h'}, v_n^{h'}, *)\}$. The adversary $\mathcal{A}$ has access to the oracles outlined below, which are part of $\mathbf{Exp}_{ES,\mathcal{A}}^{ver}$ as in Algorithm 2.

- $\mathcal{O}$register(id), which allows the adversary to get a list of registered voters. It generates (id, usk, upk, β_0), returns (id, upk, β_0), and updates $\mathbb{U} \cup (id, upk, usk, \beta_0)$.
- $\mathcal{O}$corrupt(id), which allows the adversary to obtain $(id, usk,\ upk)$ given a registered voter id. It checks that $id \in \mathbb{I}$, then returns (id, usk, upk) and updates $C \cup (id, usk, upk)$.
- $\mathcal{O}$vote(id, v), which generate a ballot β. If $id \in C$ or $id \notin \mathbb{U}$ or $v \notin \mathbb{V}$, then it returns 0, otherwise it returns $\beta \leftarrow \mathsf{Vote}(id, usk_{id}, pk_T, pk_{vs}, v, \ell)$. It then replaces any previous tuple $(id, v', \beta') \in H \cup H'$ with β and v so that $(id, v, \beta) \in H \cup H'$.

Definition 2. *Verifiability. Let ES = (*Setup*,* Register*,* Vote*,* Validate*,* Append*,* VerifyVote*,* Obfuscate*,* Tally*,* VerifyTally*) be an election scheme for an electoral roll $\mathbb{I}$, candidate list $\mathbb{V}$, and security parameter λ. We say that ES is verifiable if the advantage of any PPT adversary $\mathcal{A}$ such that $Pr[\mathbf{Exp}_{ES,\mathcal{A}}^{ver}(\lambda, \mathbb{I}, \mathbb{V}) = 1]$ is negligible in the security parameter λ.*

6.1. Loki satisfies strong verifiability

In Loki, the Voting server generates re-randomized ballots from the ballot cast by the voter. Given that the i) registration authority is honest, ii) the discrete logarithm problem assumption holds, and iii) the knowledge-soundness of NIZKP, the re-randomization of the voter's ballot does not change the voter's vote. Thus, the voting server can only generate new copies of the ballots that have been cast by the voter. Loki provides partial homomorphic tallying hence the total number of votes cast for a candidate can be calculated by adding the decrypting results of $\mathcal{BB}_1$ to $\mathcal{BB}_2$ disjointly, such that $\mathcal{BB} = \mathcal{BB}_1 \cup \mathcal{BB}_2$. Moreover, for each $\mathcal{BB}_i$ the tallying phase generates a proof of correctness of tallying.

Theorem 2. *Loki is strongly verifiable under the hardness of Discrete Logarithm Problem (DLP) in the random oracle model.*

Proof: Let the adversary $\mathcal{A}$ output a set of votes, the result R and the corresponding proof Π at the tally phase. The last ballots from the voter CBRs form a set i.e., $T = \{\beta_1, \beta_2, \ldots, \beta_n\}$. The set of the last ballots T, the result R, and the proof Π of valid decryption are published on the $\mathcal{BB}$. The homomorphic property of ElGamal and the soundness of proof Π verifies that the result R is obtained from the correct decryption of $\Pi_{i=1}^{T} ct_i^v$ where $ct_i^v \in \beta_i$. We can conclude that $\mathsf{VerifyTally}(R, \Pi)$ only returns $\top$, when R is the correct result of $T = \{\beta_1, \beta_2, \ldots, \beta_n\}$ on $\mathcal{BB}$.

Let $\beta_i \in T$ be the last ballot of L_i such that $\beta_i = (i, upk, \overline{ct}_i)$ where $\overline{ct}_i = (ct_i^v, ct_i^{\ell_v}, ct_i^{\ell_{id}}, \pi_i)$ and $\mathsf{Validate}(\mathcal{BB}, \beta_i) = \top$. We now prove that the ballot β_i is the re-randomized ballot of one of the following sets:

- $H = \{(upk_1^h, v_1^h, *), (upk_2^h, v_2^h, *), \ldots, (upk_{n_h}^h, v_{n_h}^h, *)\}$ the last votes of the honest voters who have checked their ballots.
- $H' = \{(upk_1^{h'}, v_1^{h'}, *), (upk_2^{h'}, v_2^{h'}, *), \ldots, (upk_m^{h'}, v_m^{h'}, *)\}$ the votes of the honest voters who have not checked their ballots.
- $C = \{(upk_1^c, v_1^c, *), (upk_2^c, v_2^c, *), \ldots, (upk_{n_c}^c, v_{n_c}^c, *)\}$ the last votes of the corrupted voters.

The knowledge soundness of the proof π_i on the ballot β_i ensures that β_i is generated by the voter or the Voting server.

$\mathbf{Exp}_{ES,\mathcal{A}}^{ver}(\lambda, \mathbb{I}, \mathbb{V})$:
$((pk_T, sk_T), (pk_{vs}, sk_{vs})) \leftarrow \mathsf{Setup}(\lambda, \mathbb{I}, \mathbb{V})$
$(\mathcal{BB}, R, \Pi) \leftarrow \mathcal{A}^{\mathcal{O}}(sk_{vs}, sk_T)$
if $\mathsf{VerifyTally}(\mathcal{BB}, (R, \Pi)) = \bot$ **then return** $\bot$
if $R = \bot$ **then return** $\bot$
$H = \{(upk_i^h, v_i^h, *)\}_{i=1}^{n_h}$ and $H' = \{(upk_i^{h'}, v_i^{h'}, *)\}_{i=1}^{n_{h'}}$
if $\exists \{v_i^c\}_{i=1}^{n_c}$ such that $0 \leq n_c \leq |C|$,
$\exists \{(upk_i^{h'}, v_i^{h'}, *)\}_{i=1}^{t} \subseteq H'$ such that
$R = \rho(\{v_i^h\}_{i=1}^{n_h}) + \rho(\{v_i^c\}_{i=1}^{n_c}) + \rho(\{v_i^{h'}\}_{i=1}^{t})$ **then**
  return $\bot$
else
  return $\top$
end if

$\mathcal{O}$register(id) :
if $id \notin \mathbb{U}$ **then**
  $(\beta_0, (usk_{id}, upk_{id})) \leftarrow \mathsf{Register}(id)$
  $\mathbb{U} \leftarrow \mathbb{U} \cup (L_{id}, (usk_{id}, upk_{id}))$
  return (id, upk, β_0)
else
  return $\bot$
end if

$\mathcal{O}$corrupt(id) :
if $id \in \mathbb{I}$ and $id \notin H \cup H'$ **then**
  return $C = C \cup (id, upk, usk)$
else
  return $\bot$
end if
$\mathcal{O}$vote(id, v) :
if $id \in C$ or $id \notin \mathbb{U}$ or $v \notin \mathbb{V}$ **then return** $\bot$
$\beta \leftarrow \mathsf{Vote}(id, usk_{id}, pk_T, pk_{vs}, v, \ell)$
$H \cup H' \leftarrow \mathsf{Update}(H \cup H', (upk_{id}, v, \beta))$

Algorithm 2: The verifiability experiment $\mathbf{Exp}_{ES,\mathcal{A}}^{ver}(\lambda, \mathbb{I}, \mathbb{V})$, in which the adversary $\mathcal{A}$ has access to the oracles $\mathcal{O} = \{\mathcal{O}\text{register}, \mathcal{O}\text{corrupt}, \mathcal{O}\text{vote}\}$.

Since the Voting server is only allowed to re-randomize the ballot cast by the voter, and the secret key of the voter is needed for generating a new encrypted vote, β_i must contain either the vote of an honest voter in H or H', or a vote of a corrupted voter in C. In Loki, the registration ballot contains a null vote, which is generated by the registration authority. Hence, β_i can be a null vote if a voter does not cast a vote or check their ballot.

Let the ballot β_i be the last ballot of the voter i in the CBR L_i. Let us assume that voter i voted only once with a list of indexes ℓ_v and checked their ballot with the function VerifyVote. Since generating a ballot requires the voter's secret key usk, then β_i must be the re-randomization of the voter (checked) ballot. The Voting server can re-randomize the voter's ballot based on the lists of indexes in ct^{ℓ_v} and $ct^{\ell_{id}}$. It provides a proof π_i on β_i which ensures that β_i has been generated by re-randomizing the voter's ballot if $\ell_v = \ell_{id}$ or the second-last ballot of the CBR L_i otherwise. The knowledge soundness of the disjoint proof π_i ensures that β_i is the re-randomization of the voter's last ballot even in the case that the voter voted multiple times with valid lists of indexes and then checked their ballots. Furthermore, the Voting server cannot manipulate the voter's vote by re-randomizing any ballot of L_i as π_i guarantees that the Voting server re-randomized either the last ballot of L_i or the second-last ballot of L_i. Assuming a trusted registration authority, the hardness of DLP, and the knowledge soundness of NIZKP, the adversary $\mathcal{A}$ cannot generate a new ballot for honest voters in Loki and cannot corrupt more voters than $|C|$. The adversary $\mathcal{A}$ cannot drop the ballots that have been checked by honest voters without being noticed otherwise $\mathcal{A}$ wins the game. The adversary $\mathcal{A}$ cannot re-randomize any ballot in L_i rather than the ballots cast by voters with valid lists of indexes. This proves that the result R output by $\mathsf{Tally}(\mathcal{BB}, sk_T)$ corresponds to the result of the tally function ρ computed on all last votes by honest voters who checked their ballots, by at most $|C|$ votes cast by corrupted voters, and by a subset of votes cast by honest voters who did not check their ballots.

The registration ballot that initialises the voter CBR list is also verifiable. Let $L_{id} = [id, upk, \overline{ct}_0]$ be the initial CBR list for the voter with identity id, where $\overline{ct}_0 = (ct^0, ct^{\ell_0}, ct^{\ell_{id}}, r)$. Anyone can verify that the ballot ciphertext $(ct^0, ct^{\ell_0}, ct^{\ell_{id}})$ using the encryption randomness r.

7. Coercion resistance

Our coercion-resistance definition is inspired by the receipt-free definition in Bernhard et al. [17]. As in former coercion-resistance definitions, we assume that the adversary is not monitoring the interactions of voters while they are not coerced. We assume that Voting server, bulletin board, and tally servers are trusted for coercion resistance. To model coercion resistance, we define the experiment $\mathbf{Exp}^{CR,b}_{ES,\mathcal{A}}$ as depicted in Algorithm 3, for a bit b, and election system ES and a PPT adversary $\mathcal{A}$ using four oracles.

$\mathcal{O}$voteLR allows a voter id to cast a ballot β_v for candidate $v \in \mathbb{V}$ on $\mathcal{BB}_0$. The function Append adds the ballot

$\mathbf{Exp}^{CR,b}_{ES,\mathcal{A}}(\lambda, \mathbb{I}, \mathbb{V})$:
$((pk_T, sk_T), (pk_{vs}, sk_{vs})) \leftarrow \mathsf{Setup}(\lambda, \mathbb{I}, \mathbb{V})$
Initialize $\mathcal{BB}_0$, $\mathcal{BB}_1$
$\{((upk_{id}, usk_{id}), L_{id}) \leftarrow \mathsf{Register}(id)\}_{id\in\mathbb{I}}$
$\{\mathcal{BB}_0 \leftarrow \mathsf{Append}(\mathcal{BB}_0, L_{id})\}_{id\in\mathbb{I}}$
$\{\mathcal{BB}_1 \leftarrow \mathsf{Append}(\mathcal{BB}_1, L_{id})\}_{id\in\mathbb{I}}$
$(upk_j, v_{\mathcal{A}}) \leftarrow$
$\mathcal{A}(\{(upk_{id}, L_{id})\}_{id\in\mathbb{I}}, pk_T, pk_{vs}, \{usk_{id}\}_{id\in C})$
if $v_{\mathcal{A}} \notin \mathbb{V}$ or $upk_j \notin \{upk_{id}\}_{id\in\mathbb{I}}$ or
$upk_j \in \{upk_{id}\}_{id\in C}$ **then**
  return $\perp$
else

$b' \leftarrow \mathcal{A}^{\mathcal{O}}(\{(upk_{id})\}_{id\in\mathbb{I}}, pk_T, pk_{vs}, usk_j, \{usk_{id}\}_{id\in C})$
end if
return b'

$\mathcal{O}$voteLR(id, v) :
if $v \notin \mathbb{V}$ or $(upk_{id}, L_{id}) \notin \{(upk_{id}, L_{id})\}_{id\in\mathbb{I}}$ **then**
return $\perp$
$\beta_v \leftarrow \mathsf{Vote}(id, usk_{id}, pk_T, pk_{vs}, v, \ell)$
$\beta \leftarrow \mathsf{Obfuscate}(\mathcal{BB}_1, sk_{vs}, id)$
$\mathcal{BB}_0 \leftarrow \mathsf{Append}(\mathcal{BB}_0, \beta_v)$
$\mathcal{BB}_1 \leftarrow \mathsf{Append}(\mathcal{BB}_1, \beta)$
$\mathcal{BB}_0 \leftarrow \mathsf{Append}(\mathcal{BB}_0, \mathsf{Obfuscate}(\mathcal{BB}_0, sk_{vs}, id))$
$\mathcal{BB}_1 \leftarrow \mathsf{Append}(\mathcal{BB}_1, \mathsf{Obfuscate}(\mathcal{BB}_1, sk_{vs}, id))$

$\mathcal{O}$cast$(id, \beta_{\mathcal{A}})$:
if $\mathsf{Validate}(\mathcal{BB}_0, \beta_{\mathcal{A}}) = \perp$ **then return** $\perp$
if $\mathsf{Validate}(\mathcal{BB}_1, \beta_{\mathcal{A}}) = \perp$ **then return** $\perp$
if $id \in C$ **then**
  $\beta' \leftarrow \mathsf{Obfuscate}(\mathsf{Append}(\mathcal{BB}_0, \beta_{\mathcal{A}}), sk_{vs}, id)$
else
  $\beta' \leftarrow \mathsf{Obfuscate}(\mathcal{BB}_0, sk_{vs}, id)$
end if
$\mathcal{BB}_0 \leftarrow \mathsf{Append}(\mathsf{Append}(\mathcal{BB}_0, \beta_{\mathcal{A}}), \beta')$
$\mathcal{BB}_1 \leftarrow \mathsf{Append}(\mathcal{BB}_1, \beta_{\mathcal{A}})$
$\mathcal{BB}_1 \leftarrow \mathsf{Append}(\mathcal{BB}_1, \mathsf{Obfuscate}(\mathcal{BB}_1, sk_{vs}, id))$

$\mathcal{O}$board() :
return $\mathcal{BB}_b$

$\mathcal{O}$tally() :
$(R, \Pi_0) \leftarrow \mathsf{Tally}(\mathcal{BB}_0, sk_T)$
$\Pi_1 \leftarrow \mathsf{SimTally}(\mathcal{BB}_1, R)$
return (R, Π_b)

Algorithm 3: The coercion resistance experiment $\mathbf{Exp}^{CR,b}_{ES,\mathcal{A}}(\lambda, \mathbb{I}, \mathbb{V})$, in which the adversary $\mathcal{A}$ has access to the oracles $\mathcal{O} = \{\mathcal{O}\text{voteLR}, \mathcal{O}\text{cast}, \mathcal{O}\text{board}, \mathcal{O}\text{tally}\}$. $\mathcal{O}$tally is called only once by $\mathcal{A}$, at the end of voting.

β_v to $\mathcal{BB}_0$. Assuming a coerced voter, the oracle $\mathcal{O}$voteLR ensures deniable vote updating using the function Vote on $\mathcal{BB}_0$ and Obfuscate on $\mathcal{BB}_1$. The ballot β_v submitted by the coerced voter is appended to $\mathcal{BB}_0$ as a new ballot, while the obfuscated ballot β is appended to $\mathcal{BB}_1$. The function Obfuscate generates noise ballots on both bulletin boards to

ensure deniable vote updating.

$\mathcal{O}$cast allows the adversary to cast a ballot $\beta_{\mathcal{A}}$ on behalf of a voter id. The ballot $\beta_{\mathcal{A}}$ is appended to both bulletin boards followed by noise ballots generated by the function Obfuscate. The latter, based on its input, adds a new noise ballot to each of the bulletin boards. Thus, considering the coerced voter upk_j, $\mathcal{O}$cast models the situation in which the voter upk_j evades coercion on $\mathcal{BB}_0$ (i.e., the noise ballot on $\mathcal{BB}_0$ is not related to $\beta_{\mathcal{A}}$), versus the situation in which the voter follows the instructions of the coercer (i.e., the noise ballot on $\mathcal{BB}_1$ is related to $\beta_{\mathcal{A}}$). The adversary $\mathcal{A}$ can generate a ballot $\beta_{\mathcal{A}} = \mathsf{Vote}(j, pk_T, pk_{vs}, usk_j, v_{\mathcal{A}}, \ell_{\mathcal{A}})$ for a candidate $v_{\mathcal{A}} \in \mathbb{V}$ using the secret information usk_j of the coerced voter. $\mathcal{O}$board allows the adversary to see $\mathcal{BB}_b$. $\mathcal{O}$tally allows the adversary to see the result R and proof Π_b. The result R is returned by tallying the last ballots on $\mathcal{BB}_0$ in both experiments, and the proof is simulated by SimTally when $b = 1$. The key differences between our coercion-resistance definition and the receipt-freeness one in [17] is reflected by the $\mathcal{O}$cast and $\mathcal{O}$Receipt oracles. In our $\mathcal{O}$cast, the adversary generates the ballot, hence the voter does not know the candidate chosen by the adversary. Conversely, $\mathcal{O}$Receipt in [17] requires the adversary to tell the voter the name of the candidate the voter has to provide a receipt for, hence the voter both generates and casts the ballot for the adversary. Moreover, our definition allows the adversary to force the voter to cast the adversary's ballot at the last minute of the election. In [17] this is impossible as the voter has to nullify the adversary ballot in advance.

We now define an indistinguishability game between the adversary $\mathcal{A}$ and the voter. The game tracks both bulletin boards $\mathcal{BB}_0$ and $\mathcal{BB}_1$. $\mathcal{BB}_0$ contains the ballots of all voters who have not been coerced as well as the ballots of voters who successfully resisted coercion, i.e. voters who either revoted before or after being coerced in a deniable way. $\mathcal{BB}_1$ instead contains the ballots of voters who submitted to the coercer, i.e. voters who neither voted nor revoted after being coerced. Each bulletin board also contains the ballots that were cast by corrupted voters ($id \in C$) on behalf of $\mathcal{A}$. The adversary $\mathcal{A}$ has only access to one of these bulletin boards, based on the bit b of the experiment. Both bulletin boards contain the same number of ballots regardless of the value of bit b. The result of the election is always computed on the bulletin board $\mathcal{BB}_0$. The adversary $\mathcal{A}$ has also access to all oracles while trying to guess b with which the experiment was initialized. To this end, the adversary outputs a bit b' and wins the game if $b' = b$.

Definition 3. *Coercion resistance. Let ES = (*Setup, Register, Vote, Validate, Append, VerifyVote, Obfuscate, Tally, VerifyTally*) be an election scheme for an electoral roll $\mathbb{I}$, candidate list $\mathbb{V}$, and security parameter λ. We say that ES is* coercion resistant *if there exists an algorithm* SimTally *such that for any PPT adversaries $\mathcal{A}$:*

$$|Pr[\mathbf{Exp}^{CR,0}_{ES,\mathcal{A}}(\lambda, \mathbb{I}, \mathbb{V})=1] - Pr[\mathbf{Exp}^{CR,1}_{ES,\mathcal{A}}(\lambda, \mathbb{I}, \mathbb{V})=1]|$$

be a negligible function $neg(\lambda)$ in the security param. λ.

7.1. Loki satisfies coercion resistance

Coercion resistance means that a coercer should not be able to determine the behavior of the coerced voter, that is, the coercer cannot tell whether a coerced voter submitted to coercion for the given result of the election. In Loki, a coercer should not be able to determine from the election result whether a list of indexes ℓ_v provided by the voter is valid or invalid. Furthermore, a coercer should not be able to determine whether a registered voter voted or abstained from voting in the voting phase.

Theorem 3. *Loki provides coercion resistance under the DDH assumption in the random oracle model.*

Proof: Similarly to the proof of ballot privacy, we define a sequence of games to show that Loki provides coercion resistance. As in the ballot privacy proof, the result R is computed on $\mathcal{BB}_0$ and $\mathsf{SimTally}(\mathcal{BB}_b, R)$ simulates the proof of correct decryption in tally. In Loki, the situation in which the voter evades last-minute coercion versus the situation in which they do not, is captured by the list of indexes ℓ_v. The adversary $\mathcal{A}$ generates a ballot $\beta_{v_{\mathcal{A}}}$, which is cast by the coerced voter, where $\beta_{v_{\mathcal{A}}} = (j, upk, \overline{ct}_j)$ and $\overline{ct}_j = (ct^{v_{\mathcal{A}}}, ct^{\ell_{v_{\mathcal{A}}}}, ct^{\ell_{id}}, \pi_j)$. The Voting server re-randomizes the last-second ballot of the coerced voter upk_j if $\mathsf{Dec}(sk_{vs}, ct^{\ell_v}) \neq \mathsf{Dec}(sk_{vs}, ct^{\ell_{id}})$ by executing $\mathsf{Obfuscate}(\mathcal{BB}, sk_{vs}, id)$ (defined in Section 4.4), which generates noise ballots based on the voter ballot. Based on the probability distributions $D_{\mathcal{T}}$ and $D_{\mathcal{R}}$, the noise ballots are appended on $\mathcal{BB}$ by the function Append. This is captured in $\mathbf{Exp}^{CR,0}_{ES,\mathcal{A}}$ by the oracle $\mathcal{O}$cast. The case of *last-minute coercion* is captured by calling the oracles $\mathcal{O}$voteLR and $\mathcal{O}$cast respectively, while the case of *coercion then revoting* is captured by calling the oracles $\mathcal{O}$cast and $\mathcal{O}$voteLR respectively. Deniable vote updating is captured in Loki by $\mathcal{O}$voteLR. The oracle $\mathcal{O}$voteLR appends the voter ballot β_v on $\mathcal{BB}_0$ and a noise ballot generated by $\mathsf{Obfuscate}(\mathcal{BB}_1, sk_{vs}, j)$ on $\mathcal{BB}_1$.

Game G_1. This game is similar to an election run in which the voter upk_j evades the coercion and provides an invalid list of indexes to the adversary such that $\ell_{v_{\mathcal{A}}} \neq \ell_{id}$, or the voter revotes despite being instructed by the coercer to abstain from revoting. This is equivalent to $\mathbf{Exp}^{CR,b}_{ES,\mathcal{A}}$ with $b = 0$. Hence, the content of the bulletin board in this game, which is the view of the adversary $\mathcal{A}$ in G_1, is equal to $\mathcal{BB}_0$. The tally result is equal to (R, Π_0), which is the output of $\mathcal{O}$tally of the experiment $\mathbf{Exp}^{CR,0}_{ES,\mathcal{A}}$. The coerced voter upk_j can evade coercion in the following cases.

- *Last-minute coercion*: the coerced voter upk_j casts the ballot β_v *before* the adversary's ballot $\beta_{v_{\mathcal{A}}}$ for a candidate $v_{\mathcal{A}}$, which is cast as a last-minute ballot. In this case, the CBR contains the initial ballot β_0 and the ballots generated by calling the oracles $\mathcal{O}\mathsf{voteLR}(j, v)$ and $\mathcal{O}\mathsf{cast}(j, \beta_{v_{\mathcal{A}}})$ respectively based on the probability distributions $D_{\mathcal{T}}$ and $D_{\mathcal{R}}$. Thus, the last ballot of the coerced voter's CBR is a randomized form of β_v and $\beta_{v_{\mathcal{A}}}$, respectively on $\mathcal{BB}_0$ and $\mathcal{BB}_1$. Note that the adversary

can call $\mathcal{O}$voteLR several times to cast new votes or to generate a re-randomized ballot for β_v with vote v.
For instance, let us assume that the voter does not change their vote v. The bulletin board $\mathcal{BB}_0$ contains L_j such that $L_j = [j, upk, \overline{ct}_0, \overline{ct}_v, \overline{ct}_v^{r_2}, \ldots, \overline{ct}_v^{r_k}, \overline{ct}_{v_\mathcal{A}}, \overline{ct}_v^{r_{k+1}}]$. Let $\overline{ct}_v^{r_i}$ denote the re-randomized ballot for the ballot $\beta_v = (id, upk, \overline{ct}_v)$. In Loki, the noise ballot is generated by Obfuscate based on the probability distributions $D_\mathcal{T}$ and $D_\mathcal{R}$. Similarly, the content of L_j on the $\mathcal{BB}_1$ is $L_j = [j, upk, \overline{ct}_0, \overline{ct}_0^{r_1}, \ldots, \overline{ct}_0^{r_k}, \overline{ct}_{v_\mathcal{A}}, \overline{c}_{v_\mathcal{A}}^{r_{k+1}}]$.

- *Coercion then revoting*: the voter upk_j casts the ballot β_v *after* the adversary's ballot $\beta_{v_\mathcal{A}}$. In this case, the CBR contains the the adversary's ballot $\beta_{v_\mathcal{A}}$ followed by the noise ballots generated by Obfuscate and the voter ballot β_v. This is captured in $\mathbf{Exp}^{CR,0}_{ES,\mathcal{A}}$ by calling the oracles $\mathcal{O}$cast$(j, \beta_{v_\mathcal{A}})$ and $\mathcal{O}$voteLR(j, v) respectively. For instance, the content of L_j on the $\mathcal{BB}_0$ is $L_j = [j, upk, \overline{ct}_0, \overline{ct}_{v_\mathcal{A}}, \overline{ct}_0^{r_1}, \ldots, \overline{ct}_0^{r_{k-1}}, \overline{ct}_v, \overline{ct}_v^{r_{k+1}}]$ while the content of L_j on $\mathcal{BB}_1$ is L_j=$[j, upk, \overline{ct}_0, \overline{ct}_{v_\mathcal{A}}, \overline{ct}_{v_\mathcal{A}}^{r_1}, \ldots, \overline{ct}_{v_\mathcal{A}}^{r_{k-1}}, \overline{ct}_{v_\mathcal{A}}^{r_k}, \overline{ct}_{v_\mathcal{A}}^{r_{k+1}}]$.

Game G_2. This game is equal to G_1, but with all the zero-knowledge proofs in the ballots and tally being replaced by simulations. The output of the tally proof Π_0 in the game G_1 is replaced by the output of $\mathsf{SimTally}(\mathcal{BB}_0, R)$. The games G_1 and G_2 are indistinguishable because the proof has the zero-knowledge property and can be indistinguishably simulated using the programmable random oracle. Hence, $\mathcal{A}$ has negligible advantage to distinguish G_2 from G_1.

In the following games, for each voter upk_i, we will replace step by step all last ballots on the $\mathcal{BB}_0$, which rely on $b = 0$, with the corresponding last ballots on the $\mathcal{BB}_1$. Given a tuple $(upk_i, \beta_i^0, \beta_i^1, v_i^0, v_i^1)$ where $\beta_i^0 \in (\mathcal{BB}_0, upk_i)$ and $\beta_i^1 \in (\mathcal{BB}_1, upk_i)$, β_i^0 and β_i^1 are swapped if $\beta_i^0 \neq \beta_i^1$. Therefore, the output ballots of the oracles $\mathcal{O}$voteLR and $\mathcal{O}$cast are swapped except for the adversary ballot $\beta_{v_\mathcal{A}}$. In particular, we show the advantage of $\mathcal{A}$ through these transitions. For instance, the last randomized ballot of the voter upk_j, $\overline{ct}_v^{r_{k+1}}$ on $\mathcal{BB}_0$ is replaced with the randomization of $\beta_{v_\mathcal{A}}$, i.e. $\overline{ct}_{v_\mathcal{A}}^{r_{k+1}}$, in the last-minute coercion case.

In the following game, we prove the indistinguishability between games that only replace the last vote ciphertext of upk_j on $\mathcal{BB}_0$ with the corresponding vote ciphertext on $\mathcal{BB}_1$, which can be generalised to any (last) ballot on $\mathcal{BB}_0$.

Game G_3. This game is as G_2, but the last output for the voter upk_j of $\mathcal{O}$cast in the last-minute coercion case, resp., $\mathcal{O}$voteLR in the coercion-then-revoting case, are swapped. In the last-minute coercion case, the last ballot in the CBR list L_j on $\mathcal{BB}_0$ is swapped with the last ballot in L_j on $\mathcal{BB}_1$. In the coercion-then-revoting case, the vote ciphertext $ct^v \in \overline{ct}_v^{r_{k+1}}$ in L_j, which is the re-randomized form of the ballot generated by $\mathsf{Vote}(j, usk_j, pk_T, pk_{vs}, v, \ell)$ on $\mathcal{BB}_0$, is swapped with the vote ciphertext $ct_{v_\mathcal{A}}^{r_{k+1}} \in \overline{ct}_{v_\mathcal{A}}^{r_{k+1}}$, which is generated by Obfuscate on $\mathcal{BB}_1$.

Assuming that the Voting server is trusted, the ballot encryption scheme based on ElGamal and NIZKP, is NM-CPA secure under the DDH assumption in the random oracle model [42]. NM-CPA is the same as IND-CCA2 but the adversary cannot query the decryption oracle adaptively [40]. We prove that the adversary $\mathcal{A}$ has a negligible advantage in distinguishing G_2 and G_3 in terms of the security parameter.

We provide a reduction to show that the advantage of an adversary $\mathcal{A}'$ against the NM-CPA security game can be reduced to the advantage of the adversary $\mathcal{A}$ against distinguishing between G_2 and G_3. In Loki, $\beta = (id, upk, \overline{ct})$ where $\overline{ct} = (ct^v, ct^{\ell_v}, ct^{\ell_{id}}, \pi)$. The ciphertext ct^v is an encryption of vote v with public key pk_T, ct^{ℓ_v} and $ct^{\ell_{id}}$ are the encryption of the lists of indexes with the public key pk_{vs}. We provide a reduction on ct^v, which can also be extended to ct^{ℓ_v} and $ct^{\ell_{id}}$. Let ct^* be the challenge ciphertext given to the adversary $\mathcal{A}'$ with encryption public key pk_T related to votes $\{v, v_\mathcal{A}\}$ in the NM-CPA security game. The adversary $\mathcal{A}'$ simulates G_2 and G_3 for the adversary $\mathcal{A}$. It returns $\beta = (ct^*, ct^{\ell_v}, ct^{\ell_{id}}, \pi)$ as the last ballot for the voter upk_j in the query phase. Since the result R is only derived from $\mathcal{BB}_0$, $\mathcal{A}'$ can obtain the votes related to the adversary-determined ballots using the decryption oracle of the NM-CPA challenger. Then, the adversary $\mathcal{A}'$ computes the tally result where v is a vote of upk_j and simulates the proof as G_2. Indeed, the adversary $\mathcal{A}'$ simulates G_2 if the challenge ciphertext ct^* is an encryption of v, otherwise it simulates G_3. $\mathcal{A}'$ returns the bit $b_{\mathcal{A}'}$ as a result against the NM-CPA security game based on the the output of the adversary $\mathcal{A}$. It follows that the adversarial advantage in distinguishing G_2 from G_3 is at most equal to the adversarial advantage against the NM-CPA security game, which is negligible in security parameter λ.

Game G_4. This game contains a sequence of the games $\{G_4^i\}_{i=1}^n$ that swaps the last ballots of other voters on $\mathcal{BB}_0$, namely, the set of voters $\{upk_i\}_{i=1}^n$ such that $i \in \mathbb{I}$ and $i \neq j$. For instance, G_4^1 is equal to G_3 but the last ballot of the voter upk_1 on the $\mathcal{BB}_0$ is replaced with the last ballot of the same voter upk_1 on $\mathcal{BB}_1$. At the end of G_4^n, we have replaced the view of the adversary $\mathcal{A}$ over the last ballots from $\mathcal{BB}_0$, in the experiment $\mathbf{Exp}^{CR,0}_{ES,\mathcal{A}}$ and game G_1, into $\mathcal{BB}_1$, in the experiment $\mathbf{Exp}^{CR,1}_{ES,\mathcal{A}}$. The advantage of the adversary $\mathcal{A}$ in distinguishing between through the transition over these games is negligible in security parameter λ as we proved earlier. This can be generalized to any ballot on $\mathcal{BB}_0$, including the ballots in the voters' CBR. In this case, the adversary's view of G_1, which is equal to $\mathbf{Exp}^{CR,0}$, can be transferred to the to $\mathbf{Exp}^{CR,1}$ with $\mathcal{BB}_1$.

We have now proved that the advantage of $\mathcal{A}$ through these games is $neg(\lambda)$. Note that if the coerced voter gives the adversary an $\ell \geq 2^n$, where n is the total number of ballots in the coerced CBR, the adversary can trivially determine that ℓ is not valid. However, if the given ℓ is smaller than 2^n, the adversary cannot tell whether the coercion attempt was successful or not. It can only guess all the possible valid indexes from the knowledge of the number of ballots in the CBR but cannot distinguish whether the last ballot on the coerced CBR corresponds to the coerced vote. This is because each ballot cast by a voter (possibly under coercion) is always followed by a number of noise ballots

on the CBR, according to the distributions $D_{\mathcal{R}}$ and $D_{\mathcal{T}}$.

TABLE 2: Performance of Loki.

nr. of candidates	2	4	16
Ballot generation	6.407ms	12.081ms	78.109ms
Ballot obfuscation	6.517ms	12.006ms	78.350ms
Ballot verification	0.529ms	0.596ms	5.618ms
Tally computation (10 voters)	1ms	5ms	16ms
Tally verification (10 voters)	0.7ms	3ms	12ms
Tally computation (10000 voters)	0.3s	0.4s	1s
Tally verification (10000 voters)	0.1s	0.2s	0.8ms
Tally computation (1000000 voters)	30s	44s	1.6min
Tally verification (1000000 voters)	10s	20s	1.3min

8. Performance

We implement a prototype of Loki in Python [43] using the zksk library [44] for the implementation of the disjunctive zero-knowledge proofs. We empirically show that the code implementing the key functions of Loki has a minimal impact on the overall performance of the voting system.

We run our experiments in a 2023 MacBook Pro laptop on an M2 Pro processor with 16GB of RAM. Table 2 shows the results of the average time to run ballot generation, obfuscation, and verification in Loki. It also shows the average time to tally the result as well as its verification time. Ballot generation and obfuscation for 2 candidates require less than 10ms, while ballot verification, tallying, and verification of the election result require less than 1ms. Even considering 64 candidates, both ballot generation and obfuscation can be computed under 1s, while ballot verification takes less than 200ms.

The time required for re-randomizing a ballot is close to that required for ballot generation. Both of them are not affected by the number of voters, which only affects the cost of tallying. Although our prototype implementation is written in Python, the Voting server can efficiently populate a voter CBR with periodic re-randomization of ballots. The performance of Loki with respect to voters and noise ballots only affects the capacity of the Voting server. With a large number of voters, the Voting server can be distributed into several servers, each of them responsible for a subset of voters' CBR. Differently from the other schemes, tallying complexity is linear to the number of the voters. It is not affected by the number of noise ballots because tallying is performed to only the last ballot of each CBR. Better performance can be achieved by using Baby-Step-Giant-Step to speed up the tallying and by implementing Loki in a more efficient language such as C++ or Rust.

9. Conclusion

In this paper we propose the first technique for deniable vote updating that can also evade last-minute voter coercion, which is normally ruled out by state-of-the-art voting schemes that support revoting. Our technique relies on two key insights. The first one is the introduction of a list of voters' votes (the CBR) which includes their obfuscations. This approach is similar to introducing dummy ballots as suggested, for example, in KTV-Helios [45] or VoteAgain [7]. However, our technique does not necessarily need a filtering phase removing dummy ballots as we show in Loki, therefore tallying can be very efficient. The second insight of our technique is the use of lists of indexes, which allows the Voting server to distinguish a coerced from an uncoerced ballot. Our approach takes advantage of the fact that unbeknownst to the coercer, Voting server and voter share the knowledge about which of the votes posted to the CBR on the bulletin board were intended, re-randomized, or coerced. Since by construction, this information is hidden from the coercer, our technique provides the voter with a mechanism to resist a coercer's attempt to force the voter to submit a ballot on his behalf or to abstain from voting. A successful brute-force attack against a voter CBR would require that the coercer casts several incorrect ballots. While this is doable in principle, it is noticeable to the Voting server, which can eventually refuse to add ballots at all and block such an attack. Unless the coercer has given different instructions to the voter, the latter just tells the coercer that they have not voted yet. Even if the coercer instructs the voter to vote at certain times, the coercer's chances of guessing the correct indexes are small because the probability distribution determining the time to add the ballots to the bulletin board is chosen by the Voting server and not by the voter. In conclusion, our technique and its implementation in Loki show that fake-credential and deniable vote updating approaches are not mutually exclusive and can be combined to achieve flexible vote updating.

References

[1] R. Canetti, H. Krawczyk, and J. B. Nielsen, "Relaxing chosen-ciphertext security," in *Advances in Cryptology-CRYPTO 2003: 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003. Proceedings 23*. Springer, 2003, pp. 565–582.

[2] M. R. Clarkson, S. Chong, and A. C. Myers, "Civitas: Toward a secure voting system," in *2008 IEEE Symposium on Security and Privacy (sp 2008)*, 2008, pp. 354–368.

[3] A. Juels, D. Catalano, and M. Jakobsson, *Coercion-Resistant Electronic Elections*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.

[4] R. Araújo, S. Foulle, and J. Traoré, *A Practical and Secure Coercion-Resistant Scheme for Internet Voting*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 330–342. [Online]. Available: https://doi.org/10.1007/978-3-642-12980-3_20

[5] W. Smith, "New cryptographic election protocol with best-known theoretical properties," *Frontiers in Electronic Elections (FEE 2005)*, 10 2005.

[6] D. Achenbach, C. Kempka, B. Löwe, and J. Müller-Quade, "Improved Coercion-Resistant electronic elections through deniable Re-Voting," *USENIX Journal of Election Technology and Systems (JETS)*, Aug. 2015. [Online]. Available: https://www.usenix.org/conference/jets15/workshop-program/presentation/achenbach

[7] W. Lueks, I. Querejeta-Azurmendi, and C. Troncoso, "VoteAgain: A scalable coercion-resistant voting system," in *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 1553–1570. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/lueks

[8] J. Clark and U. Hengartner, "Selections: Internet voting with over-the-shoulder coercion-resistance," in *Financial Cryptography and Data Security*, G. Danezis, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 47–61.

[9] "Cobra: Toward concurrent ballot authorization for internet voting," in *2012 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (EVT/WOTE 12)*. Bellevue, WA: USENIX Association, Aug. 2012. [Online]. Available: https://www.usenix.org/conference/evtwote12/workshop-program/presentation/Essex

[10] M. Kutylowski and F. Zagórski, "Verifiable internet voting solving secure platform problem," in *Advances in Information and Computer Security, Second International Workshop on Security, IWSEC 2007, Nara, Japan, October 29-31, 2007, Proceedings*, ser. Lecture Notes in Computer Science, A. Miyaji, H. Kikuchi, and K. Rannenberg, Eds., vol. 4752. Springer, 2007, pp. 199–213. [Online]. Available: https://doi.org/10.1007/978-3-540-75651-4_14

[11] O. Spycher, R. Haenni, and E. Dubuis, "Coercion-resistant hybrid voting systems," in *Electronic Voting 2010, EVOTE 2010, 4th International Conference, Co-organized by Council of Europe, Gesellschaft für Informatik and E-Voting.CC, July 21st - 24th, 2010, in Castle Hofen, Bregenz, Austria*, ser. LNI, R. Krimmer and R. Grimm, Eds., vol. P-167. GI, 2010, pp. 269–282. [Online]. Available: https://dl.gi.de/20.500.12116/19498

[12] S. Delaune, S. Kremer, and M. Ryan, "Verifying privacy-type properties of electronic voting protocols," *J. Comput. Secur.*, vol. 17, no. 4, p. 435–487, dec 2009.

[13] D. Unruh and J. Müller-Quade, "Universally composable incoercibility," in *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, ser. Lecture Notes in Computer Science, T. Rabin, Ed., vol. 6223. Springer, 2010, pp. 411–428. [Online]. Available: https://doi.org/10.1007/978-3-642-14623-7_22

[14] R. Canetti and R. Gennaro, "Incoercible multiparty computation," in *Proceedings of 37th Conference on Foundations of Computer Science*, 1996, pp. 504–513.

[15] R. Küsters and T. Truderung, "An epistemic approach to coercion-resistance for electronic voting protocols," in *30th IEEE Symposium on Security and Privacy (S&P 2009), 17-20 May 2009, Oakland, California, USA*. IEEE Computer Society, 2009, pp. 251–266. [Online]. Available: https://doi.org/10.1109/SP.2009.13

[16] R. Küsters, T. Truderung, and A. Vogt, "A game-based definition of coercion resistance and its applications," *J. Comput. Secur.*, vol. 20, no. 6, pp. 709–764, 2012. [Online]. Available: https://doi.org/10.3233/JCS-2012-0444

[17] D. Bernhard, O. Kulyk, and M. Volkamer, "Security proofs for participation privacy, receipt-freeness and ballot privacy for the helios voting scheme," in *Proceedings of the 12th International Conference on Availability, Reliability and Security, Reggio Calabria, Italy, August 29 - September 01, 2017*. ACM, 2017, pp. 1:1–1:10. [Online]. Available: https://doi.org/10.1145/3098954.3098990

[18] V. Cortier, D. Galindo, S. Glondu, and M. Izabachène, "Election verifiability for helios under weaker trust assumptions," in *Computer Security - ESORICS 2014 - 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part II*, ser. Lecture Notes in Computer Science, M. Kutylowski and J. Vaidya, Eds., vol. 8713. Springer, 2014, pp. 327–344. [Online]. Available: https://doi.org/10.1007/978-3-319-11212-1_19

[19] X. Boyen, T. Haines, and J. Müller, "Epoque: Practical end-to-end verifiable post-quantum-secure e-voting," in *2021 IEEE European Symposium on Security and Privacy (EuroS P)*, 2021, pp. 272–291.

[20] J. Liedtke, R. Küsters, J. Müller, D. Rausch, and A. Vogt, "Ordinos: a verifiable tally-hiding electronic voting protocol," in *IEEE 5th European Symposium on Security and Privacy (EuroS&P 2020)*, 2020.

[21] R. Küsters, J. Müller, E. Scapin, and T. Truderung, "select: A lightweight verifiable remote voting system," in *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*, 2016, pp. 341–354.

[22] R. Küsters, T. Truderung, and A. Vogt, "Formal analysis of chaumian mix nets with randomized partial checking," in *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*. IEEE Computer Society, 2014, pp. 343–358. [Online]. Available: https://doi.org/10.1109/SP.2014.29

[23] R. Kusters, T. Truderung, and A. Vogt, "Verifiability, privacy, and coercion-resistance: New insights from a case study," in *32nd IEEE Symposium on Security and Privacy, S&P 2011, 22-25 May 2011, Berkeley, California, USA*. IEEE Computer Society, 2011, pp. 538–553. [Online]. Available: https://doi.org/10.1109/SP.2011.21

[24] G. S. Grewal, M. D. Ryan, S. Bursuc, and P. Y. Ryan, "Caveat coercitor: Coercion-evidence in electronic voting," in *2013 IEEE Symposium on Security and Privacy*, 2013, pp. 367–381.

[25] V. Cortier, D. Galindo, R. Küsters, J. Müller, and T. Truderung, "SoK: Verifiability notions for e-voting protocols," in *IEEE Symposium on Security and Privacy*, 2016, pp. 779–798.

[26] J. Benaloh and D. Tuinstra, "Receipt-free secret-ballot elections (extended abstract)," in *STOC*. ACM, 1994, pp. 544–553.

[27] M. Hirt and K. Sako, "Efficient receipt-free voting based on homomorphic encryption," in *Proceedings of the 19th International Conference on Theory and Application of Cryptographic Techniques*, ser. EUROCRYPT'00. Berlin, Heidelberg: Springer-Verlag, 2000, p. 539–556.

[28] J. Cohen and M. Fischer, "A robust and verifiable cryptographically secure election scheme (extended abstract)," in *FOCS*. IEEE, 1985, pp. 372–382.

[29] J. Benaloh, "Verifiable secret-ballot elections," Ph.D. dissertation, Yale University, December 1996.

[30] V. Cortier and J. Lallemand, "Voting: You can't have privacy without individual verifiability," in *CCS*. ACM, 2018, pp. 53–66.

[31] P. Chaidos, V. Cortier, G. Fuchsbauer, and D. Galindo, "Beleniosrf: A non-interactive receipt-free electronic voting scheme," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1614–1625. [Online]. Available: https://doi.org/10.1145/2976749.2978337

[32] G. Gallegos-García, V. Iovino, A. Rial, P. B. Rønne, and P. Y. A. Ryan, "(universal) unconditional verifiability in e-voting without trusted parties," *CoRR*, vol. abs/1610.06343, 2016. [Online]. Available: http://arxiv.org/abs/1610.06343

[33] L. Hirschi, L. Schmid, and D. A. Basin, "Fixing the achilles heel of e-voting: The bulletin board," in *34th IEEE Computer Security Foundations Symposium, CSF 2021, Dubrovnik, Croatia, June 21-25, 2021*. IEEE, 2021, pp. 1–17. [Online]. Available: https://doi.org/10.1109/CSF51468.2021.00016

[34] R. Küsters, T. Truderung, and A. Vogt, "A game-based definition of coercion-resistance and its applications," in *CSF*. IEEE, 2010, pp. 122–136.

[35] N. Soroush, "A new technique for deniable vote updating," *PhD Colloquium - E-Vote-ID*, 2021.

[36] T. Haines, J. Mueller, and I. Querejeta-Azurmendi, "Scalable coercion-resistant e-voting under weaker trust assumptions," in *Proceedings of ACM SAC Conference (SAC'23)*, 2023.

[37] Mastercard, "Mastercard biometric card," https://www.mastercard.us/en-us/business/overview/safety-and-security/authentication-services/biometrics/biometrics-card.html, 2023.

[38] Samsung, "Samsung's biometric card," https://news.samsung.com/global/samsungs-biometric-card-ic-all-in-one-fingerprint-solution-for-a-new-payment-experience, 2023.

[39] Thales, "Biometrics in payment: The case of the biometric bank card," https://www.thalesgroup.com/en/markets/digital-identity-and-security/banking-payment/cards/biometrics-in-banking, 2021.

[40] R. Cramer, R. Gennaro, and B. Schoenmakers, "A secure and optimally efficient multi-authority election scheme," *European transactions on Telecommunications*, vol. 8, no. 5, pp. 481–490, 1997.

[41] R. Cramer, I. Damgård, and B. Schoenmakers, "Proofs of partial knowledge and simplified design of witness hiding protocols," in *Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings*, ser. Lecture Notes in Computer Science, Y. Desmedt, Ed., vol. 839. Springer, 1994, pp. 174–187. [Online]. Available: https://doi.org/10.1007/3-540-48658-5_19

[42] D. Bernhard, O. Pereira, and B. Warinschi, "How not to prove yourself: Pitfalls of the fiat-shamir heuristic and applications to helios," in *Advances in Cryptology–ASIACRYPT 2012: 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings 18*. Springer, 2012, pp. 626–643.

[43] R. Giustolisi, M. Sheikhi, and C. Schuermann, "Loki prototype implementation," https://github.com/fgiustol/Loki, 2023.

[44] W. Lueks, B. Kulynych, J. Fasquelle, S. Le Bail-Collet, and C. Troncoso, "Zksk: A library for composable zero-knowledge proofs," in *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society*, ser. WPES'19. New York, NY, USA: Association for Computing Machinery, 2019, p. 50–54. [Online]. Available: https://doi.org/10.1145/3338498.3358653

[45] O. Kulyk, V. Teague, and M. Volkamer, "Extending helios towards private eligibility verifiability," in *E-Voting and Identity - 5th International Conference, VoteID 2015, Bern, Switzerland, September 2-4, 2015, Proceedings*, ser. Lecture Notes in Computer Science, R. Haenni, R. E. Koenig, and D. Wikström, Eds., vol. 9269. Springer, 2015, pp. 57–73. [Online]. Available: https://doi.org/10.1007/978-3-319-22270-7_4

[46] D. Bernhard, V. Cortier, D. Galindo, O. Pereira, and B. Warinschi, "Sok: A comprehensive analysis of game-based ballot privacy definitions," in *2015 IEEE Symposium on Security and Privacy*, 2015, pp. 499–516.

Appendix A.
Strong-consistency and strong-correctness

We now show that Loki also meets strong-consistency and strong-correctness as defined by Bernhard et al. [46]. Strong-consistency ensures that the voting result R corresponds to the output of the counting function that is directly applied on valid ballots. It also captures potential information leaking in the output of the tally algorithm. Strong-correctness ensures that no adversary can generate a bulletin board $\mathcal{BB}$ such that $\mathsf{Validate}(\mathcal{BB}, \beta) = \perp$ for a ballot β which is generated honestly. The strong-consistency experiment and the strong-correctness experiment are defined in Algorithm 4 and in Algorithm 5 respectively.

$\mathbf{Exp}^{s-cons}_{ES,\mathcal{A}}(\lambda, \mathbb{I}, \mathbb{V})$:
$((pk_T, sk_T), (pk_{vs}, sk_{vs})) \leftarrow \mathsf{Setup}(\lambda, \mathbb{I}, \mathbb{V})$
$\{(L_{id}, (usk_{id}, upk_{id}))\} \leftarrow \mathsf{Register}(id)\}_{id \in \mathbb{I}}$
$\mathcal{BB} = [L_1, \ldots, L_n] \leftarrow \mathcal{A}(pk, \{usk_i\}_{i \in \mathbb{I}})$
if $\exists \beta_i$ such that $\mathsf{ValidInd}(\beta_i) = \perp$ **then return** $\perp$
$(R, \Pi) \leftarrow \mathsf{Tally}(\mathcal{BB}, sk_T)$
if $R = \perp$ **then return** $\perp$
if $R \neq \rho(\mathsf{Extract}(\beta_1, sk_T), \ldots, \mathsf{Extract}(\beta_n, sk_T))$ **then**
  return $\top$
else
  return $\perp$
end if

Algorithm 4: The strong-consistency experiment $\mathbf{Exp}^{s-cons}$, in which the adversary $\mathcal{A}$ outputs the $\mathcal{BB}$ such that (R, Π) is not consistent with the output of ρ w.r.t. the Extract function on the last ballots $\{\beta_i\}_{i \in \mathbb{I}}$ of the $\mathcal{BB}$.

$\mathbf{Exp}^{s-corr}_{ES,\mathcal{A}}(\lambda, \mathbb{I}, \mathbb{V})$:
$((pk_T, sk_T), (pk_{vs}, sk_{vs})) \leftarrow \mathsf{Setup}(\lambda, \mathbb{I}, \mathbb{V})$
$\{(L_{id}, (usk_{id}, upk_{id}))\} \leftarrow \mathsf{Register}(id)\}_{id \in \mathbb{I}}$
$(\mathcal{BB}, usk, v, id) \leftarrow \mathcal{A}(pk, \{usk_i\}_{i \in \mathbb{I}})$
$\beta \leftarrow \mathsf{Vote}(id, usk, pk_T, pk_{vs}, v, \ell)$
if $\mathsf{Validate}(\mathcal{BB}, \beta) = \perp$ **then return** $\top$
return $\perp$

Algorithm 5: The strong-correctness experiment $\mathbf{Exp}^{s-corr}$, in which the adversary $\mathcal{A}$ outputs the $\mathcal{BB}$ such that the honestly generated ballots are not valid with respect to the $\mathcal{BB}$.

Definition 4. *Strong-Consistency. Let* ES = (Setup, RegisterVoter, Vote, Validate, Append, VerifyVote, Obfuscate, Tally, VerifyTally) *be an election scheme for an electoral roll* $\mathbb{I}$*, candidate list* $\mathbb{V}$*, security parameter* λ*, and the result function* $\rho : \mathbb{I} \times \mathbb{V} \rightarrow R$*.* ES *provides strong-consistency if there exist the functions* Extract *and* ValidInd *that satisfy the following conditions:*

1) *For* $((pk_T, sk_T), (pk_{vs}, sk_{vs})) \leftarrow \mathsf{Setup}(\lambda, \mathbb{I}, \mathbb{V})$*, for all* $\{usk_i\}_{i \in \mathbb{I}}$ *output by* $(L_{id}, (usk_{id}, upk_{id})) \leftarrow \mathsf{Register}(id)$*, and for any (last) ballot* $\beta \leftarrow \mathsf{Vote}(id, usk, pk_T, pk_{vs}, v, \ell)$ *with* $v \in \mathbb{V}$*, we have* $\mathsf{Extract}(\beta, sk_T) = (id, v)$

2) *For any bulletin board and ballot generated by any PPT adversary $\mathcal{A}$, such that $(\mathcal{BB}, \beta) \leftarrow \mathcal{A}$ and* $\mathsf{Validate}(\mathcal{BB}, \beta) = \top$*, then* $\mathsf{ValidInd}(\beta) = \top$.
3) *The advantage of any PPT $\mathcal{A}$ such that $Pr[\mathbf{Exp}^{s-cons}_{ES,\mathcal{A}}(\lambda, \mathbb{I}, \mathbb{V}) = 1]$ is negligible in the security parameter λ.*

Definition 5. *Strong-Correctness. Let ES = (*Setup*,* RegisterVoter*,* Vote*,* Validate*,* Append*,* VerifyVote*,* Random*,* Tally*,* VerifyTally*) be an election scheme for an electoral roll $\mathbb{I}$, candidate list $\mathbb{V}$, and security parameter λ. The scheme ES has the strong-correctness property if the advantage of any PPT adversary $\mathcal{A}$ such that:*

$$Pr[\mathbf{Exp}^{s-corr}_{ES,\mathcal{A}}(\lambda, \mathbb{I}, \mathbb{V}) = 1]$$

is negligible in the security parameter λ.

Theorem 4. *Loki satisfies strong-consistency and strong-correctness.*

Proof: We first define the functions $\mathsf{Extract}(\beta, sk_T) = (id, v)$ and $\mathsf{ValidInd}(\beta)$ w.r.t. Loki as follows:

1) $\mathsf{Extract}(\beta, sk_T)$ takes the ballot $\beta = (id, upk, \overline{ct}_v)$ and the extraction key sk_T, and verifies all the proofs in $\overline{ct}_v$. It returns $\perp$ if any of the checks fail; Otherwise $\mathsf{Dec}(ct_v, sk_T) = v$ and returns (id, v).
2) $\mathsf{ValidInd}(\beta)$ verifies that i) $id \in \mathbb{I}$, ii) the proof of the ballot β, It returns $\top$ otherwise returns $\perp$.

It can be immediately seen that Loki meets the first two properties of strong-consistency due to the correctness of the encryption, the decryption scheme, and the zero-knowledge proofs. To prove the third property, we show that the adversary $\mathcal{A}$ cannot construct a bulletin board $\mathcal{BB}$ in such a way that the tally algorithm contradicts with the output of the result function ρ. For each voter $id \in \mathbb{I}$, the ideal result function selects the last ballot and counts the result on $(\mathsf{Extract}(\beta_1, sk_T), \ldots, \mathsf{Extract}(\beta_n, sk_T))$. Therefore, both Tally and ρ receive the same input, namely, a list of ballots, as the revote policy is based on the last ballot for each $id \in \mathbb{I}$. The homomorphic property of the ElGamal encryption and the proof of decryption ensures that the result obtained by extracting the result from the multiplied ciphertexts is equivalent to counting the votes (plaintext) of $(\mathsf{Extract}(\beta_1, sk_T), \ldots, \mathsf{Extract}(\beta_n, sk_T))$. As a result, the validity of all ballots in the $\mathcal{BB}$ and the homomorphic property of the underlying ElGamal encryption scheme guarantees that $R = \rho(\mathsf{Extract}(\beta_1, sk_T), \ldots, \mathsf{Extract}(\beta_n, sk_T))$ in Loki with overwhelming probability.

To prove strong-correctness, we observe that in Loki, an honestly generated ballot is not appended to the bulletin board if the same ballot exists. An honest ballot is the output of either the Obfuscate function or the Vote function. The probability that a ballot generated by Obfuscate or Vote is equal to a ballot already included in the bulletin board is negligible, as both Obfuscate and Vote use probabilistic encryption schemes. So, the adversary $\mathcal{A}$ has a negligible advantage in $\mathbf{Exp}^{s-corr}$.