VoteAgain: A scalable coercion-resistant voting system

Wouter Lueks, *EPFL;* Iñigo Querejeta-Azurmendi,
Universidad Carlos III Madrid/ITEFI, CSIC; Carmela Troncoso, *EPFL*

This paper is included in the Proceedings of the 29th USENIX Security Symposium.

August 12–14, 2020

978-1-939133-17-5

VOTEAGAIN: A scalable coercion-resistant voting system

Wouter Lueks
EPFL, SPRING Lab

Iñigo Querejeta-Azurmendi*
Universidad Carlos III Madrid
ITEFI, CSIC

Carmela Troncoso
EPFL, SPRING Lab

Abstract

The strongest threat model for voting systems considers *coercion resistance*: protection against coercers that force voters to modify their votes, or to abstain. Existing remote voting systems either do not provide this property; require expensive operations for tallying; or burden users with the need to store cryptographic key material and with the responsibility to deceive their coercers. We propose VOTEAGAIN, a scalable voting scheme that relies on the revoting paradigm to provide coercion resistance. VOTEAGAIN uses a novel deterministic ballot padding mechanism to ensure that coercers cannot see whether a vote has been replaced. This mechanism ensures tallying takes *quasilinear time*, making VOTEAGAIN the first revoting scheme that can handle elections with millions of voters. We prove that VOTEAGAIN provides ballot privacy, coercion resistance, and verifiability; and we demonstrate its scalability using a prototype implementation of its core cryptographic primitives.

1 Introduction

Remote electronic voting in which voters cast their ballot outside a poll-booth environment, from their own devices, is susceptible to large-scale vote buying and coercion [29]. Yet, many deployed electronic voting systems [2, 24, 36] do not support coercion resistance. This might be suitable in Western democracies where freedom and privacy are well rooted in society. However, under authoritarian regimes [27] or in younger democracies [32], coercion is a serious problem.

There are two kind of coercion-resistant electronic voting systems in the literature. The first kind provides users with fake voting credentials that voters use/produce when coerced, enabling deletion of coerced votes [11, 29]. This approach has several downsides: (i) voters need to store their true voting credential on their devices, (ii) the system cannot give feedback on whether the correct credential was used, and thus voters cannot be sure if their vote has been recorded correctly at the

time of voting, and (iii) voters need to convincingly lie while being coerced which may be a challenge. The second kind relies on the *revoting paradigm* [1, 23, 30, 33]. These schemes avoid the drawbacks associated with the fake-credential approach by allowing voters to submit fully to coercers, and later on supersede coerced votes by casting a new ballot. This approach requires that the coercer cannot detect whether a voter has cast new ballots. To achieve this, state-of-the-art schemes [1, 30] *require a quadratic number of operations*, concretely a pair-wise comparison of all ballots, to privately filter superseded ballots. As an example, for the Iowa Democratic caucus with only 176,574 voters, Achenbach et al.'s solution [1] would require 1.1 core years to filter the ballots.

We propose VOTEAGAIN, a scalable coercion-resistant (re)voting scheme. VOTEAGAIN's efficiency relies on two key insights: First, one can hide the number of ballots per user by *inserting a deterministic number of dummy ballots which depends solely on the number of voters and the number of cast ballots*. Thus, it reveals nothing about the number of ballots cast by individual voters, hiding any (re)voting patterns induced by voters or coercers. Second, because of the deterministic nature of the approach one can *execute filtering in the clear*, reducing the filtering time *from quadratic to quasilinear*: $O(n \log n)$ where n is the number of ballots. As a result, for the Iowa caucus *our construction requires under 14 core minutes*. We estimate that VOTEAGAIN using 224 cores (less than $50 on Amazon, or $75K on dedicated hardware) can filter hundreds of millions of ballots in hours.

We make the following contributions:

✓ We introduce VOTEAGAIN, a novel remote electronic revoting scheme based on well defined and widely used cryptographic constructions.

✓ We introduce a novel efficient deterministic padding scheme that hides revoting at a low cost. The complexity of the resulting filtering phase is $O(n \log n)$ where n is the number of ballots. Our experiments show that in many practical scenarios the cost can be even lower.

✓ We show that previous definitions of coercion resistance in

---

Table 1: Comparison of different voting schemes.

	Revoting					Security Properties		
	Deniable	Verif. Filter	Complexity	Crypto state	Authentication	Ballot Privacy	Verifiability	Coercion Res.
JCJ [8, 11, 29]	No[1]	Yes	n^2	Yes	k-out-of-t	k-out-of-t	k-out-of-t	k-out-of-t + AC
Black-box [21]	TTP	No	n	Yes	Unclear	k-out-of-t	TTP	TTP
Revote [1, 30]	k-out-of-t	Yes	n^2	Yes	TTP	k-out-of-t	TTP	k-out-of-t + AC
Helios [2]	*revoting is not possible*			No	TTP	k-out-of-t	TTP	N/A
VOTEAGAIN	TTP	Yes	$n \log n$	No	TTP	k-out-of-t	TTP	TTP

the revoting setting are vacuous. We provide a new coercion-resistance definition and we adapt modern definitions of ballot privacy [6] and verifiability [12, 13] to the revoting setting. We prove that VOTEAGAIN satisfies these definitions.

✓ We evaluate the scalability of VOTEAGAIN on a prototype implementation of the core cryptographic primitives. Our results show that VOTEAGAIN can support elections with millions of users.

2 Related Work

Coercion-resistant voting schemes fall under two categories: either they enable voters to generate *fake authentication credentials* or they allow the voter to *revote*. Coercion-resistant schemes using fake credentials, introduced by Juels et al. [29] (JCJ), are used in several voting schemes [3, 8, 10, 11]. In these schemes, the voter has both real and fake authentication credentials (or pre-registered passwords and panic passwords [10, 16]). When coerced, the voter lies to the coercer, using a fake authentication credential (or handling it to the coercer), resulting in a non-counted ballot. Ballots cast with the real credential are counted. These schemes provide the real authentication credential to the voter during registration phase (in which the coercer *must* be absent). The voter must securely store these authentication credentials for later use, i.e., voters need to maintain cryptographic state.

Coercion resistant schemes based on revoting allow voters to cast multiple ballots and then filter these ballots, typically counting the last ballot per voter. For such a scheme to be coercion resistant, the filtering stage must be *deniable* [1], i.e., it must not expose which ballots are filtered, as this would expose revoting actions. *Black box* filtering where a trusted third party (TTP) performs the filtering privately is deniable [21], but not verifiable. To the best of our knowledge, there exist two publicly-verifiable deniable re-voting schemes [1, 30]. To obtain public verifiability, these schemes use a distributed authority to compare each pair of ballots before shuffling to

privately mark superseded ballots, requiring $O(n^2)$ operations. After shuffling, these marks are decrypted and the tallying server verifiably filters superseded ballots. As literally specified in these papers, these schemes are *'not efficient for large scale elections'*. We confirm in Section 7 that Achenbach et al.'s scheme [1] cannot efficiently handle small elections of a hundred thousand users.

Both the JCJ based and the private revoting based schemes offer a solution with a k-out-of-t assumption for coercion resistance. However, on top of that, these schemes require the existence of anonymous communications channels (AC) to avoid coercion attacks such as forced abstention.

For authentication, most schemes require users to store cryptographic state [1, 8, 11, 21, 29, 30, 35], or remember special passwords [10, 16]. Helios [2] and Apollo [20] rely on regular username/password. To improve verifiability by distributing the trust of the entity deciding which users are eligible voters, some schemes require that voters authenticate to k out of t parties [3, 8, 11, 29]. However, this results in a complex registration phase for the user Revoting based schemes (including VOTEAGAIN) can be extended to this setting to reduce the trust assumptions required for authentication correctness (and hence verifiability). Table 1 summarizes the comparison between VOTEAGAIN and previous work.

3 System and threat model

Actors. There are five actors in VOTEAGAIN: voters, a polling authority, a bulletin board, a tally server, and the trustees.

Voters. Voters interact with the polling authority and the public bulletin board to cast their ballots. Each voter has the means to authenticate herself to the polling authority (e.g., an electronic identity card). There are n voters.

Polling Authority (PA). The PA authenticates users and provides them with ephemeral voting tokens. Voters use these tokens to sign their ballots before posting them to the public bulletin board.

Public Bulletin Board (PBB). The PBB is an append-only list of cast ballots. Ballots are posted during the election phase

---

[1]Revoting is possible, but revotes are not deniable. JCJ instead achieves coercion resistance using fake authentication credentials.

Table 2: Comparison of assumptions in pre-election phase and election phase required to mitigate coercion attacks in fake credentials and revoting-based systems.

Assumptions	Fake Credentials	Revoting
Pre-election phase		
No coercion	✓	N/A
Inalienable authentication	✓	N/A
Election phase		
Lie convincingly	✓	✗
Coercer absent at some point during election	✓	✓
Absence of coercer after coercion	✗	✓
Device holding voting secrets or need to remember special pwds	✓	✗
Inalienable authentication	✗	✓

Table 3: Trust assumptions on VOTEAGAIN entities to achieve each property.

	Ballot Privacy	Verifiability	Coercion resistance
PA	Untrusted	Trusted	Trusted
TS	Untrusted	Untrusted	Trusted
PBB	Untrusted	Untrusted	Trusted
Trustees	k-out-of-t	Untrusted	Untrusted

where this material is stored during the voting phase; and that users can lie convincingly. These assumptions are not needed in VOTEAGAIN. Our construction allows users to vote from any device, preventing coercion attacks that rely on destroying or stealing the voting device.

In VOTEAGAIN, voters authenticate against the PA every time they vote to obtain an ephemeral voting token. The PA *must* be honest with respect to verifiability and coercion resistance. To enable quasilinear filtering we also require that the TS is honest with respect to coercion resistance. This assumption is stronger than Achenbach et al.'s k-out-of-t assumption on the trustees [1], but their relaxation comes at a quadratic computational cost, see Table 1.

Finally, we require VOTEAGAIN to satisfy the following informal properties. We formalize these properties in Section 6. Table 3 summarizes the trust in each party required for achieving them.

Definition 1 (Ballot privacy [6])**.** Assuming at most $k - 1$ malicious trustees, no coalition of malicious parties (including the PA and TS) can learn the vote of an honest user.

Definition 2 (Coercion resistance)**.** Assuming that the PA, the TS, and the PBB are honest, no coercer can use the information made public on PBB to determine if coercion was successful or not, provided that the election outcome does not leak this information.

If voters use an anonymous communication system to post their ballots to the PBB, then the trust assumption on the PBB for coercion is not necessary.

Definition 3 (Verifiability)**.** Assuming that the PA is honest, VOTEAGAIN guarantees that: (i) the last ballot per voter will be tallied, (ii) adversary $\mathcal{A}$ cannot include more malicious votes in the tally than the number of voters it controls, and (iii) honest ballots cannot be replaced. If voters do not verify that their ballots are correctly appended to the PBB, ballots can be dropped or replaced by earlier ballots if those exist.

4 VOTEAGAIN: High-level overview

We sketch the key ideas of VOTEAGAIN. For simplicity, in this section we omit the zero-knowledge proofs that parties

by the voters. During the tally phase, the tally server and trustees post their proofs and results to the bulletin board. Ad-hoc implementations [25] or blockchain-based implementations [18, 19] would be suitable for VOTEAGAIN's PBB.

Tally Server (TS). The TS filters the ballots. It adds dummy ballots, shuffles the ballots, groups them by voter, and selects the last ballot for each voter.

Trustees. The trustees mix and decrypt the selected ballots to reveal the outcome of the election. Each trustee has a partial decryption key for a k-out-of-t encryption system.

Threat model. We assume an adversary $\mathcal{A}$ whose goal it is to coerce voters into casting votes for a particular candidate or to abstain. This adversary, although computationally bounded, may coerce any voter – but not *all* voters. Under coercion, the coerced voter does exactly as instructed (without needing to lie). The coercer learns all information stored and received by the voter at the time of coercion. We assume that after coercion, and before the end of the election, the coercer does not control a voter, such that the voter can cast at least one more vote. We also assume that the user's means of authentication is *inalienable* [1], that is, a coercer can neither eliminate nor duplicate a voter's means of authentication.

While these assumptions are strong, we point out that so are the assumptions behind coercion resistant solutions that rely on fake credentials [8, 11, 29] (see Table 2). Fake-credential based solutions assume that users cannot be coerced during registration and hence need inalienable means of authentication during this phase; that users can store and hide cryptographic key material and hence are required to have access to
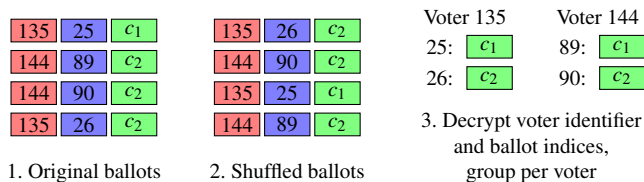
Figure 1: Basic filtering process by tally server without using dummies. Ballots consist of an encrypted voter identifier (■), an encrypted ballot index (■), and an encrypted vote (■).
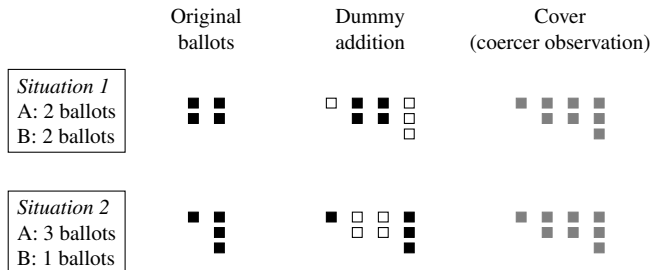

Figure 2: The original ballots' groups (■) create distinguishable situations. Adding 2 dummy voters casting a total of 4 dummy ballots (□), the situations become indistinguishable.

use to show that they performed operations correctly. We describe the protocols in detail in Section 5.1.

VOTEAGAIN proceeds in three phases: the pre-election phase, the election phase, and the tally phase. During the pre-election phase, the polling authority (PA) assigns to each voter i a random voter identifier vid_i, and a random initial ballot index m_i. These values are known only to the PA.

Casting ballots. During the election phase, voters can cast as many votes as they want. To cast a vote, voter i first authenticates to the PA using her inalienable authentication means to obtain an ephemeral voting token. This voting token includes: an encrypted voter identifier γ, containing vid_i, and an encrypted ballot index I, containing m_i. After each authentication, the PA increases m_i by one. Next, the voter encrypts her choice of candidate as v. Finally, the voter sends the encrypted vote v, the encrypted voter identifier γ, the encrypted ballot number I, and a signature using the ephemeral token to the bulletin board.

Filtering ballots. The encrypted voter identifiers and ballot indices enable the tally server (TS) to efficiently select the last ballot for each voter. The TS uses the simplest mechanism possible: It shuffles the ballots, and then decrypts the voter identifiers and ballot indices. The ballots can then publicly be grouped per voter, and the last ballot can be identified by inspection. Finally, the trustees tally the last ballot of each voter. See Figure 1.

Hiding patterns using dummies. By itself, shuffling and filtering is not a coercion-resistant mechanism. A coercer can still perform the 1009 attack [37] in which the coercer forces a voter to cast a specific number of ballots and looks for a group of that size in the filtering step. If such group does not exist, the coerced voter has revoted. In VOTEAGAIN, the TS inserts a deterministic number of *dummy ballots* and *dummy voters* before shuffling the ballots to hide such patterns while maintaining the simple public filtering procedure.

We illustrate VOTEAGAIN's dummy mechanism in Figure 2, in a scenario with two voters (*A* and *B*) where, the coercer forces voter *A* to cast 2 ballots. At the end of the election phase the coercer observes 4 ballots and must determine whether *A* revoted (situation 2) or not (situation 1). Without dummies, distinguishing these situations is trivial: if *A* revoted there is a group of 3 ballots and one of 1 ballot, and

there are two groups of 2 ballots otherwise. We add dummy ballots and voters to make both situations look identical. The idea is to find a *cover* of ballots that could result from both situations. For instance, adding to either situation two dummy voters that cast four dummy ballots total yields groups of 1, 2, 2, and 3 ballots. This observation makes both situations indistinguishable for the coercer (Figure 2, right).

To ensure that the cover is *independent* from the voters' real actions, its appearance must depend *only* on the information available to the coercer: (1) the number of ballots n_B posted by users to the bulletin board; and (2) the number of voters v that cast a ballot. The goal of the dummy generation strategy is to allocate dummy ballots such that the adversary observes the *same cover* regardless of the actual distributions of the n_B ballots over v voters.

Consider the case of two voters, i.e., $v = 2$, and 9 ballots, i.e., $n_B = 9$. As the filtering stage only reveals the sizes of the groupings and not their relation to voters the adversary's possible observations are $(1,8), (2,7), (3,6)$, and $(4,5)$. To cover all these scenarios one needs 8 voters (6 of which are dummy) casting $1, 2, 3, 4, 5, 6, 7$, and 8 ballots, for a total of $36 - 9 = 27$ dummy ballots.

We add dummy ballots to real voters as well to reduce the number of group sizes that are possible. For example, in the previous scenario one can pad the cases $(1,8), (2,7), (3,6), (4,5)$ to $(1,8), (2,8), (4,8), (4,8)$. This can be covered with a cover containing voters with $1, 2, 4, 8$ ballots each. Building this cover requires only 2 dummy voters and $15 - 9 = 6$ dummy ballots. We stress that *the number of added dummy ballots is independent of how the real ballots are actually distributed among the two voters*.

We refer to Section 5.2 for a generic, efficient algorithm for computing a cover.

Filtering with dummies. Before shuffling the ballots, the TS adds dummy ballots to achieve the desired grouping. To ensure that the TS cannot modify the election outcome, the TS assigns different tags to real and dummy ballots.

To determine how to add dummies, the TS inspects the decrypted voter identifiers and ballot indices; determines a cover; and then computes how many dummies to add to exist-

																							Voter 74				Voter 135	

1. Original ballots   2. Tagged ballots + dummies   3. Shuffled ballots   4. Decrypted voter identifiers and ballot indices, grouped per voter
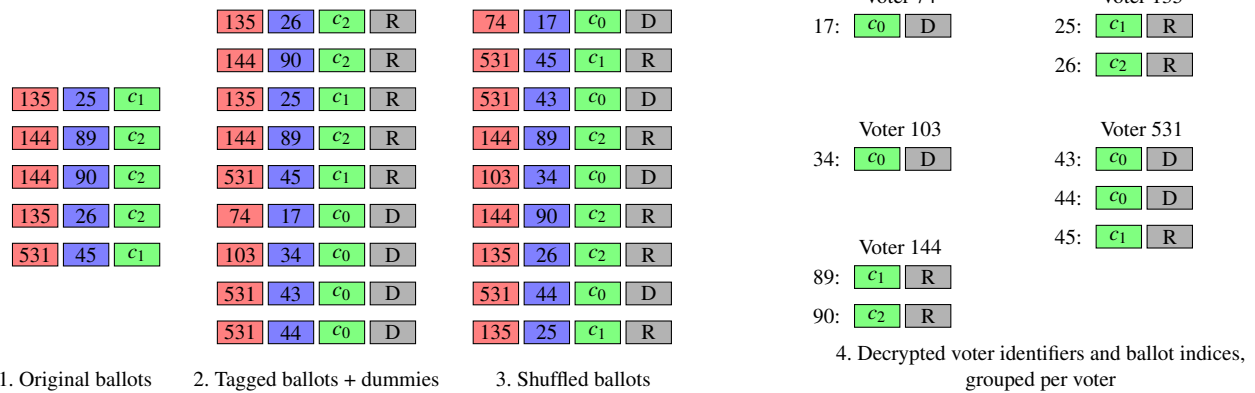
Figure 3: Filtering process by tally server including dummies. Labels as in Figure 1. To enable correctness proofs, the TS tags real ballots and dummy ballots with an encrypted marker (■).

ing voters, and how many dummies to add to dummy voters. Consider the example in Figure 3. Given 3 voters and 5 ballots, a cover with groups of size 1,1,2,2, and 3 suffices. The TS therefore adds 4 dummy ballots in step 2: 2 dummies to existing voter 531, and two dummy voters, 74 and 103, each with one dummy vote.

After adding the dummy ballots, the TS shuffles all ballots. Next, the TS decrypts the voter identifiers and ballot indices; groups ballots per voter, and selects the last ballot per voter. The tags enable the TS to prove that it did not omit real ballots cast by real voters, and it did not count dummy votes cast by dummy voters. In particular, the TS proves in zero-knowledge that the selected votes are either tagged as a real vote and therefore must correspond to the last ballot of a real voter; or the selected vote corresponds to a dummy voter (i.e., all the ballots in the group are tagged as dummies). Finally, the TS privately discards the selected votes corresponding to dummy voters. We refer the reader to Section 5.1 for the full details.

Design choices. Obtaining coercion resistance requires strong assumptions on some of the parties. In this section, we discuss our design choices and motivate our trust assumptions (see Table 2 for a comparison with other protocols). First, we believe that revoting is an easy to understand solution to achieve coercion resistance. It requires no extra devices, no memorization, no interaction with several entities during registration, and no lying. For instance, Estonians have used a revoting model for years for their elections [26] with 44% of the electorate having used internet voting [15]. Second, it does not require voters to securely store cryptographic material, allowing a vote cast from any device. This further reduces the possibility of coercion attacks by confiscating the credential storage device.

All coercion resistance schemes *require* absence of the coercer at some point during the process. Fake-credential solutions assume that the coercer is absent during registration and at some point during the voting phase. Revoting, instead,

assumes that a voter will have time after the coercion to cast the last vote. In the case of a remote registration process, a targeted attack will most likely succeed in both scenarios. However, attacks scale much better in the fake-credential setting: coercers have the entire registration period (e.g., 24 days in Spain) to coerce a voter. In contrast, coercers in the revoting setting must monitor *all* coerced voters after coercion to prevent them from revoting before the election closes.

We decide to trade-off trust with respect to coercion resistance on the PA and TS to obtain high gains in usability and efficiency: trust on the PA relieves users from keeping cryptographic state; and trust on the TS enables VOTEAGAIN's quasilinear filtering of ballots.

5 The VOTEAGAIN voting scheme

Preliminaries. Let ℓ be a security parameter. Let $\mathbb{G}$ be a cyclic group of prime order p generated by generator g. We write $\mathbb{Z}_p$ for the integers modulo p. We write $a \in_R A$ to denote that a is chosen uniformly at random from the set A.

VOTEAGAIN uses ElGamal's encryption scheme given by: A key generation algorithm $\text{EC.KeyGen}(\mathbb{G}, g, p)$ which outputs a public-private key-pair $(\text{pk} = g^{\text{sk}}, \text{sk})$ for $\text{sk} \in_R \mathbb{Z}_p$; an encryption function $\text{EC.Enc}(\text{pk}, m)$ which takes as input a public key pk and a message $m \in \mathbb{G}$ and returns a ciphertext $c = (c_1, c_2) = (g^r, m \cdot \text{pk}^r)$ for $r \in_R \mathbb{Z}_p$; and an decryption algorithm $\text{EC.Dec}(\text{sk}, c)$ which returns the message $m = c_2 \cdot c_1^{-\text{sk}}$. VOTEAGAIN uses deterministic encryption (with randomness zero) as a cheap verifiable 'encoding' for the ballot tags. Because the encryption is deterministic, verifiers can cheaply check that the encrypted tags have been correctly formed.

We use a traditional signature scheme given by: A key generation algorithm $\text{Sig.Keygen}(1^\ell)$ that generates a public-private key-pair $(\text{pk}_\sigma, \text{sk}_\sigma)$; a signing algorithm $\sigma = \text{Sig.Sign}(\text{sk}_\sigma, m)$ that signs messages $m \in \{0, 1\}^*$; and a veri-

fication algorithm $\mathsf{Sig.Verify}(\mathsf{pk}_\sigma, \sigma, m)$ that outputs $\top$ if σ is a valid signature on m and $\bot$ otherwise.

We use verifiable shuffles [4] to support coercion resistance in a private way. These enable an entity to verifiably shuffle a list of homomorphic ciphertexts in such a way that it is infeasible for a computationally bounded adversary to match input and output ciphertexts.

VOTEAGAIN uses *mixnets*, a standard approach [7, 28, 34] to compute the election result given the filtered ballots output by the TS. The trustees jointly run the $\mathsf{Vote.DKeyGen}(1^\ell, k, t, n_C)$ protocol where ℓ is the security parameter ℓ, n_C the number of candidates, t the number of trustees, and k is the number of trustees needed to decrypt ciphertexts. This protocol outputs a public encryption key pk_T and each trustee i obtains a private decryption key $\mathsf{sk}_{T,i}$. To encrypt her vote for candidate c, a voter calls $(\mathsf{v}, \pi) = \mathsf{Vote.Enc}(\mathsf{pk}_T, c)$ to obtain an encrypted vote v and proof π that v encrypts a choice for a valid candidate. We denote the encryption of the zero candidate (i.e., no candidate) with explicit randomizer $r \in_R \mathbb{Z}_p$ by $\mathsf{Vote.ZEnc}(\mathsf{pk}_T; r)$. The algorithm $\mathsf{Vote.Verify}(\mathsf{pk}_T, \mathsf{v}, \pi)$ outputs $\top$ if the encrypted vote v is correct, and $\bot$ otherwise. Given a list of selected votes $\{V_1, \ldots, V_\kappa\}$, the trustees jointly run the $(r, \Pi) \leftarrow \mathsf{Vote.MixDecryptTally}(\mathsf{pk}_T, \{V_1, \ldots, V_\kappa\})$ protocol to compute the election result r and a proof of correctness Π. Internally, $\mathsf{Vote.MixDecryptTally}$ uses a standard verifiable mix network and verifiable decryption to shuffle and decrypt the ballots, and then computes the final result in the clear. Any verifier can run $\mathsf{Vote.VerifyTally}(\mathsf{pk}_T, \{V_1, \ldots, V_\kappa\}, r, \Pi)$ to verify whether the result r is computed correctly.

The TS uses standard zero-knowledge proofs of knowledge [22] to prove that it operated correctly. We use the Fiat-Shamir heuristic [17] to convert them into non-interactive proofs of knowledge. We adopt the Camenisch-Stadler notation [9] to denote such proofs and write, for example,

$$\mathsf{SPK}\{(\mathsf{sk}) : \mathsf{pk} = g^{\mathsf{sk}} \wedge m = \mathsf{EC.Dec}(\mathsf{sk}, I)\}$$

to denote the non-interactive signature proof of knowledge that the prover knows the private key sk corresponding to pk and that I decrypts to m under sk.

5.1 VOTEAGAIN description

VOTEAGAIN proceeds in three phases: the pre-election phase, the election phase, and the tally phase. See Table 4 for a summary of frequently used symbols.

5.1.1 Pre-election phase

In the pre-election phase, the PBB publishes the candidates, and the TS and the trustees prepare their cryptographic material. The PA assigns a unique, random voter identifier vid_i to each eligible voter. The correspondence between voters and their identifiers is private to the PA. The PA also generates a

Table 4: Summary of notation.

Symbol	Description
$(\mathbb{G}, g, p)$	Group, generator and prime order
$\mathbb{Z}_p$	Integers modulo the group order p
N	Number of eligible voters
t, k	Number of trustees and decryption threshold
$\mathsf{pk}_{PA}, \mathsf{pk}_{TS}, \mathsf{pk}_T$	Public keys of PA, TS, and trustees
$\mathsf{sk}_{PA}, \mathsf{sk}_{TS}, \mathsf{sk}_{T,i}$	Private keys of PA, TS, and trustee i
vid_i, m_i	Voter identifier and ballot index of voter i
γ, I	The encrypted vid and ballot index
pk, sk	Ephemeral signing keys
τ, σ^τ	Ephemeral voting token and signature by PA
β, σ	Ballot and signature using ephemeral key pk
v, π	Encrypted vote and zero knowledge proof of correct encryption
n_B, n_D	Number of real and dummy ballots on the board
$\theta, \theta_R, \theta_D$	Ballot tags for unknown, real, and dummy ballots
$\overline{vid}_i, \overline{m}_i$	Decrypted voter identifier and ballot index
π_i^{dec}, π_i^{sel}	Zero knowledge proof of correct decryption and vote selection
$\overline{V}_i$	Selected vote for group i
r	Election result
Φ, Π	Full filter and tally proofs

random token index m_i for each of the voters to enable the selection of the last ballot per voter. More formally:

Procedure 1 (Setup). To setup an election system with security parameter ℓ, electoral roll $\mathcal{E}$, candidate list $\mathcal{C}$, threshold k, and t trustees, the different entities run the $\mathsf{Setup}(1^\ell, \mathcal{E}, \mathcal{C}, k, t)$ procedure. First, they pick a group $\mathbb{G}$ with generator g and prime order p. They then proceed with the following steps:

1. The PBB initializes the bulletin board, and adds the list of candidates $\mathcal{C}$ to the bulletin board.
2. The PA stores the electoral roll $\mathcal{E}$. Let N be the number of eligible voters on the electoral roll. The PA generates a random and unique voter identifier $vid_i \in \mathbb{G}$ and ballot index $m_i \in \{2^{\ell-2}, \ldots, 2^{\ell-1} - 1\}$ for each voter $\mathcal{V}_i$ on the electoral roll and stores them internally. Finally, the PA generates a public-private key-pair $(\mathsf{pk}_{PA}, \mathsf{sk}_{PA}) = \mathsf{Sig.Keygen}(1^\ell)$ to sign tokens. It publishes pk_{PA}.
3. The TS generates a public-private ElGamal key-pair $(\mathsf{pk}_{TS}, \mathsf{sk}_{TS}) = \mathsf{EC.KeyGen}(\mathbb{G}, g, p)$. It publishes pk_{TS}.
4. The trustees run $\mathsf{Vote.DKeyGen}(1^\ell, k, t, |\mathcal{C}|)$ to generate a public encryption key pk_T and decryption keys $\mathsf{sk}_{T,i}$ that the trustees keep private.

5.1.2 Election phase

In the election phase (see Figure 4), voters first authenticate to the PA to obtain an ephemeral voting token τ. They use this token to sign their ballot β, and post the ballot on the bulletin
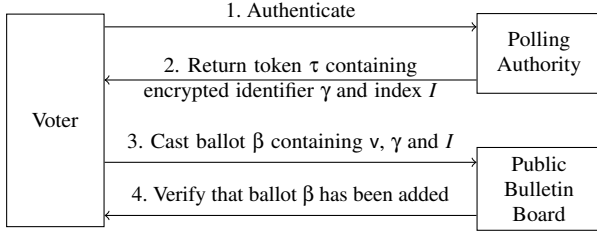
Figure 4: Election phase: Overview.

board. The bulletin board verifies that the ballot is valid. We formalize this phase in three procedures:

Procedure 2 (GetToken(id, Auth)). On input her identity id and her inalienable means of authentication Auth:

1. The voter authenticates to the PA using Auth.
2. The PA looks up the corresponding voter identifier vid_i and ballot index m_i. Then, the PA encrypts the voter identifier $\gamma = \mathsf{EC.Enc}(\mathsf{pk_{TS}}, vid_i)$ and ballot number $I = \mathsf{EC.Enc}(\mathsf{pk_{TS}}, m_i)$ (it first encodes m_i as an element of $\mathbb{G}$), and increments the ballot index $m_i := m_i + 1$. The PA hides the index m_i from the user to prevent coercers – who can see what users can see under coercion – from being able to detect whether the user revoted.
3. The PA creates an ephemeral signing key $(\mathsf{pk}, \mathsf{sk}) = \mathsf{Sig.Keygen}()$ and signs this key together with the encrypted voter identifier and ballot number:

$$\sigma^\tau = \mathsf{Sig.Sign}(\mathsf{sk_{PA}}, \mathsf{pk} \parallel \gamma \parallel I)$$

and returns the token $\tau = (\mathsf{pk}, \mathsf{sk}, \gamma, I, \sigma^\tau)$ to the user.
4. The user verifies the token $\tau = (\mathsf{pk}, \mathsf{sk}, \gamma, I, \sigma^\tau)$ by checking that $\mathsf{Sig.Verify}(\mathsf{pk_{PA}}, \sigma^\tau, \mathsf{pk} \parallel \gamma \parallel I) = \top$.

Procedure 3 (Vote(τ, c)). To cast a vote, the voter takes as private input the ephemeral voting token $\tau = (\mathsf{pk}, \mathsf{sk}, \gamma, I, \sigma^\tau)$ and a candidate $c \in C$, and then proceeds as follows:

1. Encrypts her candidate c as $(\mathsf{v}, \pi) = \mathsf{Vote.Enc}(\mathsf{pk_T}, c)$ to obtain ciphertext v and zero-knowledge proof of correct encryption π.
2. Creates the ballot

$$\beta = (\mathsf{v}, \pi, \mathsf{pk}, \gamma, I, \sigma^\tau, \sigma)$$

where $\sigma = \mathsf{Sig.Sign}(\mathsf{sk}, \mathsf{v} \parallel \pi \parallel \mathsf{pk} \parallel \gamma \parallel I \parallel \sigma^\tau)$. The voter posts the ballot β to the public bulletin board.
3. The public bulletin board runs $\mathsf{Valid}(\beta)$, see below, to check that the ballot is valid, before appending it.
4. Finally, the voter verifies that the ballot β has been appended to the bulletin board.

Procedure 4 (Valid(β)). The bulletin board verifies that the ballot $\beta = (\mathsf{v}, \pi, \mathsf{pk}, \gamma, I, \sigma^\tau, \sigma)$ is valid with respect to the current state of the bulletin board as follows:
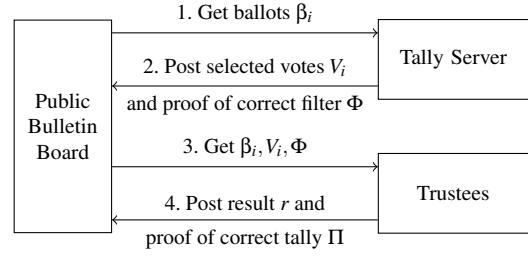

Figure 5: Tally phase: Overview

1. The PBB checks the correctness of the encrypted vote; of the user's signature using the ephemeral key pk; and the PA's signature on this ephemeral key pk, the encrypted voter identifier γ, and the encrypted ballot number I:

$$\mathsf{Vote.Verify}(\mathsf{pk_T}, \mathsf{v}, \pi) = \top$$
$$\mathsf{Sig.Verify}(\mathsf{pk}, \sigma, \mathsf{v} \parallel \pi \parallel \mathsf{pk} \parallel \gamma \parallel I \parallel \sigma^\tau) = \top$$
$$\mathsf{Sig.Verify}(\mathsf{pk_{PA}}, \sigma^\tau, \mathsf{pk} \parallel \gamma \parallel I) = \top.$$

2. The PBB checks that neither the encrypted vote v nor the key pk appear in any ballot β' on the bulletin board.

If any of these checks fails, the bulletin board returns $\bot$, otherwise the PBB returns $\top$.

5.1.3 Tally phase

In the tally phase (see Figure 5), the TS takes the ballots from the PBB, adds dummy ballots, and shuffles them. Then, it selects the last vote per voter (see Figure 6). To prevent dummy voters from causing overhead in the trustees' shuffle and decrypt phase, the TS shuffles the selected ballots and removes all ballots cast by dummy voters. Finally, the trustees shuffle and decrypt the selected ballots from real voters. Formally, we define two procedures, one to filter votes (Filter), and one to tally the selected ballots (Tally):

Procedure 5 (Filter). After the election closes, the TS selects the selected votes V_i and produces the filter proof Φ. If it aborts, it publishes the current Φ to the public bulletin board.

1. The tally server (TS) retrieves an ordered list of ballots $[\beta_1, \ldots, \beta_{n_B}]$ from the PBB, where $\beta_i = (\mathsf{v}_i, \pi_i, \mathsf{pk}_i, \gamma_i, I_i, \sigma_i^\tau, \sigma_i)$. The TS verifies the ballots by running step 1 of Valid and verifies that there are are no duplicate votes v_i or ephemeral public keys pk_i on the bulletin board. If any of these checks fails, the TS sets $\Phi = \bot$, posts it to the bulletin board, and aborts.
2. The TS removes the proofs and signatures to obtain stripped ballots. It provably tags the ballots as 'real' ballots using a deterministic ElGamal encryption (with randomness zero) of the value $g^0 = 1_{\mathbb{G}}$, $\theta_R = \mathsf{EC.Enc}(\mathsf{pk_{TS}}, g^0) = (g^0, g^0 \mathsf{pk}^0) = (1_{\mathbb{G}}, 1_{\mathbb{G}})$:

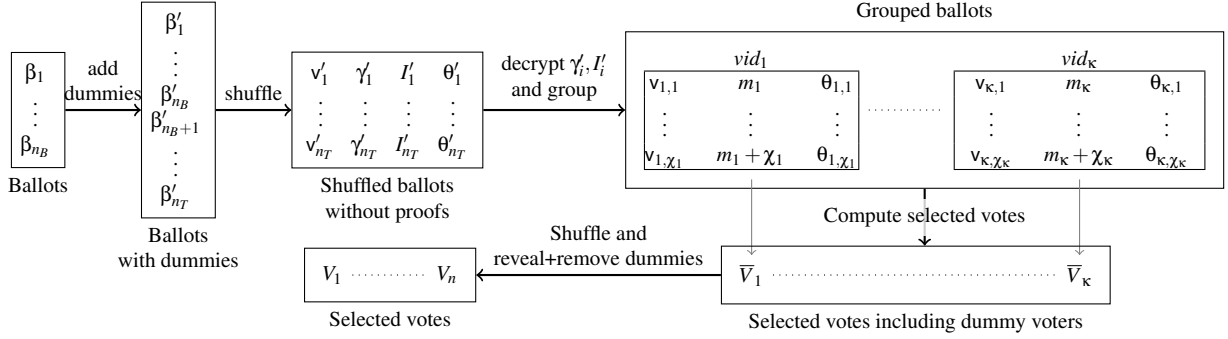$$\beta_i' = (\mathsf{v}_i, \gamma_i, I_i, \theta_R).$$

Figure 6: High-level overview of ballot filtering and grouping. Let n_B be the number of ballots, n_D be the number of dummies, $n_T = n_B + n_D$ be their sum, κ be the number of voters plus number of dummy voters, and χ_i be the number of (dummy) ballots for (dummy) voter i. First, the TS adds dummy ballots and proves they are well-formed. Then shuffles all ballots without the proofs, hiding which ballots were dummies. Then it verifiably decrypts both the encrypted voter identifiers γ_i' and the encrypted indices I_i' to group the ballots by vid and to select the last votes $\overline{V}_i$. Finally, it outputs the selected votes V_i without dummies.

Next, the TS creates n_D dummy ballots and provably tags them as such using a deterministic ElGamal encryption of the value g, $\theta_D = \mathsf{EC.Enc}(\mathsf{pk_{TS}}, g) = (1_\mathbb{G}, g \cdot \mathsf{pk}^0)$:

$$\beta_i' = (\mathsf{v}_\varepsilon, \gamma_i, I_i, \theta_D),$$

where $i > n_B$ and $\mathsf{v}_\varepsilon = \mathsf{Vote.ZEnc}(\mathsf{pk_T}; 0)$. We explain below how the TS determines the number of dummies n_D as well as the values for γ_i and I_i. The TS adds the stripped ballots $B' = [\beta_1', \ldots, \beta_{n_B + n_D}']$ to Φ.

3. The TS shuffles the stripped ballots $B' = [\beta_1', \ldots, \beta_{n_B + n_D}']$ and randomizes the ciphertexts, to obtain a list of shuffled and randomized stripped ballots $B'' = [\beta_1'', \ldots, \beta_{n_B + n_D}'']$, which it adds, together with a proof π_σ that this shuffle was performed correctly, to Φ.

4. The TS now operates on each shuffled ballot $\beta_i'' = (\mathsf{v}_i', \gamma_i', I_i', \theta_i')$. It decrypts γ_i' to recover the shuffled and decrypted identifier $\overline{vid}_i$. It also decrypts I_i' to obtain the shuffled ballot index $\overline{m}_i$ and proves it did so correctly:

$$\pi_i^{\mathsf{dec}} = \mathsf{SPK}\{(\mathsf{sk_{TS}}) : \mathsf{pk_{TS}} = g^{\mathsf{sk_{TS}}} \wedge$$
$$\overline{vid}_i = \mathsf{EC.Dec}(\mathsf{sk_{TS}}, \gamma_i') \wedge$$
$$\overline{m}_i = \mathsf{EC.Dec}(\mathsf{sk_{TS}}, I_i')\}$$

It then adds $C = [(\overline{vid}_1, \overline{m}_1, \pi_1^{\mathsf{dec}}), \ldots, (\overline{vid}_{n_B + n_D}, \overline{m}_{n_B + n_D}, \pi_{n_B + n_D}^{\mathsf{dec}})]$ to Φ. The TS aborts and adds $\bot$ to Φ if the decrypted ballot indices $\overline{m}_i$ are not unique for a given voter identifier. More precisely, it aborts if there exists indices $i, j; i \neq j$ such that $(\overline{vid}_i, \overline{m}_i) = (\overline{vid}_j, \overline{m}_j)$.

5. The TS groups the ballots with the same voter identifier, and selects the ballot with the highest ballot index from each group. Let $G_1, \ldots, G_\kappa$ be the sets of ballot indices grouped by voter identifier. Consider group G_j of size χ_j. Let $j* = \arg\max_{k, k \in G_j} \overline{m}_k$ be the index for which the ballot index $\overline{m}_{j*}$ is maximal. Group G_j either corresponds to a real, or to a fake voter. The TS produces a reencryption $\overline{V}_j$ of the encrypted votes as follows:

(a) If the group G_j corresponds to a real voter, then the TS simply reencrypts the vote corresponding to the last ballot, i.e., it picks r_j at random and sets

$$\overline{V}_j = \mathsf{v}_{j*} \cdot \mathsf{Vote.ZEnc}(\mathsf{pk_T}; r_j),$$

to a randomized encryption of v_{j*}.

(b) If the group G_j corresponds to a fake voter, then picks r_j at random and sets $\overline{V}_j$ to an empty vote:

$$\overline{V}_j = \mathsf{Vote.ZEnc}(\mathsf{pk_T}; r_j).$$

The TS proves that it computed the $\overline{V}_j$ correctly. If the corresponding voter is real, then the ballot β_{j*}'' selected in (a) should be a real ballot, so $\mathsf{EC.Dec}(\mathsf{sk_{TS}}, \theta_{j*}')$ should equal g^0. If the voter is fake, then for all tags θ_{i_k}' with $i_k \in G_j$, we have that $\mathsf{EC.Dec}(\mathsf{sk_{TS}}, \theta_{i_k}') = g^1$. Let $G_j = \{i_1, \ldots, i_{\chi_j}\}$ and $\theta = \prod_{k=1}^{\chi_j} \theta_{i_k}'$, then the TS constructs the proof

$$\pi_j^{\mathsf{sel}} = \mathsf{SPK}\{(r_j, \mathsf{sk_{TS}}) : \mathsf{pk_{TS}} = g^{\mathsf{sk_{TS}}} \wedge$$
$$((g^0 = \mathsf{EC.Dec}(\mathsf{sk_{TS}}, \theta_{j*}') \wedge \overline{V}_j = \mathsf{v}_{j*} \cdot \mathsf{Vote.ZEnc}(\mathsf{pk_T}; r_j)) \vee$$
$$(g^{\chi_j} = \mathsf{EC.Dec}(\mathsf{sk_{TS}}, \theta) \wedge \overline{V}_j = \mathsf{Vote.ZEnc}(\mathsf{pk_T}; r_j)))\}.$$

The TS adds the list of filtered encrypted votes $F = [(vid_1, \overline{V}_1, \pi_1^{\mathsf{sel}}), \ldots, (vid_\kappa, \overline{V}_\kappa, \pi_\kappa^{\mathsf{sel}})]$ to Φ.

6. The list $\mathcal{S}_D = [\overline{V}_1, \ldots, \overline{V}_\kappa]$ of selected votes contains ballots by dummy voters. In the next two steps, the TS removes these. First, the TS shuffles and randomizes the ciphertexts to obtain a new list $\mathcal{S}_D' = [\overline{V}_1', \ldots, \overline{V}_\kappa']$, which it adds, together with a proof π_σ' of correct shuffle, to Φ.

7. The TS knows the indices $\mathcal{D}$ of votes in $\mathcal{S}_D'$ that correspond to dummy voters and randomizers r_i such that $\overline{V}_i' = \mathsf{Vote.ZEnc}(\mathsf{pk_T}; r_i)$ for $i \in \mathcal{D}$. The TS adds $\mathcal{D}$ and $R = [r_i]_{i \in \mathcal{D}}$ to Φ.

8. Finally, the TS publishes the remaining votes $\mathcal{S} = [V_1, \ldots, V_n]$ and the full proof Φ to the public bulletin board.

The filter procedure ensures that the TS cannot replace ballots by real voters: a selected vote must either correspond to a ballot by a real voter (condition a) or the selected vote is empty and the voter is a dummy voter (condition b). Moreover, the TS can only remove votes cast by dummy voters.

Procedure 6 (Tally). To compute the final tally, the trustees proceed as follows:

1. The trustees verify that the TS operated honestly by running the VerifyFilter() algorithm (see below). If VerifyFilter returns $\perp$ they return $(r, \Pi) = (\perp, \perp)$.
2. Let $\mathcal{S} = [V_1, \ldots, V_n]$. The trustees jointly run the $(r, \Pi) \leftarrow$ Vote.MixDecryptTally$(\mathsf{pk}_\mathsf{T}, \mathcal{S})$. They publish the election result r and the zero knowledge proof of correctness Π to the public bulletin board.

5.1.4 Verification

Any external auditor can use the PBB to verify that all steps in the tally and filtering phases were performed correctly. We define the following verification procedures:

Procedure 7 (VerifyFilter). Any party can verify that the filtering processes was performed correctly by running VerifyFilter(). This algorithm examines the content of the bulletin board and performs the following checks:

1. First, check if all ballots are correct and that no duplicate votes or public keys are included in the ballots as per step 1 of Filter. If the checks fail, the bulletin board should contain $\Phi = \perp$; VerifyFilter returns $\perp$ if that is not the case. Otherwise, it continues.
2. Retrieve the selected votes $\mathcal{S}$ and the proof Φ from the bulletin board and continue as follows:
   (a) Verify that stripped real ballots are correctly formed. Consider ballots $[\beta_1, \ldots, \beta_{n_B}]$, where $\beta_i = (v_i, \pi_i, \mathsf{pk}_i, \gamma_i, I_i, \sigma_i^\tau, \sigma_i)$ and check that the stripped ballot $\beta'_i = (v_i, \gamma_i, I_i, \theta_R)$ has been added to Φ (where θ_R is as above).
   (b) Verify that the dummy ballots on the bulletin board are correctly formed. For ballots $\beta'_{n_B+1}, \ldots, \beta'_{n_B+n_D}$ where $\beta'_i = (v_i, \gamma_i, I_i, \theta_i)$, check that $v_i = v_\varepsilon$ and $\theta_i = \theta_D$ (where v_ε and θ_D are as above).
   (c) Let $B' = [\beta'_1, \ldots, \beta'_{n_B+n_D}]$ be all stripped ballots, and $B'' = [\beta''_1, \ldots, \beta''_{n_B+n_D}]$ the shuffled and randomized ballots. Verify the shuffle proof π_σ to check that B'' is a correct shuffle of B'.
   (d) Next, let $C = [(\overline{vid}_1, \overline{m}_1, \pi_1^\mathsf{dec}), \ldots, (\overline{vid}_{n_B+n_D}, \overline{m}_{n_B+n_D}, \pi_{n_B+n_D}^\mathsf{dec})]$ from the bulletin board, and verify the decryption proofs π_i^dec for each of the shuffled ballots β''_i.
   (e) Let vid'_i and m'_i be the plaintexts verified in the previous step. Group the ballots by voter identifier into ballot groups G_j. For each group G_j, find

ballot β_{j*} with the highest ballot index, recompute $\tau = \prod_{k=1}^{\chi_j} \tau_{i_k}$, and verify the reencryption proof π_j^sel.
   (f) Let $\mathcal{S}_D$ be the selected votes $[\overline{V}_1, \ldots, \overline{V}_\kappa]$ and $\mathcal{S}'_D = [\overline{V}'_1, \ldots, \overline{V}'_\kappa]$ the shuffled and randomized votes. Verify the shuffle proof π'_σ for $\mathcal{S}_D$ and $\mathcal{S}'_D$.
   (g) Finally, for each $i \in \mathcal{D}$ verify that $\overline{V}'_i = $ Vote.ZEnc$(\mathsf{pk}_\mathsf{T}; r_i)$ and that $\mathcal{S} = [\mathcal{S}_D[i] \mid i \notin \mathcal{D}]$.

If any of the checks fail, it returns $\perp$, and $\top$ otherwise.

Procedure 8 (Verify). Any party can verify the result r and proof Π against the public bulletin board. To do so, they proceed as follows:

1. Verify that the TS operated honestly by running the VerifyFilter() algorithm. If VerifyFilter returns $\perp$, then return $\top$ if $(r, \Pi) = (\perp, \perp)$, otherwise return $\perp$.
2. Given the selected votes $\mathcal{S}$, return the result of Vote.VerifyTally$(\mathsf{pk}_\mathsf{T}, \mathcal{S}, r, \Pi)$.

5.2 Hiding revoting patterns with dummies

In this section we provide a formal description of the dummy generation algorithm introduced in Section 4.

Finding a cover. Formally, a cover is a set $\mathfrak{C} = \{(s_i, z_i)\}_i$ formed by groupings $(s_i, z_i) \in \mathbb{Z}^+ \times \mathbb{Z}^+$. Here, s_i is the size of the ballot groups within that grouping, and z_i is the upper bound on the number of times that such a ballot group can occur in any distribution of the n_B real ballots among real voters. We aim to find a cover of minimal size $|\mathfrak{C}| = \sum_i s_i \cdot z_i$ to minimize the number of dummies added.

A sufficient cover. We derive an upper bound on the amount of dummies required to build a cover. We do not use the number of real voters for this bound. Let n_B be the number of real ballots on the PBB. For simplicity, assume padded group sizes are powers of two, i.e., $s_i = 2^i$ for $i \geq 0$. Given n_B ballots, any distribution can have at most $z_0 = n_B$ groups of size $s_0 = 1$ (one ballot per voter). Similarly, any distribution can have at most $z_1 = \lfloor n_B/2 \rfloor$ groups of size $s_1 = 2$. Recall we pad ballot groups to the next bigger size, so a ballot group of 3 would be padded to one of size $s_2 = 4$ ballots, therefore $z_2 = \lfloor n_B/3 \rfloor$. More generally, there can be at most $z_i = \lfloor n_B/(2^{i-1} + 1) \rfloor$ groups of $s_i = 2^i$ ballots. The biggest possible group (if all ballots were cast by the same voter), has size $2^{\lceil \log_2 n_B \rceil}$. Therefore, the size of the cover $|\mathfrak{C}|$ is bounded by:

$$|\mathfrak{C}| = \sum_{i=0}^{\lceil \log_2 n_B \rceil} z_i \cdot s_i = n_B + \sum_{i=1}^{\lceil \log_2 n_B \rceil} 2^i \left\lfloor \frac{n_B}{2^{i-1} + 1} \right\rfloor$$

$$\leq n_B + \sum_{i=1}^{\lceil \log_2 n_B \rceil} \frac{2^i}{2^{i-1} + 1} n_B \leq n_B + \sum_{i=1}^{\lceil \log_2 n_B \rceil} 2 n_B$$

$$= (1 + 2 \lceil \log_2 n_B \rceil) n_B.$$

An efficient cover. Knowing the number of real voters ν enables to obtain a tighter cover. Consider the example of Sec-

tion 4 with $\nu = 2$ and $n_B = 9$. If we only consider $n_B = 9$, one of the possible distributions of votes would be having $s_1 = \lfloor 9/2 \rfloor = 4$ groups of size 2. However, knowing $\nu = 2$ rules out this possibility. There can be at most one group of size two: if there were 2 groups, each of the 2 voters could only cast 2 ballots, i.e., 4 ballots in total. However, we know there are 9 ballots so at least one voter has voted more than twice, implying that $s_1 = 1$.

When the number of ballots grows this reasoning becomes intractable. Consider ballot groups with group sizes, $s = k^i$ for $i \in [0, \ldots, \lceil \log_k n_B \rceil]$ for a real number $k > 1$. We assume that $n_B > \nu$, otherwise the cover would be trivial: $\mathfrak{C} = \{(s_0 = 1, z_0 = \nu)\}$. We compute the cover as follows.

1. Consider groups of size $s_0 = k^0 = 1$. As $n_B > \nu$, at least one voter must cast more than one ballot, resulting in $(s_0, z_0) = (1, \nu - 1)$.
2. Consider groups of size $s_i = k^i$. We know that given n_B, there can be at most $\alpha_i = \lfloor n_B/(k^{i-1} + 1) \rfloor$ groups of size k^i. The number of groups is also bound by the number of voters. If $\nu \cdot s_i \geq n_B$ then all ballots can be assigned to the ν voters given groups of maximum size s_i, and we set $\nu_i = \nu$, otherwise set $\nu_i = \nu - 1$ so that one voter is not in this grouping. Finally, we need at least $z_i(k^{i-1} + 1)$ ballots to make z_i groups, but we must have enough ballots left over to make ν groups in total, i.e., $n_B \geq z_i(k^{i-1} + 1) + (\nu - z_i)$. Rewriting gives bound $\beta_i = \lfloor (n_B - \nu)/k^{i-1} \rfloor$. We set $z_i = \min(\alpha_i, \nu_i, \beta_i)$.

Assuming $n_B > \nu$, the cover has $|\mathfrak{C}| = \sum_{i=0}^{\lceil \log_k n_B \rceil} z_i s_i > n_B$ ballots, necessitating dummy ballots, and $\sum_{i=1}^{\lceil \log_k n_B \rceil} z_i > \nu$ groups, necessitating dummy voters.

Creating dummy voters and allocating dummy ballots. The TS recovers all voter identifiers *vid* by decrypting the γ_is, and the corresponding ballot indices by decrypting the I_is.

So far, we assumed that ballot index sequences are continuous. However, there can be gaps if some tokens were not used (e.g., the coercer does not use some tokens to identify index gaps in the filtering phase). The TS first requests the number of obtained tokens n'_B from the PA, and adds exactly $n'_B - n_B$ dummy ballots to fill up any gaps, such that n'_B equals the number of obtained tokens. The TS can create a dummy ballot for voter *vid* by setting $\gamma = \mathsf{EC.Enc}(\mathsf{pk_{TS}}, vid)$.

Given the current number of ballots n'_B and the number of real voters ν the TS computes a cover $\mathfrak{C} = \{(s_i, z_i)\}_i$. To this end the TS performs a search to find the best k, i.e., the one that gives the smaller cover. In our experiments in Section 7, k tends to be in the 2 to 4 range, and the search takes less than a second. The TS performs the following steps:

1. For every voter vid_j, $j \in \{1, \ldots, \nu\}$ with t ballots, let $(s_i, z_i) \in \mathfrak{C}$ be the cover group with the smallest size s_i such that $s_i \geq t$. To ensure that dummy ballots are never counted, the TS adds $t - s_i$ dummy votes to vid_j with

descending (and unused) ballot counters *smaller* than the last cast vote by this voter.
2. For each grouping $(s_i, z_i) \in \mathfrak{C}$ let z'_i be the number of real voters that were assigned to this group. The TS adds $z_i - z'_i$ dummy voters. For each dummy voter, it picks a random *vid* and initial ballot index m and creates s_i dummy ballots with increasing ballot indices.

The algorithms Filter and VerifyFilter are quasilinear in the number of real ballots n_B. The TS first adds n_D dummies, so that the bulletin board contains a total of $n_T = n_B + n_D = O(n_B \log n_B)$ ballots (see the bound above). All other steps in Filter and Verify filter are linear in n_T. The claim follows.

6 Security Analysis

We analyze VOTEAGAIN's ballot privacy, verifiability, and coercion resistance. We follow Bernhard et al. [6] and model the trustees as a single trusted party with keys $(\mathsf{pk_T}, \mathsf{sk_T})$, but we note that the result holds when trustees are distributed. We explicitly model the bulletin board PBB as an append only string BB. To ease modeling, we use the following redefinition of our voting scheme $\mathcal{V} = (\mathsf{Setup}, \mathsf{GetToken}, \mathsf{Vote}, \mathsf{Valid}, \mathsf{Filter}, \mathsf{VerifyFilter}, \mathsf{Tally}, \mathsf{Verify})$ where the algorithms output changes to the bulletin board rather than posting to it directly. While Bernhard et al. model voter registration implicitly, we make the registration step explicit using the GetToken function because it forms an integral part of our voting scheme and may happen more than once. The redefined algorithms in $\mathcal{V}$ are as follows:

- $\mathsf{Setup}(1^\ell, \mathcal{E}, \mathcal{C})$ as in Setup in procedure 1 but explicitly returns the public key $\mathsf{pk} = (\mathsf{pk_{PA}}, \mathsf{pk_{TS}}, \mathsf{pk_T})$ and the corresponding private keys $\mathsf{sk_{PA}}, \mathsf{sk_{TS}}, \mathsf{sk_T}$.
- $\mathsf{GetToken}(i)$ returns a token τ as in GetToken() in procedure 2.
- $\mathsf{Vote}(\tau, c)$ returns β as in Vote(τ, c) in procedure 3 but does not post the ballot to the bulletin board. Moreover, the voter first verifies the token τ as in step 4 of procedure 2, and returns $\bot$ if it does not validate.
- $\mathsf{Valid}(\mathsf{BB}, \beta)$ returns the result of Valid(β) in procedure 4 with respect to the bulletin board BB.
- $\mathsf{Filter}(\mathsf{BB}, n'_B, \mathsf{sk_{TS}})$ as in Filter in procedure 5, but takes the number of registrations n'_B as explicit input, and returns $\mathcal{S} \parallel \Phi$ instead of adding them to the board.
- $\mathsf{VerifyFilter}(\mathsf{BB}, \mathcal{S}, \Phi)$ runs VerifyFilter from procedure 7 on $\mathsf{BB}' = \mathsf{BB} \parallel \mathcal{S} \parallel \Phi$ and returns the result.
- $\mathsf{Tally}(\mathsf{BB}, \mathsf{sk_T})$ returns (r, Π) as in Tally in procedure 6.
- $\mathsf{Verify}(\mathsf{BB}, r, \Pi)$ is as in Verify in procedure 8 operating on the bulletin board $\mathsf{BB} \parallel r \parallel \Pi$.

6.1 Ballot privacy

We base our ballot privacy definition on the game-based definition by Bernhard et al. [6]. They model ballot privacy using

$\mathsf{Exp}_{\mathcal{A},\mathcal{V}}^{\mathsf{bpriv},b}(\ell,\mathcal{E},\mathcal{C})$:
    $(\mathsf{pk},\mathsf{sk}_{\mathsf{PA}},\mathsf{sk}_{\mathsf{TS}},\mathsf{sk}_{\mathsf{T}}) \leftarrow \mathsf{Setup}(1^\ell,\mathcal{E},\mathcal{C})$
    $b \leftarrow \mathcal{A}^O(\mathsf{pk},\mathsf{sk}_{\mathsf{PA}},\mathsf{sk}_{\mathsf{TS}})$
    Output b'

$O\mathsf{voteLR}(\tau,c_0,c_1)$:
    Let $\beta_0 = \mathsf{Vote}(\tau,c_0)$ and $\beta_1 = \mathsf{Vote}(\tau,c_1)$
    If $\mathsf{Valid}(\mathsf{BB}_b,\beta_b) = \bot$ return $\bot$
    Else $\mathsf{BB}_0 \leftarrow \mathsf{BB}_0 \parallel \beta_0$ and $\mathsf{BB}_1 \leftarrow \mathsf{BB}_1 \parallel \beta_1$

$O\mathsf{cast}(\beta)$:
    If $\mathsf{Valid}(\mathsf{BB}_b,\beta) = \bot$ return $\bot$
    Else $\mathsf{BB}_0 \leftarrow \mathsf{BB}_0 \parallel \beta$ and $\mathsf{BB}_1 \leftarrow \mathsf{BB}_1 \parallel \beta$

$O\mathsf{board}()$:
    return BB_b

$O\mathsf{tally}(\mathcal{S},\Phi)$
    If $\mathsf{VerifyFilter}(\mathsf{BB}_b,\mathcal{S},\Phi) = \bot$ return $\bot$
    $\mathsf{BB}_b \leftarrow \mathsf{BB}_b \parallel \mathcal{S} \parallel \Phi$
    $\mathsf{BB}_{1-b} \leftarrow \mathsf{BB}_{1-b} \parallel \mathsf{Filter}(\mathsf{BB}_{1-b},|\mathsf{BB}_{1-b}|,\mathsf{sk}_{\mathsf{TS}})$
    $(r,\Pi_0) \leftarrow \mathsf{Tally}(\mathsf{BB}_0,\mathsf{sk}_{\mathsf{T}})$
    $\Pi_1 = \mathsf{SimTally}(\mathsf{BB}_1,r)$
    return (r,Π_b)

Figure 7: In the ballot privacy experiment $\mathsf{Exp}_{\mathcal{A},\mathcal{V}}^{\mathsf{bpriv},b}$, the adversary $\mathcal{A}$ has access to the oracles $O = \{O\mathsf{voteLR}, O\mathsf{cast}, O\mathsf{board}, O\mathsf{tally}\}$. The adversary controls the TS and the PA. It can call $O\mathsf{tally}$ only once.

an indistinguishability game which simultaneously tracks two bulletin boards, BB_0 for the "real" world and BB_1 for the "fake" world. Only one is accessible to the adversary (see Figure 7). The adversary, controlling the polling authority (PA) and the tally server (TS), needs to determine whether the tally was evaluated over the "real" or "fake" world. It can decide how voters vote. Formally, the adversary can make calls to the oracle $O\mathsf{voteLR}(\tau,c_0,c_1)$ to let a user with token τ cast a vote for candidate c_0 on BB_0 and a vote for c_1 on BB_1; and to the oracle $O\mathsf{cast}(\beta)$ to cast ballots β (constructed by the adversary) on BB_0 and BB_1. Because the adversary controls the PA, it can create as many voting tokens as it needs.

The outcome of the election is always computed on the real bulletin board BB_0. The adversary can once ask to compute the outcome by calling the oracle $O\mathsf{tally}(\mathcal{S},\Phi)$ where $\mathcal{S}$, Φ is the output of Filter computed by the adversary. The tally oracle aborts if $\mathcal{S}$, Φ is not valid. If the adversary saw the "real" result corresponding to BB_0, the tally protocol proceeds as normal and publishes a correct tally proof Π with respect to BB_0. If the adversary saw the "fake" bulletin board BB_1, the experiment simulates the tally proof Π with respect to BB_1 using the algorithm SimTally and returns the real result r.

Definition 4. Consider a voting scheme $\mathcal{V} = ($Setup, GetToken, Vote, Valid, Filter, VerifyFilter, Tally, Verify$)$ for an electoral roll $\mathcal{E}$ and candidate list $\mathcal{C}$. We say the scheme has *ballot privacy* if there exists an algorithm SimTally such

that for all probabilistic polynomial time adversaries $\mathcal{A}$

$$\left| \Pr\left[\mathsf{Exp}_{\mathcal{A},\mathcal{V}}^{\mathsf{bpriv},0}(\ell,\mathcal{E},\mathcal{C}) = 1 \right] - \Pr\left[\mathsf{Exp}_{\mathcal{A},\mathcal{V}}^{\mathsf{bpriv},1}(\ell,\mathcal{E},\mathcal{C}) = 1 \right] \right|$$

is a negligible function in ℓ.

In the extended version of this paper [31], we prove the following theorem.

Theorem 1. VOTEAGAIN *provides ballot privacy under the DDH assumption in the random oracle model.*

Bernhard et al. [6] also define strong consistency, to ensure that the result r does not leak information about individual ballots, and strong correctness to ensure that valid ballots are never refused by the bulletin board. We restate these notions and prove that VOTEAGAIN satisfies them in the extended version of this paper [31].

6.2 Coercion resistance

Coercion resistance means that a coercer should not be able to determine whether a coerced user submitted to coercion – assuming it cannot learn this by seeing the result of the election (e.g., if there are zero votes for the selected candidate, the coercer knows the coerced user did not submit). In VOTEAGAIN, this means that the coercer should not be able to determine whether a coerced user voted again, or not.

Existing coercion resistant models are insufficient. Juels, Catalano and Jakobsson (JCJ) model coercion resistance by comparing a real-world game with an ideal game [29]. In JCJ, voters evade coercion by providing the coercer with a fake credential. The real-world models normal execution. The adversary plays the role of the coercer and chooses a set of corrupted voters and identifies the coerced voter. Then, the honest voters cast their ballots (or abstain). If the coerced voter does not submit she also casts her true ballot. Thereafter, the adversary is given the credentials of all corrupt users, a credential for the coerced voter (which is fake if that voter resists), and the current bulletin board. The adversary can now cast more ballots. Upon seeing the result and the tally proof the adversary decides if the coerced voter submitted. In the ideal game, the adversary is not shown the content of the bulletin board, and she is given the true credential of the coerced voter and can therefore cast real ballots for the coerced voter. However, a modified tally function does not count ballots for the coerced voter cast by the adversary if the coerced voter resists. Once the election phase is over, the adversary is shown only the tally result, not the tally proof.

The JCJ model does not work for the revoting setting where the coerced voter casts another ballot *after* casting the ballot under coercion. Achenbach et al. [1] propose a variant in which the coerced voter acts after the adversary has cast his votes, revoting if she resists or doing nothing if she submits. Thereafter, the adversary is shown the new bulletin board and

the resulting tally and proof. In the ideal model, the adversary is only provided the length of the bulletin board.

The model proposed by Achenbach et al. [1] does not capture coercion resistance. Following the real/ideal paradigm, in the ideal game it should hold with overwhelming probability that the adversary cannot distinguish between a submitting and a resisting coerced voter. Then, the proof would show that the adversary cannot learn more in the real world than it could in the ideal world. However, in the ideal game proposed by Achenbach et al., the coercion resistance property does not hold. The adversary can *always* distinguish between these two cases by simply observing the length of the bulletin board (which increases by one ballot if the coerced voter revotes). Therefore, proofs in this model say nothing about whether the real scheme offers coercion resistance. The Achenbach et al. [1] scheme seems to be coercion resistant, but coercion resistance does not follow from the proof in their model.

Finally, the model by Achenbach et al. does not capture the leakage resulting from the state kept by the voter, or as in our protocol, by the polling authority. Our protocol deliberately hides the ballot counter from the voter, so that if the coercer coerces the voter again, it cannot determine whether the coerced voter re-voted based on this counter. Achenbach et al.'s model does not capture this property, as the coercer is not allowed to coerce a voter more than once.

A new coercion resistance definition. We propose a new game-based coercion resistance definition inspired by Bernhard et al.'s ballot privacy definition. The game tracks two bulletin boards, BB_0 and BB_1, of which only one is accessible to the adversary (depending on the bit b). We ensure that regardless of the bit b, the same number of ballots are added to the bulletin board. The goal of the adversary is to determine b (see Figure 8). Recall that we assume that the PA, TS, and trustees are honest with respect to coercion resistance. We model honesty of the PBB (respectively, the use of an anonymous communication channel) by not revealing which voter posted to the bulletin board.

To model submits versus resists, we provide the adversary with an $O\mathsf{voteLR}(i_0, c_0, i_1, c_1)$ oracle to let voter i_0, a "coerced" voter, cast a vote for candidate c_0 in BB_0, and voter i_1, any other voter, cast a vote for candidate c_1 in BB_1. The adversary is allowed to make this call multiple times. Regardless of the value of b, every call to $O\mathsf{voteLR}$ results in a single ballot being added to each bulletin board. This prevents the trivial win in the Achenbach et al. model. Since the polling authority keeps state, we work with two PAs: PA_0 and PA_1.

We model a coercion attack as follows. The adversary can cast votes using any user by calling $O\mathsf{gettoken}(i)$ to obtain a voting token τ for voter i on the board that it can see, and a token τ' for the other board. It can then run $\beta = \mathsf{Vote}(\tau, c)$ and $\beta' = \mathsf{Vote}(\tau', c)$ itself to create ballots for candidate c, on both boards and cast them using $O\mathsf{cast}(\beta, \beta')$. Note that per our assumptions, the adversary does not get access to the voter's means of authentication. Moreover, we require that the adversary always casts valid ballots to both boards (but the encoded candidate need not be the same).

Finally, the adversary can make one call to $O\mathsf{tally}()$ which performs the filtering step and returns the result r (always computed on BB_0) and the tally proof. The result of Filter is accessible using $O\mathsf{board}$. To correct for leakage stemming from the tally result, as in the ballot privacy game, we simulate the filter and tally proofs if the adversary sees BB_1.

This game models all the coercion attacks applicable to VOTEAGAIN:

- *The 1009 attack.* The adversary casts a ballot as coerced voter i_0 using $\tau, \tau' = O\mathsf{gettoken}(i_0)$, $\beta = \mathsf{Vote}(\tau, c)$, $\beta' = \mathsf{Vote}(\tau', c)$ and then $O\mathsf{cast}(\beta, \beta')$ 1009 times. (Both boards now contain 1009 ballots by voter i_0.) Then it calls $O\mathsf{voteLR}(i_0, c, i_1, c)$. If $b = 0$ the coerced voter revotes for candidate c on BB_0, otherwise it does not, and the alternative voter casts a ballot for candidate c on BB_1 visible to the adversary. Note that if the result of Filter Φ reveals the size of a group of ballots, the adversary can win this game (SimFilter does not model this leakage as it only gets n'_B and r as input).

- *Returning coercer.* Let voter i_0 be the coerced voter. First the coercer runs $\tau, \tau' = \mathsf{GetToken}(i_0)$, $\beta = \mathsf{Vote}(\tau, c)$ and $\beta' = \mathsf{Vote}(\tau', c)$, and $O\mathsf{cast}(\beta, \beta')$ to cast one vote as the coerced user on both boards and to observe the token τ corresponding to the board BB_b it can see. Then it runs $O\mathsf{voteLR}(i_0, c_0, i_1, c_1)$, causing i_0 to cast a vote on the bulletin board BB_b if $b = 0$, and i_1 to casts a vote on BB_b if $b = 1$. Thereafter, it can examine the state by running $\tau, \tau' = \mathsf{GetToken}(i_0)$ again. If the new token τ leaks whether voter i_0 voted again (on board BB_b), then the adversary wins the coercion resistance game.

Definition 5. Consider a voting scheme $\mathcal{V} = (\mathsf{Setup}, \mathsf{GetToken}, \mathsf{Vote}, \mathsf{Filter}, \mathsf{VerifyFilter}, \mathsf{Tally}, \mathsf{Verify})$ for an electoral roll $\mathcal{E}$ and candidate list $\mathcal{C}$. We say the scheme has *coercion resistance* if there exist algorithms SimFilter and SimTally such that for all probabilistic polynomial time adversaries $\mathcal{A}$

$$\left| \Pr\left[\mathsf{Exp}_{\mathcal{A},\mathcal{V}}^{cr,0}(\ell, \mathcal{E}, \mathcal{C}) = 1 \right] - \Pr\left[\mathsf{Exp}_{\mathcal{A},\mathcal{V}}^{cr,1}(\ell, \mathcal{E}, \mathcal{C}) = 1 \right] \right|$$

is a negligible function in ℓ.

In Appendix A, we prove the following theorem.

Theorem 2. VOTEAGAIN *provides coercion resistance under the DDH assumption in the random oracle model.*

6.3 Verifiability

In their analysis, Achenbach et al. [1] adapt the correctness definition of Juels et al. [29] to the revoting setting. However, Achenbach et al.'s model does not take into account that voters may not check that their ballots are cast correctly, nor that

$\mathsf{Exp}^{\mathsf{cr},b}_{\mathcal{A},\mathcal{V}}(\ell,\mathcal{E},\mathcal{C})$:
    $(\mathsf{pk},\mathsf{sk}_{\mathsf{PA}},\mathsf{sk}_{\mathsf{TS}},\mathsf{sk}_{\mathsf{T}}) \leftarrow \mathsf{Setup}(1^\ell,\mathcal{E},\mathcal{C})$
    Create PA_0 and PA_1 with keys $\mathsf{pk}^0_{\mathsf{PA}},\mathsf{pk}^1_{\mathsf{PA}}$
    $b' \leftarrow \mathcal{A}^O(\mathsf{pk},\mathsf{pk}^b_{\mathsf{PA}},\mathsf{pk}^{1-b}_{\mathsf{PA}})$
    Output b'

$O\mathsf{voteLR}(i_0,c_0,i_1,c_1)$:
    Let $\tau_0 \leftarrow \mathsf{PA}_0.\mathsf{GetToken}(i_0)$ and $\tau_1 \leftarrow \mathsf{PA}_1.\mathsf{GetToken}(i_1)$
    Let $\beta_0 = \mathsf{Vote}(\tau_0,c_0)$ and $\beta_1 = \mathsf{Vote}(\tau_1,c_1)$
    If $\mathsf{Valid}(\mathsf{BB}_b,\beta_b) = \bot$ return $\bot$
    Else $\mathsf{BB}_0 \leftarrow \mathsf{BB}_0 \| \beta_0$ and $\mathsf{BB}_1 \leftarrow \mathsf{BB}_1 \| \beta_1$

$O\mathsf{gettoken}(i)$:
    Let $\tau_0 \leftarrow \mathsf{PA}_0.\mathsf{GetToken}(i)$ and $\tau_1 \leftarrow \mathsf{PA}_1.\mathsf{GetToken}(i)$
    return $\tau = \tau_b, \tau' = \tau_{1-b}$

$O\mathsf{cast}(\beta,\beta')$:
    Let $\beta_b \leftarrow \beta$ and $\beta_{1-b} \leftarrow \beta'$
    If $\mathsf{Valid}(\mathsf{BB}_0,\beta_0) = \bot$ or $\mathsf{Valid}(\mathsf{BB}_1,\beta_1) = \bot$ return $\bot$
    Else $\mathsf{BB}_0 \leftarrow \mathsf{BB}_0 \| \beta_0$ and $\mathsf{BB}_1 \leftarrow \mathsf{BB}_1 \| \beta_1$

$O\mathsf{board}()$:
    return BB_b

$O\mathsf{tally}()$
    Let n'_B be the number of tokens obtained from PA_0.
    Let $\mathcal{S}_0,\Phi_0 = \mathsf{Filter}(\mathsf{BB}_0,n'_B,\mathsf{sk}_{\mathsf{TS}})$
    Let $(r,\Pi_0) \leftarrow \mathsf{Tally}(\mathsf{BB}_0 \| \mathcal{S}_0 \| \Phi_0,\mathsf{sk}_{\mathsf{T}})$
    Let $(\mathcal{S}_1,\Phi_1) \leftarrow \mathsf{SimFilter}(\mathsf{BB}_1,n'_B,r)$
    Let $\mathsf{BB}_0 \leftarrow \mathsf{BB}_0 \| \mathcal{S}_0 \| \Phi_0$ and $\mathsf{BB}_1 \leftarrow \mathsf{BB}_1 \| \mathcal{S}_1 \| \Phi_1$
    $\Pi_1 = \mathsf{SimTally}(\mathsf{BB}_1,r)$
    return (r,Π_b)

Figure 8: In the coercion resistance experiment $\mathsf{Exp}^{\mathsf{cr},b}_{\mathcal{A},\mathcal{V}}$, adversary $\mathcal{A}$ has access to oracles $O = \{O\mathsf{voteLR}, O\mathsf{gettoken}, O\mathsf{cast}, O\mathsf{board}, O\mathsf{tally}\}$. It can call $O\mathsf{tally}$ only once, thereafter it can see the result Φ_b by using $O\mathsf{board}()$.

newer ballots should supersede older ballots *even* if voters have been coerced or corrupted. To address these cases, we adapt the qualitative game-based verifiability definition of Cortier et al. [12] – which accounts for a malicious bulletin board and voters not checking their ballots – to our setting by adding the GetToken function and explicitly modeling revoting. As in Cortier et al. [12], our game does not model voter's intent, and assumes that the voting hardware, i.e., the device and software running Vote, is honest. We refer to Cortier et al. [13] for a formal process-based computational model that does model verifiability with voter intent. We note that the correctness definition by Juels et al. [29] was renamed to 'verifiability' by Cortier et al. [12], and therefore any model satisfying the latter also satisfies the former.

In a nutshell, a voting scheme is verifiable [12] if for n_C corrupt voters, the result of the election always includes: (1) all votes by honest voters that verified whether their ballots were cast correctly, (2) at most n_C corrupted votes, and (3) a subset of the votes by honest voters that did not check if their ballots were cast correctly. These conditions ensure that while a malicious bulletin board can drop ballots of voters that do not check, it can insert at most n_C new votes.

Extending the current verifiability definition. We extend the definition presented by Cortier et al. [12] for the revoting setting to explicitly consider the number of votes cast by a voter, see Figure 9. The PA is honest, but the adversary controls the bulletin board, the TS, and the trustees. The system implicitly tracks the number of tokens #tokens(i) that have been obtained by voter i. The game tracks when each voter is corrupted in a (initially empty) list of corruption events C, and tracks the honest votes in HVote. The adversary can call two oracles: $O\mathsf{vote}(i,c)$ to request that honest voter i outputs a ballot for candidate c, and $O\mathsf{gettoken}(i)$ to get a voting token for user i. This models *both corruption and coercion* of voter i. After a call to $O\mathsf{gettoken}(i)$, voter i is considered corrupted until it casts an honest ballot using $O\mathsf{vote}(i,c)$. Eventually,

the adversary outputs a bulletin board BB, the selected votes $\mathcal{S}$ and proof Φ, the election outcome $r \in \mathbb{R}$, and a tally proof Π (line 3). The adversary loses if Φ or Π do not verify (line 4). If it verifies, the adversary wins if the result does not satisfy the three intuitive conditions above.

The game computes the following groups of voters:

- Corrupted (line 6): voters considered corrupted, i.e., voters that were once corrupted (by calling $O\mathsf{gettoken}$) and thereafter never cast a checked honest vote.
- Checked (line 7): voters that verified a ballot and were not corrupted thereafter.
- Unchecked (line 8): voters that were never corrupted, but did not check their ballots either.

The game computes allowed candidates for honest voters:

- AllowedVotes[i] (line 9): A list of candidates that voter i honestly voted for in or after the last checked ballot. If voter i never checked a ballot, this list includes all candidates this voter ever voted for.

The adversary wins if the result r verifies but violates any of the following conditions (lines 10–13): (1) For each honest voter that verified a ballot and was not thereafter corrupted (i.e., voters in Checked) the result should include either the candidate in that ballot, or a candidate in a later ballot. This corresponds to the candidates $\{c_i^V\}_{i=1}^{n_V}$ in the game. (2) Of the honest voters that did not check their ballots but were never corrupted (i.e., voters in Unchecked), at most one candidate that the honest voter voted for (in any ballot) can be included. This corresponds to the candidates $\{c_i^U\}_{i=1}^{n_U}$ in the game. Note that n_U can be smaller than |Unchecked| or in fact zero. (3) At most n_C corrupted (or bad) votes were counted (i.e., the candidates $\{c_i^B\}_{i=1}^{n_B}$).

In the game, the sum of these choices is modeled by the tallying function $\bar{\rho} : \mathcal{C}^* \to \mathbb{R}$ that maps the voter's choices in $\mathcal{C}$ to an election result in $\mathbb{R}$. This function should support

```
0   Exp_{A,V}^{ver,b}(ℓ, E, C):
1   (pk, sk_PA, sk_TS, sk_T) ← Setup(1^ℓ, E, C)
2   Set HVote ← ∅ and C ← ∅
3   (BB, S, Φ, r, Π) ← A^O(pk, sk_TS, sk_T)
4   If VerifyFilter(BB, S, Φ) = ⊥ or Verify(BB ∥ S ∥ Φ, r, Π) = ⊥ return 0
5   Let Verified = {(i_1, ctr_1), ..., (i_λ, ctr_λ)} correspond to checked ballots.
6   Let Corrupted = {i | (i, ctr) ∈ C ∧ ∀(i, ctr') ∈ Verified : ctr' < ctr}
7   Let Checked = {i | (i, _) ∈ Verified} \ Corrupted
8   Let Unchecked = {i | (i, _, _) ∈ HVote ∧ (i, _) ∉ C} \ Checked
9   Let AllowedVotes[i] = {c | (i, ctr, c) ∈ HVote s.t. ∀(i, ctr') ∈ Verified : ctr ≥ ctr'}
10  If ∃ c_1^V, ..., c_{n_V}^V s.t. c_j ∈ AllowedVotes[i_j^V] where Checked = {i_1^V, ..., i_{n_V}^V}
11     ∃ (i_1^U, c_1^U), ..., (i_{n_U}^U, c_{n_U}^U) s.t. i_j^U ∈ Unchecked, c_j^U ∈ AllowedVotes[i_j^U], i_j^U different
12     ∃ c_1^B, ..., c_{n_B}^B ∈ C s.t. 0 ≤ n_B ≤ |Corrupted|
13     s.t. r = ρ̄({c_i^V}_{i=1}^{n_V}) ⋆_R ρ̄({c_i^U}_{i=1}^{n_U}) ⋆_R ρ̄({c_i^B}_{i=1}^{n_B})
14  Then return 0, otherwise return 1
```

```
Ovote(i, c):
    Let τ = GetToken(i)
    Add (i, #tokens(i), c) to HVote
    Return Vote(τ, c)

Ogettoken(i):
    Let τ = GetToken(i)
    Add (i, #tokens(i)) to C
    return τ
```

Figure 9: In the verifiability game experiment $Exp_{A,V}^{ver,b}$, the adversary A has access to the oracles $O = \{Ogettoken, Ovote\}$.

partial tallying, i.e., for any two lists S_1 and S_2 we have that $\bar{\rho}(S_1 \cup S_2) = \bar{\rho}(S_1) \star_R \bar{\rho}(S_2)$ for a commutative binary operator $\star_R : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$. Note that a tally function that outputs the number of votes per candidate naturally admits partial tallying.

Definition 6. Consider a voting scheme $V = ($Setup, GetToken, Vote, Filter, VerifyFilter, Tally, Verify$)$ for an electoral roll E and candidate list C. We say the scheme is *verifiable* if for all probabilistic polynomial time adversary A

$$\left| \Pr\left[Exp_{A,V}^{ver,0}(\ell, E, C) = 1 \right] - \Pr\left[Exp_{A,V}^{ver,1}(\ell, E, C) = 1 \right] \right|$$

is a negligible function in ℓ.

In the extended version of this paper [31], we prove the following theorem.

Theorem 3. VOTEAGAIN *is verifiable under the DDH assumption in the random oracle model.*

7 Performance Evaluation

We evaluate the performance of VOTEAGAIN using a Python prototype implementation of its core cryptographic operations.[2] We did not implement the GetToken protocol, but we note that as it relies on standard cryptography it can be implemented easily and cheaply; nor did we implement the bulletin board as it is not core to our design. We use the petlib [14] binding to OpenSSL for the group operations using the fast NIST P-256 curve. We ran all experiments in Linux on a single core of an Intel i3-8100 processor running at 3.60GHz. We expect nation-wide elections to have much more processing power available. For example, the Swiss CHVote system,

which aims to support 8 million voters, has around 32 cores available per party in the system. We also include performance *estimates* of running the system on a large machine with 8 Intel Xeon Platinum 8280L processors with 28 cores each, running at 2.7Ghz. As our scheme is almost completely parallelizable (only the hash functions for the non-interactive zero-knowledge proofs need to be computed sequentially), we estimate a 90% parallelization gain: a speedup of 170 times when using the 8x28 cores with respect to the single core.

For all experiments we empirically select the best cover size k by sweeping over values from 1 to 64. In the majority of cases the optimal k is in the range $[2, 4]$.

Creating a ballot. We use an ElGamal ciphertext to encrypt the voter's choice, and a Bayer and Groth [5] zero-knowledge proof of membership to show that the selected candidate is eligible. Creating a ballot from 1000 eligible candidates costs 1.2 seconds, while verifying its correctness costs 0.17 seconds. The size of this proof is 1.5 kB.

Impact of revoting. Figure 10 shows the overhead depending on the number of votes, in terms of number of dummies per real ballot. This overhead influences the computation time of shuffling and filtering in the tally phase. In the leftmost figure we model users' revoting behaviour as a percentage of the number of voters: 50% models that half of the voters revoted once, and 200% models that all voters revote twice. We note that the overhead of 100% voters revoting once is equivalent to, for example, 25% of the voters revoting 4 times. As expected, the overhead increases with both the number of voters and the number of revoted ballots. However, even for 100 million voters revoting twice (200% revotes), the overhead is at most a factor of 32 (Figure 10 left).

Casting a vote takes time. Thus, revoting patterns are constrained by the number of ballots that can be cast during an election. We consider an election period of 24h (larger than most countries), and bound how often a *single* voter can vote

---

[2]The code is open source and can be found here: https://github.com/spring-epfl/voteagain.
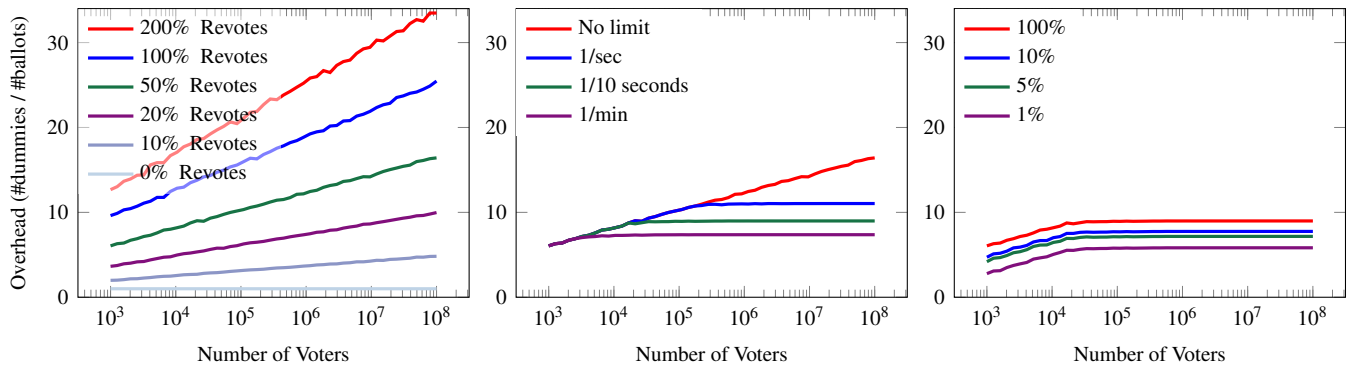
Figure 10: Dummy ballots overhead: Varying percentages of extra ballots with respect to the total number of voters (left); effect of a rate-limit on revoting voters assuming at most 50% extra ballots (center); and limiting the percentage of voters that revotes assuming at most 50% extra ballots and a rate-limit of 1 ballot per 10 seconds (right).
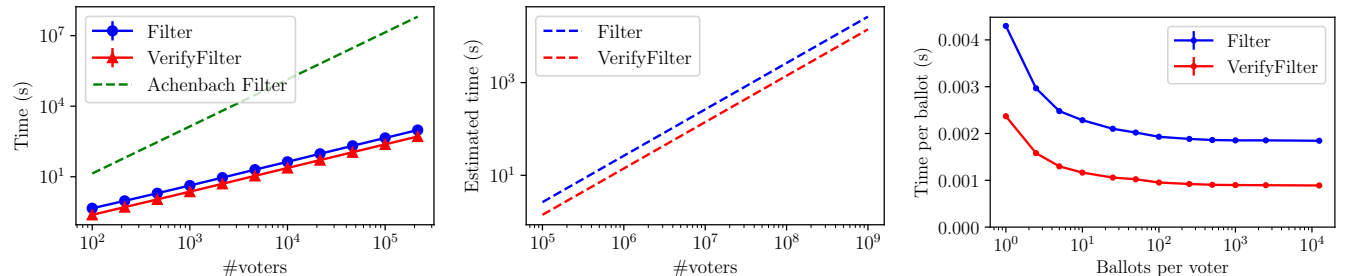

Figure 11: Cost of Filter and VerifyFilter: Measured cost on single core (left); estimated cost on 8 processor machine (8×28 cores, center); and effect of different distributions of 50 000 ballots (including dummies) among voters (right). Note that one ballot per voter causes the highest processing time.

(1 ballot per second, per ten seconds, and per minute). As this limits the number of voters with a large amount of ballots, we do not need large covers, reducing the overhead (see Figure 10, center). Similarly, assuming that all voters will revote is very conservative. In a normal election one expects the vast majority of voters to vote once. In Figure 10, right, we show the overhead when the number of voters that cast more than one vote is limited. As fewer voters revote, the total amount of votes is smaller and so are the covers.

Filtering. We implemented a non-optimized version of Bayer-Groth's verifiable shuffle protocol [4] to implement steps 3 and 6 of Procedure 5. We measure the execution time of filtering and verifying, when varying the number of voters. Figure 11 left shows the times to run Filter and VerifyFilter on a single core machine. Figure 11 middle shows the estimated processing times on the big 8 processor Xeon machine. We estimate that the 8 processor machine can filter and tally the second round presidential election in Brazil (147 million registered voters) in 65 minutes if no voter revotes, and within 11 hours assuming 50% extra ballots and at most one ballot per voter per ten seconds. We note that elections usually tally ballots per state, city, or smaller electoral district. In general we expect the number of ballots to be much smaller. All ballot groups in Figure 11 left and center have size one. Figure 11 right shows the effect of larger ballot groups result-ing from revoting and dummy voters. As the average group size increases, the computation time goes down. Therefore, Figure 11 gives an upper bound on the processing time, given a known cover size.

For comparison we computed a lower bound on the filter cost of Achenbach et al.'s filter method by counting the number of group operations needed per ballot. We used this number to compute the estimate in Figure 11 left. A small-town election with 100.000 ballots takes 5 core months to filter in their scheme. Even on the large Xeon machine, an election with 1 million ballots takes over four months to complete. Our method needs respectively 7 core minutes and 30 seconds. The sizes of the tally proofs in VOTEAGAIN for these examples are 54 and 501 MB respectively.

Smaller regions. Many countries report election results per region, such as a province, a city, or a neighborhood. In those cases, results can be computed per region at lower computation cost. However, even in this setting, Achenbach et al.'s quadratic approach scales poorly. We note that the allowable size of reporting regions depend on local regulations, with the smallest regions likely being cities or neighborhoods, which can easily total 100.000s of voters. As Figure 11 (left) shows, even in this configuration, Achenbach et al.'s quadratic approach requires 3 to 4 orders of magnitude more computation resources than VOTEAGAIN.

Tallying. We also measured the execution time of a single step of the mix network – a single shuffle and one verifiable decryption – using our verifiable shuffle implementation. Our results show that one step is a factor of three times faster than our filter protocol, e.g., mix-and-decrypting the 100.000 ballots takes less than 2 core minutes and 1 million ballots take less than 7 seconds on the Xeon machine.

8 Conclusion

Due to its complexity and cost, coercion resistance has been often overlooked in remote voting schemes. We introduced VOTEAGAIN, a revoting scheme that enables cleartext filtering thanks to efficient deterministic padding. VOTEAGAIN does not require users to store cryptographic material, and can efficiently handle millions of votes. We provided a new coercion resistance definition and updated existing definitions for ballot privacy and verifiability to the revoting setting. We have proven that VOTEAGAIN satisfies all of them.

Acknowledgments

References

[1] Dirk Achenbach, Carmen Kempka, Bernhard Löwe, and Jörn Müller-Quade. Improved Coercion-Resistant Electronic Elections through Deniable Re-Voting. *USENIX Journal of Election Technology and Systems (JETS)*, (2), 2015.

[2] Ben Adida. Helios: Web-based Open-audit Voting. In *USENIX*, 2008.

[3] Roberto Araújo, Amira Barki, Solenn Brunet, and Jacques Traoré. Remote Electronic Voting Can Be Efficient, Verifiable and Coercion-Resistant. In *VOTING*, 2016.

[4] Stephanie Bayer and Jens Groth. Efficient zero-knowledge argument for correctness of a shuffle. In *EUROCRYPT*, 2012.

[5] Stephanie Bayer and Jens Groth. Zero-knowledge argument for polynomial evaluation with application to blacklists. In *EUROCRYPT*, 2013.

[6] David Bernhard, Véronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. Sok: A comprehensive analysis of game-based ballot privacy definitions. In *S&P*, 2015.

[7] Philippe Bulens, Damien Giry, and Olivier Pereira. Running mixnet-based elections with helios. In *EVT/WOTE*, 2011.

[8] Sergiu Bursuc, Gurchetan S Grewal, and Mark D Ryan. Trivitas: Voters Directly Verifying Votes. In *VOTE-ID*, 2012.

[9] Jan Camenisch and Markus Stadler. Efficient Group Signature Schemes for Large Groups (Extended Abstract). In *CRYPTO*, 1997.

[10] Jeremy Clark and Urs Hengartner. Selections: Internet Voting with Over-the-Shoulder Coercion-Resistance. In *FC*, 2012.

[11] Michael R. Clarkson, Stephen Chong, and Andrew C. Myers. Civitas: Toward a secure voting system. In *S&P*, 2008.

[12] Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachène. Election Verifiability for Helios under Weaker Trust Assumptions. In *ESORICS*, 2014.

[13] Véronique Cortier, David Galindo, Ralf Küsters, Johannes Müller, and Tomasz Truderung. SoK: Verifiability Notions for E-Voting Protocols. In *S&P*, 2016.

[14] George Danezis. Petlib: A python library that implements a number of privacy enhancing technolgies. https://github.com/gdanezis/petlib. Accessed: June 22, 2020.

[15] Official e Estonia Website. e-governance / i-voting, accessed May 21 , 2014. https://e-estonia.com/solutions/e-governance/i-voting/.

[16] Aleksander Essex, Jeremy Clark, and Urs Hengartner. Cobra: Toward concurrent ballot authorization for internet voting. In *EVT/WOTE*, 2012.

[17] Amos Fiat and Adi Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *CRYPTO*, 1986.

[18] Conner Fromknecht, Dragos Velicanu, and Sophia Yakoubov. A Decentralized Public Key Infrastructure with Identity Retention. Cryptology ePrint Archive, Report 2014/803, 2014.

[19] Christina Garman, Matthew Green, and Ian Miers. Decentralized Anonymous Credentials. In *NDSS*, 2014.

[20] Dawid Gaweł, Maciej Kosarzecki, Poorvi L. Vora, Hua Wu, and Filip Zagórski. Apollo – End-to-end verifiable internet voting with recovery from vote manipulation. In *E-VOTE-ID*, 2016.

[21] Kristian Gjøsteen. Analysis of an Internet Voting Protocol. Cryptology ePrint Archive, Report 2010/380, 2010.

[22] S Goldwasser, S Micali, and C Rackoff. The Knowledge Complexity of Interactive Proof-systems. In *STOC*, 1985.

[23] Rüdiger Grimm and Melanie Volkamer. Multiple Cast in Online Voting – Analyzing Chances. In *Electronic Voting*, 2006.

[24] Rolf Haenni, Reto E. Koenig, Philipp Locher, and Eric Dubuis. CHVote System Specification. Cryptology ePrint Archive, Report 2017/325, 2017.

[25] James Heather and David Lundin. The Append-Only Web Bulletin Board. In *Formal Aspects in Security and Trust*, 2009.

[26] Sven Heiberg and Jan Willemson. Verifiable internet voting in Estonia. In *2014 6th International Conference on Electronic Voting: Verifying the Vote (EVOTE)*, pages 1–8. IEEE, 2014.

[27] Internet Voting Task Force. Findings and assessment report of Internet Voting Task Force (IVTF) on voting rights of overseas Pakistanis, 2018. https://www.ecp.gov.pk/ivoting/IVTF%20Report%20Executive%20Version%201.5%20Final.pdf.

[28] Markus Jakobsson, Ari Juels, and Ronald L. Rivest. Making mix nets robust for electronic voting by randomized partial checking. In *USENIX Security Symposium*, 2002.

[29] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *WPES*, 2005.

[30] Philipp Locher, Rolf Haenni, and Reto E Koenig. Coercion-Resistant Internet Voting with Everlasting Privacy. In *FC*, 2016.

[31] Wouter Lueks, Iñigo Querejeta-Azurmendi, and Carmela Troncoso. VoteAgain: A scalable coercion-resistant voting system, 2020. https://arxiv.org/abs/2005.11189.

[32] Pippa Norris, Thomas Wynter, and Sarah Cameron. Corruption and coercion: The year in elections 2017, 2018.

[33] Gerald V. Post. Using re-voting to reduce the threat of coercion in elections. *Electronic Government*, 7(2):168–182, 2010.

[34] P. Y. A. Ryan, D. Bismark, J. Heather, S. Schneider, and Z. Xia. Prêt à voter: a voter-verifiable voting system. *IEEE Transactions on Information Forensics and Security*, 4(4):662–673, Dec 2009.

[35] Peter Y A Ryan, Peter B Rønne, and Vincenzo Iovino. Selene: Voting with Transparent Verifiability and Coercion-Mitigation. In *Financial Cryptography Workshop VOTING*, 2015.

[36] Scytl. Fully Verifiable and Secure Online Voting, 2018. https://www.scytl.com/en/online-voting-technology-security/.

[37] W.D. Smith. New cryptographic election protocol with best-known theoretical properties. In *Workshop Frontiers in Electronic Elections (FEE)*, 2005.

A Proof of Coercion Resistance

Proof of theorem 2. We first specify how to construct SimTally and SimFilter. As in the ballot privacy proof, SimTally(BB, r) simply simulates the proof of shuffle and the proof of correct decryption in Tally, so that regardless of the values in $\mathcal{S}$, r is the correct outcome.

The algorithm SimFilter(BB, n'_B, r) proceeds similarly. It takes as input the bulletin board BB, which it uses to determine the number of ballots n_B, the number of registrations n'_B, and the result r. Moreover, it derives the number of real voters n using r. It uses these data to compute the cover, and it adds the correct number of dummy ballots (for these, it sets γ and I to random ciphertexts) to obtain B'. Then it computes a list of zero ciphertexts (encryptions of zero) of equal length, and simulates the shuffle proof π_σ. It then generates fake voter identifiers $\overline{vid}$ and $\overline{m}$ corresponding to the cover it computed earlier, associates these to shuffled ballot β_i, and simulates the proofs π_i^{dec}. Next, for each resulting group, it generates a random encryption of zero $\overline{V}_j = \text{Vote.ZEnc}(\text{pk}_\text{T}, r_j)$ and simulates the corresponding proof π_j^{sel}. Then, it returns the randomness r_j and the indices of the dummy voters corresponding to the cover it computed early. Finally, for each remaining vote, it generates a random V_j and simulates the shuffle proof π'_σ.

In this proof, we will step by step replace all the ciphertexts that depend on the bit b by random ciphertexts. In particular, we first show that the adversary learns nothing about b during the election phase. We then show that it also learns nothing about b during the tally phase. The result follows.

Game G_1. Game G_1 is as the $\text{Exp}_{\mathcal{A},\mathcal{V}}^{\text{cr},b}(\ell, \mathcal{E}, \mathcal{C})$ experiment. (Note that contrary to the proof of ballot privacy we do not fix the value for b.)

Game G_2. Game G_2 is as game G_1, but we compute the result directly based on the ballots on BB_0. Let $[\beta_1, \ldots, \beta_{n_B}]$ be the list of ballots where

$$\beta_i = (\text{v}_i, \pi_i, \text{pk}_i, \gamma_i, I_i, \sigma_i^\tau, \sigma_i).$$

Let $c_i = \text{Vote.Dec}(\text{sk}, \text{v}_i)$, $vid_i = \text{EC.Dec}(\text{sk}_\text{TS}, \gamma_i)$, and $m_i = \text{EC.Dec}(\text{sk}_\text{TS}, I_i)$. Then compute the result:

$$r = \rho(((vid_1, m_1), c_1), \ldots, ((vid_{n_B}, m_{n_B}), c_{n_B}))$$

As per strong consistency, games G_2 and G_1 are indistinguishable.

Game G_3. Game G_3 is as game G_2, but with all the zero-knowledge proofs replaced by simulations. This includes the shuffle proof π_σ, the decryption proofs of π_i^{dec} of the shuffled γ_i' and I_i's, the reencryption proofs π_i^{sel}, and the shuffle proof π_σ' produced in Filter; as well as the tally proof Π_0 which we replace by the output of $\mathsf{SimTally}(\mathsf{BB}_0, r)$. We use the random oracle to simulate this step, which is indistinguishable by the simulatability of the zero-knowledge proof system.

Game G_4. Game G_4 is as game G_3 but we do not decrypt the γ_i and I_i anymore when running Filter. Instead, we proceed as follows. All ballots $\beta_i = (\mathsf{v}_i, \pi_i, \mathsf{pk}_i, \gamma_i, I_i, \sigma_i^\tau, \sigma_i)$ on the bulletin boards are valid. Hence, σ_i is a valid signature by PA_0 resp. PA_1 on γ_i and I_i. Since the signature scheme is unforgeable, we know these ciphertexts were created by PA_0 resp. PA_1. Hence, we can associate to them the corresponding plaintexts vid_i and m_i. Moreover, we know the permutation used by the TS during Filter, so we can also provide the correct plaintexts in step 4 of Filter on BB_0 (recall the proofs of decryption π_i^{dec} are already simulated).

Game G_5. Game G_5 is as game G_4, but we replace the ciphertexts γ_i and I_i in the token τ_i by random ciphertexts for all tokens. Similarly, we replace the γ_i and I_i ciphertexts for the dummy ballots by random ciphertexts. Note that per the change in game G_4 we still associate the correct plaintexts vid_i and m_i in the Filter protocol. A hybrid argument with reductions to the CPA security of the ElGamal encryption scheme shows that games G_5 and G_4 are indistinguishable. This reduction is possible since we no longer need to decrypt these ciphertexts.

Game G_6. Game G_6 is as game G_5, but we replace the encrypted votes v_i in the $\mathcal{O}\mathsf{voteLR}()$ call by encryptions of the zero vector, i.e., $\mathsf{v}_i = \mathsf{Vote.ZEnc}(\mathsf{pk}_\mathsf{T}, r)$ for a uniformly random randomizer r. As in the ballot privacy proof, a hybrid argument with a reduction to the NM-CPA security of the ElGamal encryption scheme with zero-knowledge proof shows that games G_6 and G_5 are indistinguishable. Note that in this reduction we use the $\mathcal{O}\mathsf{dec}$ of the NM-CPA challenger to decrypt votes in the adversary-determined ballots before computing the result r.

Note that as of game G_6, the adversary's view of the bulletin board before calling $\mathcal{O}\mathsf{tally}()$ is independent of the value of b. (The ballots resulting from the $\mathcal{O}\mathsf{voteLR}$ call also contain a

random ephemeral public key pk and the signatures σ^τ and σ, but these are also independent of the actual voter selected.)

We now proceed to show that the adversary also cannot learn anything from the output of Filter. Notice that, regardless of the value of b, the filter step is computed with the same number of voters ν, the same number of ballots n_B and the same number of obtained tokens n_B'. Therefore, the output of Filter applied to BB_0 and that of SimFilter applied to BB_1 should be indistinguishable. In the following game steps we replace the ciphertexts after shuffling by zero-ciphertexts and show that these steps are indistinguishable for the adversary.

Game G_7. Game G_7 is the same as game G_6, but we replace the ciphertexts γ_i', I_i' and θ_i' after shuffling by random encryptions of zero. We proceed as if they still decrypt to the correct values. Note that we already simulate the shuffle proof and decryption proofs. Again, a hybrid argument with reductions to the CPA security of the ElGamal encryption scheme shows that the games G_7 and G_6 are indistinguishable. This reduction is possible since we no longer need to decrypt these ciphertexts.

Game G_8. Game G_8 is the same as game G_7, but we replace the shuffled encrypted votes v_i' by random encryptions of zero. Similarly, we replace the randomizations, R, of the votes corresponding to dummy voters by the corresponding new randomization. This causes the pre-selected votes $\overline{V}_j$ per group to be incorrect, but this does not matter as we simulate the second shuffle proof, π_σ', anyway. As before, the indistinguishability of this step follows from the NM-CPA security of the vote encryption scheme.

Game G_9. Game G_9 is the same as game G_9, but we replace the second shuffled votes V_j by random encryption of zero. This causes the selected votes V_j after the shuffle to be incorrect with respect to the result, but this does not matter as we simulate the proof of the tally. As before, the indistinguishability of this step follows from the NM-CPA security of the vote encryption scheme.

Game G_{10}. Game G_{10} is as game G_8, but we replace the filter and tally proofs on BB_0 by simulations: we set $(\mathcal{S}_0, \Phi_0) \leftarrow \mathsf{SimFilter}(\mathsf{BB}_0, n_B', r)$ and $\Pi_0 \leftarrow \mathsf{SimTally}(\mathsf{BB}_0, r)$. Note that this difference is purely syntactic, as per the changes we made before, we already computed exactly the output of SimFilter on BB_0 and the result r.

Clearly the resulting view is independent of b. And coercion resistance follows. $\qquad\square$