Scalable Coercion-Resistant E-Voting under Weaker Trust Assumptions

Thomas Haines
Australian National University
Canberra, Australia
thomas.haines@anu.edu.au

Johannes Müller
University of Luxembourg
Esch-Sur-Alzette, Luxembourg
johannes.mueller@uni.lu

Iñigo Querejeta-Azurmendi
Independent researcher
Donostia, Spain
azinig@querejeta.me

ABSTRACT

Electronic voting (e-voting) is regularly used in many countries and organizations for legally binding elections. In order to conduct such elections securely, numerous e-voting systems have been proposed over the last few decades. Notably, some of these systems were designed to provide *coercion-resistance*. This property protects against potential adversaries trying to swing an election by coercing voters.

Despite the multitude of existing coercion-resistant e-voting systems, to date, only few of them can handle large-scale Internet elections efficiently. One of these systems, VoteAgain (USENIX Security 2020), was originally claimed secure under similar trust assumptions to state-of-the-art e-voting systems without coercion-resistance.

In this work, we review VoteAgain's security properties. We discover that, unlike originally claimed, VoteAgain is no more secure than a trivial voting system with a completely trusted election authority. In order to mitigate this issue, we propose a variant of VoteAgain which effectively mitigates trust on the election authorities and, at the same time, preserves VoteAgain's usability and efficiency.

Altogether, our findings bring the state of science one step closer to the goal of scalable coercion-resistant e-voting being secure under reasonable trust assumptions.

CCS CONCEPTS

• **Security and privacy** → **Formal security models**; **Privacy-preserving protocols**;

KEYWORDS

Electronic Voting, Verifiability, Privacy, Coercion-resistance, Attack

ACM Reference Format:
Thomas Haines, Johannes Müller, and Iñigo Querejeta-Azurmendi. 2023. Scalable Coercion-Resistant E-Voting under Weaker Trust Assumptions. In *Proceedings of ACM SAC Conference (SAC'23).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

SAC'23, March 27–April 2, 2023, Tallinn, Estonia

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In Australia, Brazil, Estonia, India, the US, and many other countries, systems for electronic voting (e-voting) are regularly used for legally binding political elections. In such high-stake elections, it is crucial to protect voters against being coerced to vote or not to vote for a certain candidate, to abstain from voting, or to sell their votes. A legal approach to mitigate the risk of coercion is to ensure that "[any] coercion of voters should be prohibited by penal laws and those laws should be strictly enforced", as required by the international standards of elections of the *UN Committee on Human Rights* [27]. Since, realistically, the risk of being penalized may not be sufficient to deter possible coercers, the threat of coercion must also be counteracted at a technical level. To this end, numerous e-voting systems have been designed that aim to protect against coercion (see, e.g., [1, 2, 6, 7, 12, 17, 22, 32]), or to mitigate its risk (see, e.g., [5, 14, 20, 28, 29]), by technical means. This property is called *coercion-resistance*.

Coercion-resistance. In a coercion-resistant e-voting system, each coerced voter has the option to run some *counter-strategy* instead of obeying the coercer. By running the counter-strategy, the coerced voter can achieve her own goal (e.g., to vote for her favorite candidate). At the same time, the coercer cannot distinguish whether the coerced voter followed his instructions (e.g., voted for the coercer's favorite candidate) or ran the counter-strategy. From a technical perspective, there exist three different approaches in the literature which implement this concept: *fake credentials*, *masking*, and *deniable vote updating*. We will briefly explain these different approaches next.

Fake credentials are used, for example, in [1, 6, 7, 12], and they work as follows. Each voter is provided with a unique and secret credential $\hat{c}$. A voter uses $\hat{c}$ to submit her vote when she is not under coercion. Otherwise, if a voter is under coercion, she can create a so-called *fake credential* c to submit her coerced vote. Since the voter's fake credential is invalid, the respective vote will be secretly removed by the voting authorities. At the same time, the fake credential c and the real one $\hat{c}$ are indistinguishable from a coercer's perspective.

The *masking technique* is employed, for example, in [2, 32]. Its idea is the following one. Each voter is provided with a unique and secret mask $\hat{m}$. A voter uses $\hat{m}$ to blind her actual vote $\hat{v}$ when she is not under coercion. Otherwise, if a voter is being coerced to vote for a different choice v, then she computes a fake mask m such that the resulting blinded vote still remains a vote for her actual vote $\hat{v}$.

In both the fake credential and masking approach, the counter-strategies appear to be hardly usable by human voters (see, e.g., [21,

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

Figure 1: Trust assumptions under which VoteAgain was originally claimed to provide the respective security properties. PA denotes the polling authority, TS the tally server, and PBB the public bulletin board. (*We consider the case that voters submit their ballots anonymously.)

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

Figure 2: Trust assumptions which we identified to be necessary in VoteAgain to provide the respective security properties. Differences to originally claimed trust assumptions (Fig. 1) are bold. Those trust assumptions that we were able to mitigate are marked with *.

26]) so that these two concepts may be rendered completely ineffective for real practical elections. Achieving coercion-resistance via deniable vote updating, as described next, is more promising.

The idea of e-voting with *deniable vote updating* (e.g., [17, 22]) is to enable each voter to overwrite her previously submitted ballot, that she may have cast under coercion, such that no-one else, including a possible coercer, can see whether or not the voter has subsequently updated her vote.

VoteAgain. *VoteAgain* is an e-voting system that aims for coercion-resistance via deniable vote updating. It was proposed by Lueks, Querejeta-Azurmendi, and Troncoso [25] (Usenix Security 2020). They designed VoteAgain to improve usability compared to previous approaches by relieving voters to store cryptographic state (e.g., secret signing keys).[1] Lueks et al. implemented a prototype of VoteAgain to evaluate its practicality: their benchmarks demonstrate that VoteAgain is very efficient, even for large-scale elections.

Importantly, Lueks et al. formally analyzed the security of VoteAgain in terms of coercion-resistance as well as *ballot privacy*, which guarantees that the protocol does not leak more information on each single voter's choice than what can be derived from the final election result, and *verifiability*, which guarantees that it can be verified whether the election result corresponds to the voters' choices. In a nutshell, they stated that VoteAgain provides

- ballot privacy if the *trustees*, the voting authorities under whose joint public key voters encrypt their votes, are trusted,
- verifiability if the *polling authority*, the party which provides voters with anonymous voting tokens, is trusted, and
- coercion-resistance if the *tally server*, the voting authority which hides the voters' re-voting pattern, and the polling authority are trusted.

These trust assumptions are summarized in Fig. 1. They specify those threat scenarios for which VoteAgain was claimed secure originally [25].

Our contributions. In this work, we revisit VoteAgain from a security perspective. We will show that VoteAgain falls short of the security it aimed to achieve originally:

(1) We demonstrate that the polling authority needs to be trusted for all security properties (see Fig. 2), not only for verifiability and coercion-resistance, as claimed originally (see Fig. 1). We show that this issue immediately relates to the core of VoteAgain.
(2) We show that the trustees, unlike claimed originally [25], and thus all voting authorities, need to be trusted in VoteAgain for coercion-resistance (see Fig. 2).
(3) We explain that a malicious public bulletin board, unlike claimed originally [25], can break privacy, verifiability, and coercion-resistance (see Fig. 2). While trust on the public bulletin board can be mitigated by means independent of the VoteAgain protocol, our findings are yet another example that the importance of the public bulletin board for secure e-voting must not be underestimated.

The most critical of these observations is the first one, i.e., that the polling authority in VoteAgain needs to be trusted for all security properties. In fact, if the overall security of a voting protocol reduces to a single voting authority being uncorrupted, then one might as well replace all voting authorities by the completely trusted one without affecting security. This means that VoteAgain, in its original state, is insecure.

Now, the obvious question is: Can this problem be fixed or is it intrinsic to VoteAgain's approach? We will see that, as long as the polling authority needs to be fully trusted for verifiability, it also needs to be trusted for privacy. Hence, the only possibility to mitigate this issue is to distribute trust among other authorities. To this end, we propose a modification of VoteAgain, in which the power of the polling authority is strictly limited and in which therefore significantly less trust in this authority is required both for verifiability and privacy. In Fig. 2, we have marked those trust assumptions which were able to mitigate. Importantly, our modifications preserve usability and efficiency of VoteAgain.

Altogether, our findings bring the state of science one step closer to the goal of scalable coercion-resistant e-voting being secure under reasonable trust assumptions.

Outline of the paper. In Sec. 2, we give an overview of the VoteAgain protocol and the idea of the pitfalls we discovered. In Sec. 3, we describe VoteAgain with full technical details. In Sec. 4, we present the pitfalls of VoteAgain, several attacks to exploit them, and the inaccuracies in the formal analysis of the original paper. In Sec. 5, we show how to effectively mitigate the main issue we discovered. We conclude in Sec. 6.

Note. The first two authors published a preliminary version of this paper as a technical report on eprint [16].

2 OVERVIEW

In this section, we describe the concept of VoteAgain and the pitfalls of its approach. In Sec. 2.1, we briefly recall the security properties that VoteAgain aimed to achieve. In Sec. 2.2, we explain the idea

[1] As to the best of our knowledge, to date, usability of VoteAgain has not yet been studied.

of VoteAgain; a more detailed protocol description is presented in Sec. 3. In Sec. 2.3, we explain the pitfalls of VoteAgain's approach that we discovered, with full technical details on our attacks in Sec. 4.

2.1 Security Properties

We recall the security notions of ballot privacy, end-to-end verifiability, and coercion-resistance.

Ballot privacy. For most elections, it is important that outside or even inside observers (e.g., voting authorities) should not be able to tell how individual voters voted. This property is called *(ballot) privacy* [4]. It guarantees that the data published during the election (including, for example, voters' ballots, talliers' proofs of integrity, etc.) does not leak more information on the voters' plain choices than what can be derived from the final election result.

Verifiability. Numerous e-voting systems suffer from flaws which open up the opportunity for inside or outside attackers to change the election result without being detected (see, e.g., [15, 30, 31]). Therefore, modern secure e-voting systems strive for what is called *(end-to-end) verifiability* [8]. This fundamental security property enables voters or external auditors to verify whether the published election result is correct, i.e., corresponds to the votes cast by the voters, even if, for example, voting devices and servers have programming errors or are outright malicious.

Coercion-resistance. A voting protocol is *coercion-resistant* [23, 24] if any coerced voter, instead of obeying the coercer, can run some counter-strategy such that (i) by running the counter-strategy, the coerced voter achieves her goal (e.g., successfully votes for her favorite candidate), and (ii) the coercer is not able to distinguish whether the coerced voter followed his instructions or tried to achieve her own goal. There exist several concepts in the literature to construct coercion-resistant e-voting systems. The approach taken in VoteAgain is called *deniable vote updating*: if a voter is coerced to vote for a certain candidate, then the voter's counter-strategy is to update her vote after the coercer has left. In this way, she can overwrite her previously cast choice. At the same time, due to some technical mechanisms in the background, it is guaranteed that the coercer is not able to distinguish whether or not the voter has subsequently updated her vote.

2.2 VoteAgain

We now recall how VoteAgain works. In this section, we present VoteAgain in such a way that the idea of the pitfalls below can be followed. We provide more technical details of VoteAgain in Sec. 3.

Idea. As mentioned above, VoteAgain follows the concept of coercion-resistance via *deniable vote updating*: each voter can overwrite her previously submitted ballots, that she may have cast under coercion, such that no-one else, including a possible coercer, can see whether or not the voter has updated her vote.

VoteAgain implements this idea as follows. In addition to the standard voting authorities which are commonly used in modern secure e-voting systems, namely a *public bulletin board (PBB)* and a *trustee (T)*, VoteAgain employs two further parties, the *polling authority (PA)* and the *tally server (TS)*. The role of the PA is to guarantee that each voter can cast ballots without revealing her identity to the public bulletin board. At the same time, the PA ensures that it can be verified whether incoming ballots were submitted by eligible voters only. The role of the TS is to hide the voters' re-voting/vote-updating pattern. In combination, these two voting authorities are supposed to securely guarantee deniable vote updating: on the one hand, every observer can verify that only eligible voters have cast the ballots on the bulletin board and that each eligible voter did not overwrite any other voter's ballot (under the assumption the PA is honest), while, on the other hand, it remains secret to any outsider whether or not a given voter has updated her ballot.

In what follows, we describe VoteAgain's approach more precisely, with full technical details provided in Sec. 3.

Participants. VoteAgain is run among the following participants:

- *Voters* $\mathcal{V}_1, \ldots, \mathcal{V}_n$: Each voter $\mathcal{V}_i$ interacts with the polling authority and the public bulletin board to cast her ballots. It is assumed that each voter can authenticate herself to the polling authority. The voters encrypt their choices under the trustee's public key.
- *Polling Authority (PA)*: The PA provides each eligible voter with a one-time voting token. The voter can then use this token to sign her ballot without revealing her identity to the public bulletin board.
- *Public Bulletin Board (PBB)*: The PBB is an append-only list which contains all public information, including the voters' cast ballots, as well as proofs and results published by the election/tallying authorities during the tally phase.
- *Tally Server (TS)*: The TS adds dummy ballots (encryptions of 0 under the trustee's public key), shuffles all ballots, groups them by voter, and selects the last ballot for each voter.
- *Trustee (T)*:[2] The trustee shuffles and decrypts the last ballots per voter that were selected by TS.

Protocol phases. VoteAgain proceeds in three phases (the invoked procedures are defined in Sec. 3):

(1) *Pre-election phase*: The election authorities set up their public/private key material. The polling authority PA initializes the voters' anonymous identifiers (Procedure 1).
(2) *Election phase*: Voters authenticate to PA every time they vote to obtain a one-time voting token (Procedure 2). They then use the token to cast a ballot (Procedure 3). The public bulletin board PBB verifies the correctness and eligibility of each incoming ballot (Procedure 4). Note that voters can re-vote multiple times.
(3) *Tally phase*: The tally server TS adds dummy ballots to hide the voters' re-voting pattern, makes real and dummy ballots publicly indistinguishable, selects the last ballot for each real and dummy voter, and removes the dummies again (Procedure 5). The trustee T shuffles and decrypts the ballots previously returned by TS (Procedure 6).

The verification program of VoteAgain follows immediately from the protocol description: essentially, the proofs published by the different parties on PBB are checked. We refer to [25] (Procedures 7 and 8) for details.

[2]The role of the trustee T is distributed in the original VoteAgain protocol [25]. For the sake of brevity, we assume that the trustee is a single entity.

2.3 Pitfalls

We show that VoteAgain [25] is not secure under those trust assumptions for which it was claimed to be (Fig. 1). We describe our findings on an intuitive level in what follows, with full technical details presented in Sec. 4. Our results are summarized in Fig. 2.

Impact of corrupted PA. Recall that the polling authority PA provides each eligible voter with one-time voting tokens. In order to guarantee deniable vote updating (and thus coercion-resistance), the links between the individual voters and their ballots signed with the voting tokens remain hidden. Since a malicious PA could tamper with the distribution of the voting tokens undetectably, the PA needs to be trusted for verifiability. This was already stated in the original VoteAgain paper but its implication to ballot privacy was apparently overlooked (Fig. 1). In secure e-voting, there exists a strong relationship between verifiability and ballot privacy: if ballots can be dropped or replaced undetectably, then privacy of the remaining ballots is undermined [10].

In Sec. 4, we show how this general threat applies to VoteAgain in its worst form. Unlike in most other e-voting protocols (e.g., Helios) where only a *small* number of ballots can be *dropped* without being detected, a malicious PA in VoteAgain can *replace* an *arbitrary* number of ballots *completely* secretly. This vulnerability results into two risks which, in combination, can have a devastating effect: (1) By replacing many ballots except for just a few, privacy of the untouched ballots can be broken with extremely high probability. (To see this, assume that all but one ballot are replaced.) Since, in political elections, typically the final result of each district is published, such an attack can be executed in each district separately so that, in total, ballot privacy of many voters may be put at risk. (2) By replacing ballots with reasonably distributed choices, the tracks of the privacy attack can easily be covered.

The consequence of this observation is disillusioning: there exists a voting authority in VoteAgain (namely the PA) which needs to be trusted for *all* security properties: ballot privacy, verifiability, and coercion-resistance. This means that VoteAgain is as (in)secure as a trivial voting protocol with a single and completely trusted voting authority which is responsible for the whole voting process. Fortunately, as we will demonstrate in Sec. 5, we were able to effectively mitigate trust on the PA both in terms of verifiability and privacy, without affecting VoteAgain's usability and efficiency.

Impact of corrupted trustee. The role of the tally server TS is to hide the voters' re-voting pattern by adding indistinguishable dummy ballots which are later removed. Clearly, the PA and the TS need to be trusted for coercion-resistance because both of them know whether a voter re-voted. This was already stated in the original VoteAgain paper (Fig. 1). We discovered that, additionally, there exists a subtle yet significant relationship between coercion-resistance and a possibly corrupted trustee. Assume that a coercer forces a voter to submit a sequence of ballots in which each ballot encrypts a vote for a randomly chosen candidate. By this, the coerced voter's sequence of plain votes is essentially unique and thus a "fingerprint". Now, if the trustee is corrupted, then the coerced voter's submitted ballots can be linked to the voter, even if some dummy ballots are added to the voter's ciphertexts. In other words, the secrecy of the vote updating process is undermined, even if the PA and the TS are trusted. As a result, the trustee needs to be trusted for coercion-resistance as well (Fig. 2). Therefore, all voting authorities need to be trusted for coercion-resistance which is an assumption arguably too strong as well.

Impact of corrupted PBB. We discovered that the importance of the public bulletin board PBB was underestimated originally. Lueks et al. [25] stated that even if the PBB is malicious, VoteAgain provides ballot privacy, verifiability, and coercion resistance (Fig. 1). We discovered that, in fact, the PBB needs to be trusted for all security properties (Fig. 2), as explained next.

If the PBB is malicious, then it can show a "faked" view on the bulletin board to a voter which includes this voter's submitted ballot. At the same time, the PBB does not append the ballot to the "real" bulletin board which it shows to the remaining parties [18]. In this way, the voter's ballot is effectively dropped even though the voter verified that her ballot is on "the" bulletin board. This demonstrates that the PBB needs to be trusted for verifiability.

By executing the above attack against verifiability for several voters, the choices of the remaining ballots are hidden behind less further choices, which undermines ballot privacy. We note, however, that the effect of this privacy attack (which applies to virtually all e-voting protocols) is much weaker than the privacy attack of a malicious PA described above because the number of ballots that can be dropped undetectably is more limited for the following two reasons. Firstly, if at least one of the affected voter cross-checks her view on PBB, the attack is detected. Secondly, in order to obtain significant information on the remaining votes, many ballots have to be dropped but this is then reflected in the low number of votes of the final result which would raise suspicion. However, at least formally, it follows that the PBB needs to be trusted for ballot privacy as well, which disproves the original security claim.

Furthermore, a malicious PBB can break coercion-resistance, even if voters submit their ballots anonymously, as described next. The coercer forces a voter to submit a ballot for a certain candidate and to reveal all secret information on the submitted ballot. The PBB identifies and accepts this ballot but drops all subsequently incoming ballots (similarly to the attack on verifiability above). By this, the coerced voter can no longer update her choice, even if the coercer is absent for the rest of the submission phase.

The PBB is a critical bottleneck in all e-voting systems, not only VoteAgain. There are several approaches to mitigate trust on the PBB which could also be used in VoteAgain (see, e.g., [11, 18, 19]). Nevertheless, our findings are yet another example to demonstrate that the importance of the PBB for secure e-voting must not be underestimated.

3 VOTEAGAIN PROTOCOL

In this section, we precisely describe the VoteAgain protocol. The original VoteAgain protocol [25] employs specific cryptographic primitives centered around ElGamal public-key encryption [13]. We chose to abstract away from this concrete instantiation because our attacks on VoteAgain (Sec. 4) exploit the protocol design but no specific cryptographic details. We also think that our slightly more abstract presentation simplifies comprehension of the complex protocol.

Cryptographic primitives. VoteAgain employs the following cryptographic primitives:

- An IND-CPA-secure public-key encryption (PKE) scheme $\mathcal{E} = (\mathsf{EncKeyGen}, \mathsf{Enc}, \mathsf{Dec})$ which is homomorphic.
- A NIZKP of correct encryption $(\mathsf{Prove}^{\mathsf{Enc}}, \mathsf{Verify}^{\mathsf{Enc}})$ for the PKE scheme $\mathcal{E}$ and a voting relation R which specifies valid choices.
- A NIZKP of correct decryption $(\mathsf{Prove}^{\mathsf{Dec}}, \mathsf{Verify}^{\mathsf{Dec}})$ for the PKE scheme $\mathcal{E}$.
- A shuffle algorithm Shuffle [3] which takes as input a vector of ciphertexts C (w.r.t. $\mathcal{E}$), re-encrypts each entry of C, permutes the vector uniformly at random, and returns the resulting shuffled ciphertext vector C' together with a proof π^{Shuffle} that C' is correct.
- An EUF-CMA-secure signature scheme $(\mathsf{SigKeyGen}, \mathsf{Sign}, \mathsf{Verify})$.

Procedure 1 (Setup). The election authorities generate their public/private key pairs and send the public keys to the bulletin board. The polling authority PA runs $(\mathsf{pk}_{\mathsf{PA}}, \mathsf{sk}_{\mathsf{PA}}) \leftarrow \mathsf{SigKeyGen}(1^\ell)$ and sends $\mathsf{pk}_{\mathsf{PA}}$ to PBB. The tally server TS runs $(\mathsf{pk}_{\mathsf{TS}}, \mathsf{sk}_{\mathsf{TS}}) \leftarrow \mathsf{EncKeyGen}(1^\ell)$ and sends $\mathsf{pk}_{\mathsf{TS}}$ to PBB. The trustee T runs $(\mathsf{pk}_{\mathsf{T}}, \mathsf{sk}_{\mathsf{T}}) \leftarrow \mathsf{EncKeyGen}(1^\ell)$ and sends pk_{T} to PBB.

For each voter $\mathcal{V}_i$, the polling authority PA generates a pair (vid_i, m_i) uniformly at random where vid_i is the voter's (secret) identifier, and m_i is (the initial state of) the voter's ballot counter. More precisely, PA runs the following program for each $\mathcal{V}_i$:

(1) $vid_i \xleftarrow{r} \mathcal{M}_{\mathsf{Enc}}$[3]
(2) $m_i \xleftarrow{r} \{2^{\ell-2}, \ldots, 2^{\ell-1} - 1\} \subset \mathcal{M}_{\mathsf{Enc}}$
(3) store $(\mathcal{V}_i, vid_i, m_i)$ internally

Procedure 2 (GetToken). Each time a voter $\mathcal{V}_i$ wants to submit a ballot, the voter needs to authenticate herself to the polling authority PA. If authentication of $\mathcal{V}_i$ is successful, then PA sends certain one-time credentials to $\mathcal{V}_i$ which enable $\mathcal{V}_i$ to cast a single ballot without having to reveal her identity. For this purpose, PA essentially encrypts $\mathcal{V}_i$'s identifier vid_i as well as her ballot counter m_i under the public key $\mathsf{pk}_{\mathsf{TS}}$ of the tally server TS.

By this, on the one hand, it is not possible to coerce $\mathcal{V}_i$ into revealing vid_i or m_i, while, on the other hand, it can be verified that $\mathcal{V}_i$'s ballot was submitted by an eligible voter and that only $\mathcal{V}_i$'s last ballot is counted (if PA is trusted).

More precisely, PA executes the following steps after voter $\mathcal{V}_i$ authenticated herself correctly:

(1) $\gamma \leftarrow \mathsf{Enc}(\mathsf{pk}_{\mathsf{TS}}, vid_i)$
(2) $I \leftarrow \mathsf{Enc}(\mathsf{pk}_{\mathsf{TS}}, m_i)$
(3) $m_i \leftarrow m_i + 1$
(4) $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{SigKeyGen}(1^\ell)$
(5) $\sigma^\tau \leftarrow \mathsf{Sign}(\mathsf{sk}_{\mathsf{PA}}, \mathsf{pk} \| \gamma \| I)$
(6) send $\tau \leftarrow (\mathsf{pk}, \mathsf{sk}, \gamma, I, \sigma^\tau)$ to $\mathcal{V}_i$

The voter can then check whether $\mathsf{Verify}(\mathsf{pk}_{\mathsf{PA}}, \sigma^\tau, (\mathsf{pk} \| \gamma \| I)) = \top$ holds true. If this is the case, then $\mathcal{V}_i$ can use τ to cast a vote, as described next.

Procedure 3 (Vote). Voter $\mathcal{V}_i$ takes as input $\tau = (\mathsf{pk}, \mathsf{sk}, \gamma, I, \sigma^\tau)$ from PA (see GetToken) as well as a candidate $c \in C$ and executes the following steps to cast her ballot β:

(1) $(v, aux) \leftarrow \mathsf{Enc}(\mathsf{pk}_{\mathsf{T}}, c)$
(2) $\pi^{\mathsf{Enc}} \leftarrow \mathsf{Prove}^{\mathsf{Enc}}((\mathsf{pk}_T, v), (aux, c))$
(3) $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, v \| \pi^{\mathsf{Enc}} \| \mathsf{pk} \| \gamma \| I \| \sigma^\tau)$
(4) $\beta \leftarrow (v, \pi^{\mathsf{Enc}}, \mathsf{pk}, \gamma, I, \sigma^\tau, \sigma)$
(5) send β to PBB

For each incoming ballot β, PBB checks whether $\mathsf{Valid}(\beta) = \top$ holds true (see below for algorithm Valid), and if this is the case, then PBB appends β. Voter $\mathcal{V}_i$ can then verify whether β was appended to PBB.

Procedure 4 (Valid). For each incoming ballot β, PBB verifies whether β contains a valid choice, whether eligibility was acknowledged by the polling authority PA, and whether it does not contain a duplicate entry of a ballot β' that was already appended.

More precisely, Valid returns $\top$ if and only if the following conditions are satisfied:

(1) $\mathsf{Verify}^{\mathsf{Enc}}(\mathsf{pk}_{\mathsf{T}}, v, \pi^{\mathsf{Enc}}) = \top$, and
(2) $\mathsf{Verify}(\mathsf{pk}, \sigma, v \| \pi^{\mathsf{Enc}} \| \mathsf{pk} \| \gamma \| I \| \sigma^\tau) = \top$, and
(3) $\mathsf{Verify}(\mathsf{pk}_{\mathsf{PA}}, \sigma^\tau, (\mathsf{pk}^{\mathsf{Enc}} \| \gamma \| I)) = \top$, and
(4) $(v, \ldots) \notin \beta'$ and $(\ldots, \mathsf{pk}, \ldots) \notin \beta'$ for some appended β'

Procedure 5 (Filter). The tally server TS reads the list of ballots $B \leftarrow (\beta_i)_{i=1}^{n_B}$ from PBB and verifies for each $\beta \in B$ whether $\mathsf{Valid}(\beta) = \top$ holds true. If this is not the case, then TS aborts. Otherwise, TS continues as follows.

Adding dummies. For each ballot $\beta_i = (v_i, \ldots, \gamma_i, I_i, \ldots) \in B$, the tally server executes the following steps:

(1) $\theta_R \leftarrow \mathsf{Enc}(\mathsf{pk}_{\mathsf{TS}}, 1; 0)$
(2) $\beta'_i \leftarrow (v_i, \gamma_i, I_i, \theta_R)$

In other words, TS creates a "stripped" ballot β'_i which consists of the respective voter's encrypted candidate v_i, the voter's encrypted identifier γ_i, the encrypted ballot counter I_i, as well as a deterministic ciphertext θ_R to "tag" real ballots.

Additionally, and this is one of the main ideas of VoteAgain, the tally server TS generates a number of *dummy ballots* which are used to hide the re-voting pattern of real voters. To this end, the tally server TS creates n_D further ballots $\beta'_i = (v_\epsilon, \gamma_i, I_i, \theta_D)$, where

- $v_\epsilon \leftarrow \mathsf{Enc}(\mathsf{pk}_{\mathsf{T}}, 0; 0)$
- $\theta_D \leftarrow \mathsf{Enc}(\mathsf{pk}_{\mathsf{TS}}, g; 0)$ for some $g \neq 1$

holds true.

This means that each dummy ballot contains a 0-vote (with trivial randomness 0), as well as a deterministic ciphertext θ_D to "tag" dummy ballots. The ciphertext γ_i either encrypts the identifier of a real voter $\mathcal{V}_i$ in which case the encrypted ballot counter I_i of the dummy ballot is smaller than the one of the last ballot cast by $\mathcal{V}_i$, or the ciphertexts γ_i encrypts the identifier of a "fake" voter.[4]

The tally server TS sends the resulting list of ballots $B' \leftarrow (\beta'_i)_{i=1}^{n_B+n_D}$ to PBB.

Shuffling. The tally server TS verifiably shuffles the ciphertext vector B':

(1) $(B'', \pi_\sigma) \leftarrow \mathsf{Shuffle}(B')$
(2) send (B'', π_σ) to PBB.

[3] $\mathcal{M}_{\mathsf{Enc}}$ denotes the message space of the PKE scheme $\mathcal{E}$ (for public key $\mathsf{pk}_{\mathsf{TS}}$).

[4] We refer to [25] (Sec. 5.2) for a detailed description of how dummy ballots are constructed precisely because the vulnerabilities of VoteAgain presented in this paper are independent of the specific cover.

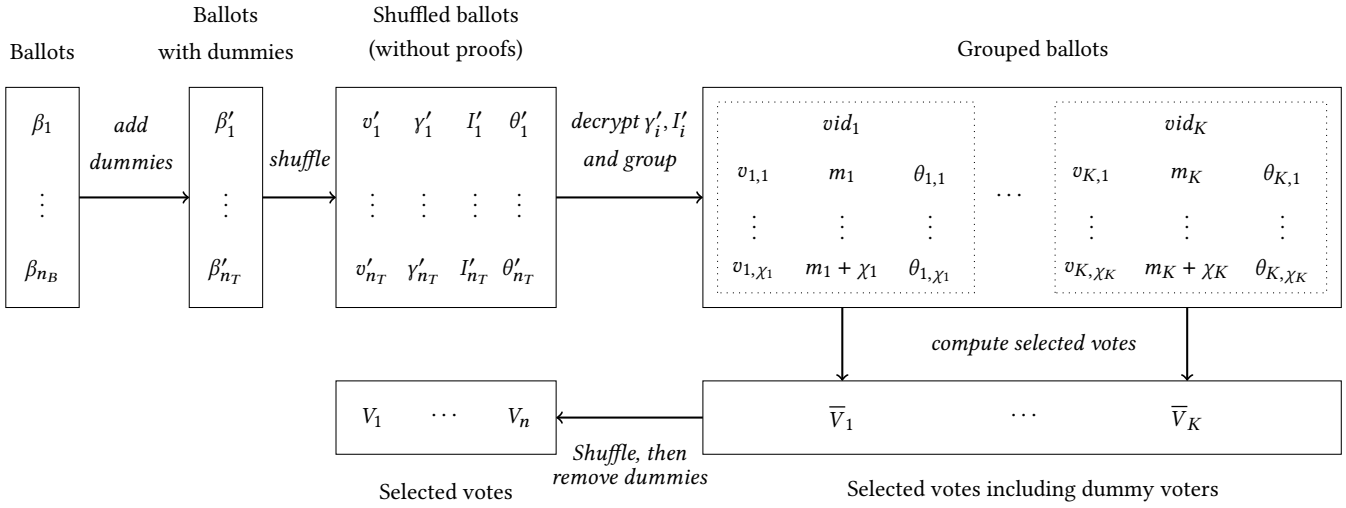

Figure 3: Overview of Procedure 5 (Filter), adopted from [25]. Notation: n_B denotes the number of ballots, n_D the number of dummies added by TS, $n_T = n_B + n_D$ denotes their sum, K the number of (real) voters plus the number of dummy voters, and χ_i the number of ballots for voter i (including dummy voters).

Grouping. For each $\beta'' = (v''_i, \gamma''_i, I''_i, \theta''_i) \in B''$, the tally server TS uses its secret key $\mathsf{sk}_{\mathsf{TS}}$ to verifiably decrypt the encrypted identifiers and ballot counters:

(1) for all $i \le n_B + n_D$:
  (a) $\overline{vid}_i \leftarrow \mathsf{Dec}(\mathsf{sk}_{\mathsf{TS}}, \gamma''_i)$
  (b) $\pi^{\mathsf{Dec}}_{i,1} \leftarrow \mathsf{Prove}^{\mathsf{Dec}}((\mathsf{pk}_{\mathsf{TS}}, \gamma''_i), \mathsf{sk}_{\mathsf{TS}})$
  (c) $\overline{m}_i \leftarrow \mathsf{Dec}(\mathsf{sk}_{\mathsf{TS}}, I''_i)$
  (d) $\pi^{\mathsf{Dec}}_{i,2} \leftarrow \mathsf{Prove}^{\mathsf{Dec}}((\mathsf{pk}_{\mathsf{TS}}, I''_i), \mathsf{sk}_{\mathsf{TS}})$
  (e) $\pi^{\mathsf{Dec}}_i \leftarrow (\pi^{\mathsf{Dec}}_{i,1}, \pi^{\mathsf{Dec}}_{i,2})$
(2) $C \leftarrow (\overline{vid}_i, \overline{m}_i, \pi^{\mathsf{Dec}}_i)_{i=1}^{n_B+n_D}$
(3) send C to PBB

If there exist $i \neq j$ such that $(\overline{vid}_i, \overline{m}_i) = (\overline{vid}_j, \overline{m}_j)$ holds true, then TS aborts. Otherwise, TS groups all $(\overline{vid}_i, v_i, \overline{m}_i, \theta_\star)$ according to the identifiers $\overline{vid}$. We denote the resulting groups by $(G_j)_{j=1}^K$.

Selecting. For each group G_j, the tally server TS opts for the ciphertext $v_{j\star}$ assigned to the highest ballot counter m in G_j. If the respective tag θ refers to a real voter, then TS re-encrypts the ciphertext $v_{j\star}$, and otherwise, the TS replaces the ciphertext by a 0-vote:

(1) $j\star \leftarrow$ index of maximal $\overline{m}$ in G_j
(2) if $\mathsf{Dec}(\mathsf{sk}_{\mathsf{TS}}, \theta_{j\star}) = 0$, then $\overline{V}_j \leftarrow v_{j\star} \cdot \mathsf{Enc}(\mathsf{pk}_{\mathsf{T}}, 0)$
(3) else $\overline{V}_j \leftarrow \mathsf{Enc}(\mathsf{pk}_{\mathsf{T}}, 0)$
(4) $\pi^{\mathsf{Sel}}_j \leftarrow$ NIZKP of correctness of previous steps[5]
(5) $F \leftarrow (\overline{vid}_j, \overline{V}_j, \pi^{\mathsf{Sel}}_j)_{j \le K}$
(6) send F to PBB

Removing dummies. The tally server TS verifiably shuffles the vector of selected encrypted votes $\mathcal{S}_D \leftarrow (\overline{V}_j)_{j \le K}$, i.e., TS computes $(\mathcal{S}'_D, \pi'_\sigma) \leftarrow \mathsf{Shuffle}(\mathcal{S}_D)$ and sends $(\mathcal{S}'_D, \pi'_\sigma)$ to PBB.

[5]We refer to [25] for the precise relation to be proven by this NIZKP. The details are not relevant for our purposes.

Next, TS creates the list of indices $\mathcal{D}$ of encrypted dummy votes $\overline{V}'_j$ in the shuffled ciphertext vector $\mathcal{S}'_D$, and for each $j \in \mathcal{D}$, the tally server proves that $\overline{V}'_j$ is an encryption of 0:

(1) $\mathcal{D} \leftarrow$ indices of dummy votes in $\mathcal{S}'_D$
(2) $R \leftarrow (r_j)_{j \in \mathcal{D}}$ such that $\overline{V}'_j = \mathsf{Enc}(\mathsf{pk}_{\mathsf{T}}, 0; r_j)$[6]
(3) send $(\mathcal{D}, R)$ to PBB

Finally, the tally server TS publishes the list of votes to be decrypted by the trustee T, i.e., TS computes $\mathcal{S} \leftarrow (\mathcal{S}'_D \setminus \mathcal{S}'_{\mathcal{D}})$ and sends $\mathcal{S}$ to PBB.

Procedure 6 (Tally). In order to obtain the final election result, the trustee T verifiably shuffles $\mathcal{S}$ and then uses its secret key sk_{T} to verifiably decrypt the resulting shuffled ciphertexts.

4 PITFALLS OF VOTEAGAIN

We elaborate on the pitfalls of VoteAgain's approach which we sketched in Sec. 2.3.

4.1 Impact of corrupted PA

It was claimed in the original VoteAgain paper [25] that VoteAgain provides ballot privacy if the trustee is honest, while PA, TS, and PBB can be malicious (Fig. 1). We will now show that a malicious PA can break privacy. The idea is that the PA can impersonate any voter by simulating Procedure 2 and Procedure 3.

Attack: Let $\mathcal{V}_1, \ldots, \mathcal{V}_n$ be the voters. Assume that the PA is malicious and wants to know how $\mathcal{V}_1, \ldots, \mathcal{V}_l$ voted. After all voters $\mathcal{V}_{l+1}, \ldots, \mathcal{V}_n$ have submitted their (last) ballots, the PA runs the voting process $n - (l+1)$ times. In each of these processes $i \in \{l+1, \ldots, n\}$, the malicious PA runs Procedure 2 for voter $\mathcal{V}_i$, uses $\mathcal{V}_i$'s token to run Procedure 3 to submit a ballot for some arbitrary candidate c_i, and stores (i, c_i).

[6]Each r_j is a simple combination of the randomness that TS used to create $\overline{V}_k$ and of the randomness that TS then used to re-encrypt $\overline{V}_k$ to obtain $\overline{V}'_j$.

Impact: By design, it is not possible to verify whether $\mathcal{V}_i$, $i \in \{l+1, \ldots, n\}$, has updated her vote. Therefore, the final election result consists of $\mathcal{V}_1, \ldots, \mathcal{V}_l$'s votes plus $n - (l+1)$ votes (i, c_i) submitted by PA. The adversary can now substract all $(i, c_i)_{i \geq l+1}$ from the public election result to obtain the (aggregated) choices of $\mathcal{V}_1, \ldots, \mathcal{V}_l$.

Remarks: We have described the attack in its most general form, i.e., for some arbitrary l. In order to obtain much information about the votes of $\mathcal{V}_1, \ldots, \mathcal{V}_l$ from the aggregation of their votes, it is necessary to restrict l. Clearly, for $l = 1$, ballot privacy of $\mathcal{V}_1$ is completely broken, but even for $l > 1$, significant information can be leaked, e.g., if the adversary wants to know whether all voters $\mathcal{V}_1, \ldots, \mathcal{V}_l$ voted for the same candidate.

Recall from Sec. 2.3 that in political elections, typically the final result of each district is published, so that the above attack can be executed in each district separately and thus privacy of many voters in total may be put at risk.

Observe that, in order to not raise any suspicion, the adversary can easily choose the replacing choices c_i such that the manipulated final election result appears completely reasonable.

We note how this attack is a particular instance of the work presented at CCS18 [10] by Cortier *et al.*. In that paper, it is noted that if voters do not have individual verifiability, their ballots can be dropped undetectably, reducing the privacy of voters whose ballots have not been dropped. In VoteAgain, if the PA is not trusted, then it can impersonate *any* voter, resulting in a complete lost of individual verifiability and, by consequence, privacy of any set of voters of the adversary's choice.

4.2 Impact of corrupted trustee

It was claimed in the original VoteAgain paper [25] that VoteAgain provides coercion-resistance if the PA and TS are honest, while the PBB (if voters submit anonymously) and the trustees can be malicious (Fig. 1). We now describe how an honest-but-curious trustee T can break coercion-resistance of VoteAgain. The idea is that for each voter, trustee T is able to decrypt the individual sequences of ciphertexts assigned to this voter's (anonymous) voter id $\overline{vid}$.

Attack: The adversary chooses a sequence $(c_j)_{j=1}^{l}$ over the set of candidates C uniformly at random. The coercer instructs a targeted voter $\mathcal{V}$ to submit a ballot β_j (i.e., run Procedures 2 and 3) for each element c_j of this sequence (preserving the order of the sequence) and then a ballot β for the adversary's favorite candidate c.

Impact: Since the trustee is corrupted, the adversary is able use sk_T to decrypt all $v_{i,\star}$ for each $\overline{vid}$ in the grouped ballots $(G_j)_{j=1}^{K}$ (after the "grouping" phase in Procedure 5). The coercer (removes all dummy votes injected by TA and) verifies whether there exists a group G_j which contains the chosen sequence of candidates. If this is the case, the adversary knows that the voter obeyed (with overwhelming probability in l).

4.3 Impact of corrupted PBB

We show that a malicious PBB can break verifiability, ballot privacy, and coercion-resistance of VoteAgain.

Verifiability. It was claimed in the original VoteAgain paper [25] that VoteAgain provides verifiability if the PA is honest, while the TS, PBB, and the trustee can be malicious (Fig. 1). We will now describe an attack of a malicious PBB which breaks verifiability of VoteAgain. Note that the effect of this attack can be increased by repeating it multiple times for different voters.

Attack: Let $\mathcal{V}$ be an arbitrary voter. In Procedure 3, the PBB shows a "faked" view on the bulletin board to voter $\mathcal{V}$ which includes $\mathcal{V}$'s ballot β. However, PBB does not append β to the "real" bulletin board which it shows to the remaining parties.

Impact: At the end of Procedure 3, the voter verifies successfully that β was appended to the "faked" bulletin board. However, $\mathcal{V}$'s choice is not included in the input to Procedure 5 and therefore not in the final election result. Because this manipulation remains undetected, verifiability is broken.

Ballot Privacy. It was claimed in the original VoteAgain paper [25] that VoteAgain provides ballot privacy if the trustee is honest, while PA, TS, and PBB can be malicious (Fig. 1). We will now show that a malicious PBB can break ballot privacy.

Attack: Let $\mathcal{V}_1, \ldots, \mathcal{V}_n$ be the voters. Assume that the PA is malicious and wants to know how $\mathcal{V}_1, \ldots, \mathcal{V}_l$ voted. For each voter $\mathcal{V}_i$, $i \in \{l+1, \ldots, n\}$, the PBB executes the verifiability attack described above.

Impact: The final election result consists only of $\mathcal{V}_1, \ldots, \mathcal{V}_l$'s aggregated votes.

Remark: As already noted in Sec. 2.3 this vulnerability (which applies to virtually all e-voting protocols) is of rather theoretical concern in contrast to the privacy attack of a malicious PA described above which can be devastating in real practical elections.

Coercion-Resistance. It was claimed in the original VoteAgain paper [25] that VoteAgain provides coercion-resistance if the PA and TS are honest, while the PBB (if voters submit anonymously) and the trustee can be malicious (Fig. 1). We will now show that a malicious PBB can break coercion-resistance even if all voters submit their ballots anonymously.

Attack: The coercer chooses a candidate $c \in C$ and instructs an arbitrary voter $\mathcal{V}$ to submit a ballot for this candidate (i.e., run Procedure 2 and Procedure 3). Furthermore, the coercer tells the voter to reveal the submitted ballot β, including all secret information from Procedure 3. The malicious PBB identifies the incoming (no longer anonymous) ballot β, append it, and drop all subsequently incoming ballots by any voter (see PBB's attack on verifiability).

Impact: The affected voter can no longer "overwrite" the coercer's choice c even if the coercer is absent for the rest of submission phase.

4.4 Inaccuracies in the formal analysis

In this section we expose the points where the formal analysis in VoteAgain is inaccurate, mainly the game description of the ballot privacy experiment (Figure 7) and the proof of coercion resistance (Appendix B).

The ballot privacy game definition. There are two implicit assumptions in the ballot privacy model: the $O\mathtt{voteLR}()$ and the $O\mathtt{cast}()$ oracles. One the one hand, Lueks *et al.* restrict $O\mathtt{voteLR}()$ to only receive as input one token and two different candidates. However, the adversary controls the PA, and therefore the authority that generates the tokens (and therefore, the voter identifiers). In order to model an adversary that generates the voter identifiers, the latter should be allowed to choose different tokens for each

run of the protocol. In particular, the oracle should take as input $(\tau_0, c_0, \tau_1, c_1)$, and depending on the game bit b, use one or the other. It is now trivial for the adversary to win the game by the attack described in Section 4.1. Once the grouping of the filtering phase is published in the PBB, the adversary can inspect whether one identifier or the other appears.

On the other hand, there is an implicit assumption in $O\texttt{voteLR}()$ and $O\texttt{cast}()$, that if a ballot is valid, then the PBB will append it in the unique view of the bulletin board. However, this implicitly assumes that the PBB also provides the "must append" property, and is therefore required to be honest.

Implicit assumption on Coercion Resistance proof. The mistake on the attack of coercion resistance by the trustees comes in the proof, rather than in the modeling. In Game 6 of the proof of Theorem 2, the encryptions of the votes published after a call to oracle $O\texttt{voteLR}()$ are replaced by encryptions of zero. The claim on indistinguishability between Game 5 and Game 6 is based on the Non-Malleability (NM) property of ElGamal (extended by a proof of plaintext knowledge). However, the trustees are not required to be honest, and therefore the adversary controls the decryption key, making it trivial to distinguish both games. The same inaccuracy is presented between Games 8 and 9.

5 MITIGATION OF TRUST IN VOTEAGAIN

In this section, we describe how to effectively mitigate the main issue of VoteAgain's security, namely that the PA in VoteAgain needs to be trusted for all security properties (Sec. 4.1). Importantly, our modifications preserve VoteAgain's efficiency and usability.

We note that trust on the trustees for coercion-resistance (Sec. 4.2) can be mitigated by specifying that the TS, which is trusted for coercion-resistance anyway, holds a share of the trustee's secret key. Recall that mitigating trust on the PBB (Sec. 4.3) is a problem that needs to be addressed independently of the actual e-voting protocol.

Idea. Recall that in the original VoteAgain protocol, the PA issues anonymous voting tokens for the voters, which is the root of VoteAgain's main security issue. We demonstrate how to assign less responsibilities to the PA in a reasonable manner, which will result into significantly less trust on PA both in terms of privacy and verifiability. More precisely, we will replace the PA with a new authority, called Counting Authority (CA), which *only* tracks ballot counters. Moreover, we will specify that voters not only authenticate to the CA when they want to submit a ballot, but also to the PBB; this assumption is common in e-voting, for example in Belenios [9]. Importantly, it will be easy to see that our modifications preserve usability and efficiency of VoteAgain.

Details. Our modifications are (essentially) restricted to the pre-election phase and the ballot submission phase of VoteAgain.

In the *pre-election phase*, we need to provide a publicly verifiable mechanism to check that each eligible voter was assigned a *unique* voting identifier. To this end, we specify that the TS creates a public mapping between voters and unique encrypted voter identifiers. More precisely, during the pre-election phase, the TS generates n_B random and unique identifiers for all real voters and all potential dummy voters. The TS publishes all identifiers, as well as a publicly verifiable encryption of each identifier on the public bulletin board. Then, the TS shuffles the encrypted identifiers and publishes the shuffled encrypted identifiers together with a ZKP of correct shuffle on the public bulletin board. Next, it publicly assigns the n eligible voters uniquely to the first n encrypted identifiers by linking the first n shuffled encrypted identifiers to each of the eligible voters' public identifier (such as their National ID number). The TS will later use the remaining identifiers for dummy voters. Every public observer can verify that all encrypted identifiers are unique by first checking that the plaintext identifiers are unique, and then verifying the encryption and shuffle. Moreover, every voter will be able to use their corresponding voter identifier to generate a voting request (and not need to trust the PA as in VoteAgain). Finally, the CA generates a random counter for each voter and stores it privately.

In the *ballot submission phase*, to cast a ballot, a voter first retrieves her encrypted identifier from the PBB, re-randomizes it and creates ZKP of correct randomization. Then, the voter authenticates to the CA and sends the re-randomized encrypted identifier and the ZKP to the CA. If the user authenticates correctly, and the re-randomized encrypted identifier matches the authenticated user, the CA looks up the voter's ballot counter, encrypts it, and provides it to the user. The CA also sends a signed token binding the encrypted identifier to the encrypted counter. Finally, the voter authenticates to the PBB, and submits a tuple consisting of the re-randomized encrypted identifier and the corresponding ZKP, the encrypted counter, CA's signature, and the encrypted vote. The PBB checks that the re-randomized encrypted counter matches the authenticated voter, checks CA's signature, and only then accepts and publishes the ballot.

The tally phase remains essentially as in VoteAgain. The only difference is that the TS, instead of using fresh identifiers for the dummy votes, uses the remaining $n_B - n$ shuffled and encrypted identifiers generated during the pre-election phase.

Effect. We argue why our modifications mitigate trust on the PA, which is now called CA, both in terms of verifiability and privacy.

Recall that in the original VoteAgain protocol, a corrupted PA could completely determine the full election result. Now, due to our modifications, the impact of a corrupted CA is more limited. In fact, a malicious CA is only able to tamper with the individual order of ballots submitted by the voters. For example, if a voter first submits a vote for candidate A and then changes her mind and submits a vote for B, then a malicious CA could manipulate that voter's counter so that her update is not effective, i.e., that her vote for A is counted. In any case, however, CA can neither replace a voter's vote by an arbitrary other vote, nor can it drop a voter's vote. In particular, for all voters who vote only once (which is probably the vast majority in real elections), integrity is fully guaranteed.

Recall that the attack against privacy by a corrupted PA (Sec. 4.1) is only possible because a PA in VoteAgain is able to impersonate all voters. In our improved version, this is no longer possible because voters also authenticate to the PBB.

We note that, even though we described an effective way to prevent known attacks by a corrupted PA, our high-level reasoning does not guarantee that no other attacks are possible. To this end, a formal security analysis in a reasonable security framework is required. We leave this challenge as interesting future work.

6 CONCLUSION

We discovered several security pitfalls of VoteAgain's approach. In particular, we observed that VoteAgain is no more secure than a trivial voting protocol with a single completely trusted voting authority. In order to solve this issue, we proposed a variant of VoteAgain which effectively mitigates trust on the voting authorities, without affecting VoteAgain's efficiency and usability. It remains interesting future work to formally prove that our modifications are sufficient to improve VoteAgain's security. Moreover, independently of the insights presented in this paper, it is worthwhile to study usability of VoteAgain's approach. Altogether, our findings bring the state of science one step closer to the goal of scalable coercion-resistant e-voting being secure under reasonable trust assumptions.

ACKNOWLEDGMENTS

Thomas Haines is the recipient of an Australian Research Council Australian Discovery Early Career Award (project number DE220100595). Johannes Müller was supported by the Luxembourg National Research Fund (FNR), under the CORE Junior project FP2 (C20/IS/14698166/FP2 /Mueller). We thank Wouter Lueks for valuable discussions on an earlier version of this paper.

REFERENCES

[1] Roberto Araújo, Amira Barki, Solenn Brunet, and Jacques Traoré. 2016. Remote Electronic Voting Can Be Efficient, Verifiable and Coercion-Resistant. In *Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Revised Selected Papers (Lecture Notes in Computer Science)*, Vol. 9604. Springer, 224–232. https://doi.org/10.1007/978-3-662-53357-4_15

[2] Michael Backes, Martin Gagné, and Malte Skoruppa. 2013. Using mobile device communication to strengthen e-Voting protocols. In *Proceedings of the 12th annual ACM Workshop on Privacy in the Electronic Society, WPES 2013*. ACM, 237–242. https://doi.org/10.1145/2517840.2517863

[3] Stephanie Bayer and Jens Groth. 2012. Efficient Zero-Knowledge Argument for Correctness of a Shuffle. In *Advances in Cryptology - EUROCRYPT 2012 (Lecture Notes in Computer Science)*, Vol. 7237. Springer, 263–280.

[4] David Bernhard, Véronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. 2015. SoK: A Comprehensive Analysis of Game-Based Ballot Privacy Definitions. In *2015 IEEE Symposium on Security and Privacy, SP 2015*. 499–516.

[5] Pyrros Chaidos, Véronique Cortier, Georg Fuchsbauer, and David Galindo. 2016. BeleniosRF: A Non-interactive Receipt-Free Electronic Voting Scheme. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 1614–1625.

[6] Jeremy Clark and Urs Hengartner. 2011. Selections: Internet Voting with Over-the-Shoulder Coercion-Resistance. In *Financial Cryptography and Data Security - 15th International Conference, FC 2011, Revised Selected Papers (Lecture Notes in Computer Science)*, Vol. 7035. Springer, 47–61. https://doi.org/10.1007/978-3-642-27576-0_4

[7] Michael R. Clarkson, Stephen Chong, and Andrew C. Myers. 2008. Civitas: Toward a Secure Voting System. In *2008 IEEE Symposium on Security and Privacy (S&P 2008)*. IEEE Computer Society, 354–368. https://doi.org/10.1109/SP.2008.32

[8] Véronique Cortier, David Galindo, Ralf Küsters, Johannes Müller, and Tomasz Truderung. 2016. SoK: Verifiability Notions for E-Voting Protocols. In *IEEE Symposium on Security and Privacy, SP 2016*. 779–798.

[9] Véronique Cortier, Pierrick Gaudry, and Stéphane Glondu. 2019. Belenios: A Simple Private and Verifiable Electronic Voting System. In *Foundations of Security, Protocols, and Equational Reasoning - Essays Dedicated to Catherine A. Meadows (Lecture Notes in Computer Science)*, Vol. 11565. Springer, 214–238. https://doi.org/10.1007/978-3-030-19052-1_14

[10] Véronique Cortier and Joseph Lallemand. [n. d.]. Voting: You Can't Have Privacy without Individual Verifiability. In *ACM Conference on Computer and Communications Security 2018*. ACM, 53–66.

[11] Chris Culnane and Steve A. Schneider. 2014. A Peered Bulletin Board for Robust Use in Verifiable Voting Systems. In *IEEE 27th Computer Security Foundations Symposium, CSF, 2014*. 169–183.

[12] Aleksander Essex, Jeremy Clark, and Urs Hengartner. 2012. Cobra: Toward Concurrent Ballot Authorization for Internet Voting. In *2012 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections, EVT/WOTE '12, 2012*. USENIX Association. https://www.usenix.org/conference/evtwote12/workshop-program/presentation/essex

[13] Taher El Gamal. 1984. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *Advances in Cryptology, Proceedings of CRYPTO '84*. 10–18.

[14] Gurchetan S. Grewal, Mark Dermot Ryan, Sergiu Bursuc, and Peter Y. A. Ryan. 2013. Caveat Coercitor: Coercion-Evidence in Electronic Voting. In *2013 IEEE Symposium on Security and Privacy, SP 2013*. IEEE Computer Society, 367–381. https://doi.org/10.1109/SP.2013.32

[15] Thomas Haines, Sarah Jamie Lewis, Olivier Pereira, and Vanessa Teague. 2020. How Not to Prove Your Election Outcome. In *2020 IEEE Symposium on Security and Privacy, SP 2020*. IEEE, 644–660. https://doi.org/10.1109/SP40000.2020.00048

[16] Thomas Haines and Johannes Müller. 2020. How not to VoteAgain: Pitfalls of Scalable Coercion-Resistant E-Voting. *IACR Cryptol. ePrint Arch.* (2020), 1406. https://eprint.iacr.org/2020/1406

[17] Sven Heiberg, Tarvi Martens, Priit Vinkel, and Jan Willemson. 2016. Improving the Verifiability of the Estonian Internet Voting Scheme. In *Electronic Voting - First International Joint Conference, E-Vote-ID 2016, Proceedings (Lecture Notes in Computer Science)*, Vol. 10141. Springer, 92–107. https://doi.org/10.1007/978-3-319-52240-1_6

[18] Lucca Hirschi, Lara Schmid, and David A. Basin. 2021. Fixing the Achilles Heel of E-Voting: The Bulletin Board. In *34th IEEE Computer Security Foundations Symposium, CSF 2021*. IEEE, 1–17. https://doi.org/10.1109/CSF51468.2021.00016

[19] Aggelos Kiayias, Annabell Kuldmaa, Helger Lipmaa, Janno Siim, and Thomas Zacharias. 2018. On the Security Properties of e-Voting Bulletin Boards. In *Security and Cryptography for Networks - 11th International Conference, SCN 2018, Proceedings*. 505–523.

[20] Aggelos Kiayias, Thomas Zacharias, and Bingsheng Zhang. 2015. End-to-End Verifiable Elections in the Standard Model. In *Advances in Cryptology - EUROCRYPT 2015, Proceedings, Part II (Lecture Notes in Computer Science)*, Vol. 9057. Springer, 468–498. https://doi.org/10.1007/978-3-662-46803-6_16

[21] Oksana Kulyk and Stephan Neumann. 2020. Human Factors in Coercion Resistant Internet Voting - A Review of Existing Solutions and Open Challenges. In *Electronic Voting - 5th International Joint Conference, E-Vote-ID 2020*.

[22] Oksana Kulyk, Vanessa Teague, and Melanie Volkamer. 2015. Extending Helios Towards Private Eligibility Verifiability. In *E-Voting and Identity - 5th International Conference, VoteID 2015, Proceedings (Lecture Notes in Computer Science)*, Vol. 9269. Springer, 57–73. https://doi.org/10.1007/978-3-319-22270-7_4

[23] Ralf Küsters and Johannes Müller. 2017. Cryptographic Security Analysis of E-voting Systems: Achievements, Misconceptions, and Limitations. In *Electronic Voting - Second International Joint Conference, E-Vote-ID 2017, Proceedings (Lecture Notes in Computer Science)*, Vol. 10615. Springer, 21–41. https://doi.org/10.1007/978-3-319-68687-5_2

[24] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. 2011. Verifiability, Privacy, and Coercion-Resistance: New Insights from a Case Study. In *32nd IEEE Symposium on Security and Privacy, S&P 2011*. 538–553.

[25] Wouter Lueks, Iñigo Querejeta-Azurmendi, and Carmela Troncoso. 2020. VoteAgain: A Scalable Coercion-Resistant Voting System. In *29th USENIX Security Symposium, USENIX Security 2020*. USENIX Association, 1553–1570. https://www.usenix.org/conference/usenixsecurity20/presentation/lueks

[26] André Silva Neto, Matheus Leite, Roberto Araújo, Marcelle Pereira Mota, Nelson Cruz Sampaio Neto, and Jacques Traoré. 2018. Usability Considerations For Coercion-Resistant Election Systems. In *Proceedings of the 17th Brazilian Symposium on Human Factors in Computing Systems, IHC 2018*. ACM, 40:1–40:10. https://doi.org/10.1145/3274192.3274232

[27] UN Committee on Human Rights. 1996. General Comment 25 of the Human Rights Committee. (1996). https://www.ohchr.org/en/elections

[28] Kim Ramchen, Chris Culnane, Olivier Pereira, and Vanessa Teague. 2019. Universally Verifiable MPC and IRV Ballot Counting. In *Financial Cryptography and Data Security - 23rd International Conference, FC 2019, Revised Selected Papers (Lecture Notes in Computer Science)*, Vol. 11598. Springer, 301–319. https://doi.org/10.1007/978-3-030-32101-7_19

[29] Peter Y. A. Ryan, Peter B. Rønne, and Vincenzo Iovino. 2016. Selene: Voting with Transparent Verifiability and Coercion-Mitigation. In *Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Revised Selected Papers (Lecture Notes in Computer Science)*, Vol. 9604. Springer, 176–192. https://doi.org/10.1007/978-3-662-53357-4_12

[30] Michael A. Specter and J. Alex Halderman. 2021. Security Analysis of the Democracy Live Online Voting System. In *30th USENIX Security Symposium, USENIX Security 2021*. USENIX Association.

[31] Michael A. Specter, James Koppel, and Daniel J. Weitzner. 2020. The Ballot is Busted Before the Blockchain: A Security Analysis of Voatz, the First Internet Voting Application Used in U.S. Federal Elections. In *29th USENIX Security Symposium, USENIX Security 2020*. USENIX Association, 1535–1553. https://www.usenix.org/conference/usenixsecurity20/presentation/specter

[32] Roland Wen and Richard Buckland. 2009. Masked Ballot Voting for Receipt-Free Online Elections. In *E-Voting and Identity, Second International Conference, VoteID 2009, Proceedings (Lecture Notes in Computer Science)*, Vol. 5767. Springer, 18–36.