USENIX Journal of Election Technology and Systems (JETS)
Volume 3, Number 2 • August 2015
www.usenix.org/jets/issues/0302

Improved Coercion-Resistant Electronic Elections through Deniable Re-Voting

Dirk Achenbach, Karlsruhe Institute of Technology
Carmen Kempka, NTT Secure Platform Laboratories
Bernhard Löwe, Karlsruhe Institute of Technology
Jörn Müller-Quade, Karlsruhe Institute of Technology

In a democracy, it is essential that voters cast their votes independently and freely, without any improper influence. Particularly, mechanisms must be put into place that prevent—or at least severely impede—the coercion of voters. One possible countermeasure to coercion is *revoting*: after casting a vote under coercion, the voter can re-cast and overwrite her choice. However, revoting is only meaningful as a strategy to evade coercion if the adversary cannot infer whether the voter has modified her choice—revoting needs to be *deniable*, while still being publicly verifiable. We define the notions of correctness, verifiability, and deniability for a tallying protocol which allows for revoting. We also present a protocol realizing these notions. To the best of our knowledge, our solution is the first to achieve both deniability and public verifiability without asking information about the voter's previously-cast ballots for revoting. A seemingly competitive line of work, started by the well-known work of Juels, Catalano, and Jakobsson, uses fake credentials as a strategy to evade coercion: the voter presents to the adversary a fake secret for voting. In this work, we extend Juels et al.'s work to achieve deniable revoting. Their solution also allows for revoting, however not deniably. Our solution supports fake credentials as an opt-in property, providing the advantages of both worlds.

1. INTRODUCTION

Free elections are the backbone of a democracy. In traditional elections, voters fill out their ballot in a voting booth to ensure the privacy of the vote. Because traditional presence elections are tedious, there is a significant interest in carrying out elections over the Internet. It seems impossible to give the same privacy guarantee that a voting booth does in an online setting. Further, advances in consumer electronics even call the privacy of the voting booth into question. To sell their vote, voters can easily record their choice with their smartphone [Post 2010; Benaloh 2013].

To mitigate these problems, several solutions have been developed to make elections resistant against such attacks. *Fake credentials* [Juels et al. 2005] and *revoting* [Post 2010; Volkamer and Grimm 2006] are two of these. While *fake credentials* successfully disable the adversary from obtaining a voter's credential by coercion, they are contradictory to a meaningful response whether the ballot has been cast successfully. *Revoting*, on the other hand, allows for a meaningful response.

The idea behind revoting is that, if a voter is allowed to overwrite her vote arbitrarily many times, she effectively cannot be coerced. We stress, however, that this proposed revoting must be perfectly deniable—no party, including all servers, must be able to tell whether a coerced voter has evaded coercion by revoting.

As a side-effect, deniable revoting is applicable to elections which allow the voter to change her mind after casting a legit vote. An example for this is delegated voting [Green-Armytage 2014; Miller 1969], in which the voter can delegate her vote to a so-called proxy, who then votes in her stead. She can later overwrite this delegation with a new delegation or by casting a vote by herself.

We investigate deniable revoting as a strategy for evading coercion. We build upon the well-known work of Juels, Catalano, and Jakobsson [Juels et al. 2010]. Specifically, we adapt their definitions of correctness and coercion-resistance to include revoting and its deniability. While their construction enables voters to re-cast their vote, their security model does not account for revoting as a strategy to evade coercion. Indeed, their construction allows the adversary to observe whether a ballot is superseded. We present a tallying protocol which allows voters to deniably re-cast their votes, thus making revoting a viable strategy to evade coercion. The macrostructure of our protocol is the same as that of Juels et al., hence it is compatible with fake voter credentials. To facilitate a simpler exposition, we do not investigate the possibility of offering both evasion strategies simultaneously.

Existing protocols that allow revoting either do not achieve both deniability and public verifiability at the same time, or require the voter to remember some information about previous votes [Kutyłowski and Zagórski 2007]. To the best of our knowledge, we are the first to propose a publicly verifiable method for enabling deniable revoting without requiring the voter to save state between votes.

There is one caveat to the revoting approach. If a coercer observes the voter's behavior until after the polls close, the voter cannot re-cast her vote and thus revoting does not work as a strategy for evading coercion. This caveat applies to the strategy in general and thus cannot be remedied by any protocol. For this reason, our adversarial model does not allow the adversary to observe the voter's behavior indefinitely—he must give up his control before the polls close. On the other hand, a coerced voter does not employ any evasion strategy during adversarial observation—she submits to the adversary's instructions completely.

1.1. Our Contribution

We investigate revoting as a strategy for evading coercion. To this end, we extend the well-established notions of correctness, verifiability, and coercion-resistance by Juels, Catalano, and Jakobsson [Juels et al. 2010] to take deniable revoting into account. Juels et al.'s approach also allows voters to re-cast their vote, but not as a strategy for evading coercion. In fact, an adversary can learn whether a specific ballot cast under coercion is superseded by a more recent one. Our work improves upon the method of Juels et al. of eliminating duplicates such that no adversary can tell how often (and when) a particular voter casts her vote. We present a method for counting votes which ensures the deniability of revotes, and prove the correctness of our protocol as well as the deniability of revoting. This work seeks to serve as a proof of concept and does not claim to provide an efficient solution.

1.1.1. Juels, Catalano, and Jakobsson's Approach. Let us briefly recap the approach of Juels, Catalano, and Jakobsson. To cast a vote, voters post their ballot on a public bulletin board, together with proofs of knowledge of their credential and the validity of their choice. After the voting phase, the tallying authority computes the tally in five steps:

(1) Check proofs: The tallying authority checks all proofs associated with the ballots and discards ballots with invalid proofs.
(2) Remove duplicate ballots: At most one ballot per credential is kept. To identify duplicate ballots, *plaintext equivalency test*s (PETs) are employed on the encrypted voter credentials. For a pair of ballots with identical voter credentials, a pre-determined policy decides which ballot is kept.
(3) Shuffle: The remaining ballots are shuffled.
(4) Check credentials: The tallying authority discards ballots with invalid voter credentials.
(5) Tally: All remaining valid ballots are decrypted and counted.

This solution uses fake credentials as a strategy to evade coercion. Revoting is supported in the sense that double votes are handled. However, voters cannot plausibly deny having revoted—this is not the aim of the construction. Imagine an adversary forcing a voter to cast a certain ballot in his presence. Since the ballots are only shuffled (Step (3)) after duplicates have been removed (Step (2)), the adversary can easily monitor if his observed ballot is deleted. This way, he can deduce that the coerced voter has later cast a ballot with the same credential. This does not impose a problem in Juels et al.'s approach—the observed ballot was cast with a fake credential, and does not need to be superseded to evade coercion. To employ revoting as a strategy for evading coercion however, the sorting process must ensure its deniability. A first step is to shuffle the list of ballots before removing duplicates. This conceals the chronological order of ballots, however. For two ballots cast with the same credential, it cannot be determined anymore which one was cast more recently. We devised a method of an "encrypted labeling" which allows voters to privately re-cast their ballot, while preserving enough information about the chronological order of ballots.

1.2. Achieving Deniable Revoting

In this section we sketch the challenges of deniable revoting, and how to overcome them, in more detail. For revoting to be suitable as a strategy for evading coercion, the voter must be able to *deniably* do so. Intuitively, we say a voter can deniably re-cast her vote if no adversary can, by his observations, tell whether she did. At the same time, we still require the tally to be computed correctly. In particular, at any time during the voting period, for each participating voter who has already cast a vote, there

is exactly one *valid* ballot (i.e. one ballot which counts as long as it is not superseded), so a re-cast ballot must invalidate this one ballot with matching voter credentials, and become the new valid ballot corresponding to this credential. These requirements impose several privacy challenges on the tallying process.

More concretely, the deniability requirement of the revoting process implies that the adversary must not be able to figure out how often a certain voter has re-cast her ballot. Further, he must not even infer which of the voters did revote at all. Also, if he can tell whether any particular ballot is part of the final tally, revoting is not deniable. Nevertheless, to maintain the universal verifiability of the tally, the correctness of each tallying step must be transparent. To give an intuition for the subtle challenges of authenticating re-castable ballots, we describe two exemplary attacks.

For the first example recap the attack described in the former section. There, the adversary is forcing the voter to cast a ballot in his presence. He can then observe the ballot appearing on the bulleting board right after. If this particular ballot is deleted or marked as invalid in the tallying process, he can deduce that his coercion attempt was not successful. As mentioned above, we employ encrypted labelling and re-encryption mixes after the casting phase to solve this problem.

While Step (2) in Juels et al.'s construction (see Section 1.1.1) does hide the identity of voters with duplicate ballots, it leaks information about the number of removed duplicates per credential. Consider the following attack, which is similar to the "1009 attack" described by Warren Smith [Smith 2005]: The adversary forces the voter to vote, in his presence, a number of times no other voter is likely to vote, e.g. exactly 1009 times. During the tallying phase he then verifies that the tallying authority indeed identifies 1009 duplicate ballots for some credential. This becomes fatal in the absence of fake credentials[1]: the adversary can be sure that the coerced voter did not revote after casting her vote in the adversary's presence. Consequently, a tally which supports deniable revoting must even hide the number of revotes *any* voter has cast. We achieve this by anonymously checking ballots two-by-two for matching voter credentials, to avoid grouping ballots with the same credential.

As we detailed above, the ballots on the bulletin board must be shuffled before discarding superseded ballots. Because the chronological order of ballots is not retained in the shuffling process, we "memorize" the information whether a ballot was superseded with an encrypted label: Before shuffling, for each ballot we calculate an encrypted value o_i which is computed in a verifiable way by comparing voter credentials $\mathrm{pk}_i/\mathrm{pk}_j$ between the current ballot b_i and each more recent ballot b_j. The ballots themselves are not shuffled until o_i is computed for all ballots. Only then we shuffle the encrypted choice together with the o_i tag.

$$\begin{array}{rccccccccc} 12 = & 81 & \cdot & 53 & \cdot & 87 & \cdot & 48 & \bmod N \\ & \downarrow & & \downarrow & & \downarrow & & \downarrow & \\ 1 = & 1 & \cdot & 1 & \cdot & 1 & \cdot & 1 & \bmod N \end{array}$$

(a) Ballot is not superseded. No factor equals 1.

$$\begin{array}{rccccccccc} 25 = & 81 & \cdot & 53 & \cdot & 1 & \cdot & 48 & \bmod N \\ & \downarrow & & \downarrow & & \downarrow & & \downarrow & \\ 139 = & 1 & \cdot & 1 & \cdot & 139 & \cdot & 1 & \bmod N \end{array}$$

(b) Ballot is superseded once. One factor equals 1.

Fig. 1: One cannot tell from a product if one of the factors was $= 1$. To exploit the homomorphic property of the encryption, we swap the encryption of an arbitrary number with an encryption of a 1 and vice versa. Now one can tell from the product whether one of the factors was $\neq 1$.

We briefly sketch our construction. To make the duplicate test private, we mark each ballot with the encrypted information o_i whether there is another ballot that supersedes it. We compute o_i by comparing each ballot with each subsequent ballot. To do so in a publicly verifiable and private manner, we run an *encrypted plaintext equivalency test* (EPET) on the credentials. An EPET receives two ciphertexts as input, and outputs an encryption of a 1 if the contained plaintexts are equal, and the encryption of an arbitrary number otherwise. We compare ballot i with each subsequent ballot

[1] As Smith described, this attack is fatal for fake credentials, too: if the adversary can infer that a heap with 1009 ballots is discarded because of an invalid credential, the adversary knows the credential presented to him was a fake one.

by comparing its voter credential $Enc(pk_i)$ to the credentials $Enc(pk_{i+1})$, $Enc(pk_{i+2})$, etc. using an EPET.

A ballot is superseded if and only if its voter credential is equal to that of *at least one* newer ballot. We seek to exploit the homomorphic property of the encryption to aggregate the comparison results without revealing the result of any single comparison. A single EPET result is the encryption of a 1 iff the voter credentials of the ballots match. Thus, our goal is to determine for a number of EPET results whether at least one of them encrypts a 1. We cannot exploit the homomorphy directly, since a factor of 1 does not change the product (see Figure 1). As a remedy, we "convert" the values: We swap an encryption of a 1 with an encryption of an arbitrary number and vice versa. Now the product o_i tells us whether at least one factor is not equal to 1—if so, $o_i \neq \mathsf{Enc}(1)$ and the ballot is superseded, otherwise $o_i = \mathsf{Enc}(1)$ and the ballot is counted. We stress that the content of o_i is hidden by the encryption. Before checking o_i to potentially remove a ballot, the ballot's connection to its appearance on the bulletin board on the time of casting is broken by shuffling and re-encryption.

Our protocol results in a list of ballots containing the newest, valid, *encrypted* choice of each voter, and security is proven up to this point. From there, ballots can be tallied in an arbitrary way, using standard techniques like decryption mixes or homomorphic tallying, depending on the form of the choice. Security in combination with our protocol is then given by the security of the tallying procedure. As a consequence, our protocol supports arbitrary election formats, including *single transferable vote* (STV) or write-in candidates. However, our security definitions do not take into account information which is leaked by the tally result itself, which is possibly published in the form of decrypted votes. Information leaks like names of write-in candidates or a preference order can impede a voter's privacy. This problem is shared by any voting scheme which publishes the tally result or decrypted ballots, and needs to be accounted for separately by the tallying procedure.

1.3. Authentication for Coercion-Resistant Voting Schemes

All voting schemes use some method to authenticate voters. Even in elections where everybody is eligible to vote, authentication is necessary to make sure that of each voter only one vote is counted. Voter authentication can—in principle—follow any of three paradigms: authentication using something you know, something you have, or something you are. There are two properties of the authentication mechanism which are of grave importance for any voting scheme, regardless of the paradigm used: First, the adversary must not be able to impersonate the voter (e.g., by learning her secret key). Second, the adversary must not be able to prevent the voter from authenticating (e.g., by stealing her authentication token). We call an authentication *inalienable* when both properties hold. They are necessary conditions for the incoercibility of the voter. After a ballot is cast however, authentication and thus its inalienability lose their relevance. In an election which offers revoting, ballots are only ultimately cast when the polls close. Hence, any voting scheme that offers revoting requires the inalienability of the authentication until the polls have closed.

Revoting Versus Fake Credentials. Fake credentials and revoting are complementing strategies. While handing out fake credentials protects the confidentiality of the voter's true credentials, re-casting ballots "undoes" an adversary's coercion attempts. If the adversary cannot learn whether the voter has re-cast her ballot, revoting is a valid strategy for evading coercion (see Section 1.2).

A voting scheme can only support fake credentials if it does not provide any feedback on whether the voter authenticated successfully. Consequently, the voter does not learn whether her ballot was cast correctly. As Juels et al.'s construction uses fake credentials as a countermeasure against coercion, it has this property. Since our construction is an extension of theirs, their strategy for producing fake keys also works for our scheme. On the other hand, to give feedback about the success of authentication, one can publish the list of all registered voters and their public keys before the voting period—eliminating fake credentials as a strategy for evading coercion.

The fake credential strategy is an effective means to make the authentication inalienable, thus helping to achieve coercion-resistance. For the strategy to work however, the adversary must not take total control over the voter's actions, even if only temporarily. On the other hand, the revoting

strategy requires an inalienable authentication, but achieves a form of coercion-resistance that is robust even against temporary-but-perfect corruption.

1.4. Organization of this Paper

After discussing related work in Section 1.5, we introduce preliminaries and our assumptions in Section 2. In Section 3, we adapt the security notions of Juels et al. [Juels et al. 2010] to model deniable revoting. We present our voting scheme and prove its security in Section 4.

1.5. Related Work

The importance, advantages and challenges of revoting were discussed by Volkamer and Grimm [Volkamer and Grimm 2006], as well as by Post [Post 2010], though no solution for deniable revoting was given. Several internet voting schemes allow for revoting, not necessarily only as a defense against coercion. Neither of the voting schemes known to us uses a revoting strategy which is both deniable and publicly verifiable, while not requiring the voter to save state between votes. For example, in Kutyłowski and Zagórski's scheme [Kutyłowski and Zagórski 2007], in order to revoke her vote, the voter must reproduce the random challenge from the casting protocol. They further do not provide a formal proof of coercion-resistance for their scheme. In a similar vein, Spycher et al. [Spycher et al. 2010] propose a hybrid voting system where the voter can overrule her online vote in person. To this end, she must produce a re-encryption of the ballot formerly posted.

The election of the university president of the Université Catholique de Louvain in 2008 was conducted using an adapted version of Helios [Adida et al. 2009]. Because of the election's low risk of coercion attempts, revoting was supported mostly for convenience. Newer ballots of a voter substituted her older ballots on the bulletin board, so revoting was not deniable. The voting scheme designed for the 2011 local government elections of Norway [Gjøsteen 2010] supports non-deniable revoting. Verification is done by auditors, who can see the number of votes of the same voter. The Riigikogu Election Act demands the possibility of revoting for the Estonian election [Riigikogu 2002], but to the best of our knowledge, no deniable solution is offered [Maaten 2004].

As described in more detail above, the work of Juels et al. [Juels et al. 2005] supports revoting in principle, but since their work concentrates on fake voting credentials as a strategy to evade coercion, no deniable revoting strategy is proposed. Fake voting credentials are an important alternative approach for achieving coercion-resistance. After the work of Juels et al. [Juels et al. 2005] and its implementation Civitas [Myers et al. 2008], two password-based voting schemes, Selections [Clark and Hengartner 2011a; 2011b] and Cobra [Essex et al. 2012], were published. They use panic passwords as fake credentials, which can easily be created by a human. Selections is proven secure in an adapted version of the model introduced by Juels et al. [Juels et al. 2010]. Verifiable revoting is possible in both Selections and Cobra, but the number of votes cast with the same credential, or the fact that a certain ballot has been overwritten, is not hidden. Efficiency improvements of Juels et al.'s voting scheme [Juels et al. 2010] were proposed by Arajo et al. [Araújo et al. 2010], using another form of fake credential, and by Smith [Smith 2005], who introduced a faster removal procedure of duplicate ballots. Revoting is supported, but not deniable. The fake credential approach overcomes the problem of being coerced shortly before the end of the voting phase. However, the voter gets no sound confirmation upon vote casting. Especially human memory based systems like panic passwords pose the problem of accidentally using a fake credential without noticing.

By introducing the caveat coercitor [Grewal et al. 2013], Grewal et al. relaxed the requirement of coercion-resistance to coercion-evidence: a voter can be coerced, but the coercion is detectable. Their work addresses the problem of silent coercion, where the voter loses her credential to the adversary without noticing. With their approach, changing one's mind and overwriting a legit ballot is not possible (it would be recognized as a coercion attempt). Our work, in contrast, does not investigate the problem of silent coercion.

Various definitions of coercion-resistance and verifiability of e-voting schemes exist in the literature, see for example [Moran and Naor 2006; Delaune et al. 2009; Küsters et al. 2012; Unruh and Müller-Quade 2010; Canetti and Gennaro 1996] for an incomplete list. Several of them are

also applicable to our voting scheme. However, because our work aims to extend the work of Juels et al. [Juels et al. 2010], we stay close to their model for better comparability.

2. MODEL AND ASSUMPTIONS

2.1. Preliminaries and Notation

We give an overview of the notation we use in the definitions and in our protocol.

Our protocol is divided into five phases: A setup phase, a registration phase, a publication phase, the voting phase, and finally a tallying phase. We describe the phases in detail in Section 4.3. The tallying authority gets a key pair $(PK_{\mathcal{T}}, SK_{\mathcal{T}})$ during the setup phase. The secret key $SK_{\mathcal{T}}$ is shared among the servers which make up the tallying authority. In the registration phase a registrar $\mathcal{R}$ creates a public key secret key pair (pk_i, sk_i) for each voter.

Let b_i denote a ballot which is published on a bulletin board $\mathcal{BB}$, where $i \in \{1,\ldots,n\}$, let $L[i]$ denote the ith entry of a list L, and let ts_i denote the point in time when b_i has been published. The ballot represents the voter's choice $\beta_i \in \mathcal{C} = \{c_1,\ldots,c_{n_C}\}$. We describe the ballot in detail in Section 4.3. Overall n_V voters participate in the election. Let n_A denote the number of voters which are completely controlled by the adversary $\mathcal{A}$. Considering the adversary tries to coerce exactly one voter, there are $n_U = n_V - n_A - 1$ voters which add noise to the tally $\mathbf{X}$. The noise (choices of these voters) are defined by the distribution D_{n_U,n_C} which we describe in Section 3.

We use different functions of building blocks like encryption $\mathsf{Enc}(x)$, decryption $\mathsf{Dec}(c)$, verifiable shuffles $\mathsf{Shuffle}(\mathsf{L})$, (encrypted) plaintext equivalence tests (E)PET, and more. We describe them in Section 4.1 and Section 4.2.

In the experiments $\mathbf{Exp}^{corr}_{ES,\mathcal{A}}$, $\mathbf{Exp}^{ver}_{ES,\mathcal{A}}$, $\mathbf{Exp}^{revoting\text{-}c\text{-}resist}_{ES,\mathcal{A},H}$, $\mathbf{Exp}^{revoting\text{-}c\text{-}resist\text{-}ideal}_{ES,\mathcal{A},H}$ we use security parameters k_1 for the registration, k_2 for the voting function, and k_3 for the tallying process.

2.2. Attack Model

The adversary may corrupt a minority of the servers forming the tallying authority. He may also corrupt a number of voters and act in their place. Then, the adversary selects one (uncorrupted) voter he tries to coerce. In contrast to the model of Juels et al., we allow the coercion to be perfect—the coerced voter does exactly as told by the adversary. The adversary does not learn the voter's secret key, however. (See Section 2.3 for details on this assumption.)

Further, we require all adversarial control of the coerced voter, including direct observation, to end before the polls close. More concretely, we enable the coerced voter to re-cast her vote in secret. Intuitively, if the adversary cannot infer whether the coerced voter has re-cast her vote after his coercion attempt, we say the scheme offers *deniable revoting*. Clearly, revoting offers no protection against adversaries who observe or control the voter until after polls have closed.

2.3. Assumptions

We rely on various assumptions for our construction.

— Voter List. We assume an accepted voter list, i.e., there is a general agreement on who is allowed to vote.
— Bulletin Board. As is common for electronic voting schemes, we assume a publicly accessible append-only bulletin board. Anyone can read and append to the bulletin board, but no party can remove data from it.
— Authentic Timestamps. To ensure the correct order of votes, we rely on correct timestamps. Without loss of generality, we further assume that timestamps are unique.
— Anonymous Channel for Voters. To cast a vote, voters post their ballot to the bulletin board during the casting phase. For the votes to remain anonymous, we assume an anonymous channel the voters have access to. As stated by Juels et al. [Juels et al. 2010], anonymous channels can be realized with mix-nets (see Section 4.1.1).
— Inalienable Secret Credentials. In our construction, with each voter we associate a secret that she uses to authenticate her vote. The secret is used as a signature key for ensuring the integrity of

the ballot. Also, the corresponding public key is contained on the ballot in encrypted form. In a practical realization of our scheme, one would have to take special precautions to guarantee that the adversary can neither learn the secret credential nor deny the voter access to it. Realizing this assumption is outside the scope of this work. However, we point out that the assumption can be argued for: The election secret could be stored on a tamper-proof device that also serves a different function (e.g. a federal identity card), such that the voter cannot reasonably be without it. Voters would have reservations against handing out such a device (see also Section 1.3).

3. SECURITY NOTIONS

To model voter behavior when one is allowed to re-cast one's vote, we define D_{n_U,n_C} to be a distribution over all vectors over all (bounded) series of all candidates, including abstentions and invalid votes. Let ϕ denote the null ballot (abstention) and λ an invalid ballot. Then D_{n_U,n_C} is a distribution over vectors $((\beta_1)_j,(\beta_2)_j,\ldots,(\beta_{n_U})_j),\beta_i \in (C \cup \{\phi,\lambda\})$. For technical reasons, the length of the vote series is bound by a constant, lest the length of the voter's choices exceeds the runtime of all machines involved. In practice, one can imagine this bound to be the number of nanoseconds between the start of the voting period and its end, for example.

Further, we define the number of uncertain votes, i.e. votes cast by voters not under adversarial control or coercion, as $n_U := n_V - n_A - 1$. Let $\leftarrow$ denote the assignment operation, and $\Leftarrow$ the append operation. For any experiment $\mathbf{Exp}^x_{\mathcal{A}}$, where $x \in \{corr, ver, revoting\text{-}c\text{-}resist, revoting\text{-}c\text{-}resist\text{-}ideal\}$, we define the adversary's success probability as $\mathbf{Succ}^x_{\mathcal{A}}(k_1,k_2,k_3,n_V,n_C) := Pr[\mathbf{Exp}^x_{\mathcal{A}}(k_1,k_2,k_3,n_V,n_C) = 1]$. All algorithms are implicitly assumed to be PPT, i.e., run in probabilistic polynomial time for some fixed polynomial. A function $f(k)$ is *negligible in parameter k* if for all positive integers c there is an l_c such that $f(k) < k^{-c}$ for all $k > l_c$.

3.1. Correctness

Our notion of correctness follows that of Juels et al. We model voters not as posting one ballot, but a series of ballots. The adversary may corrupt a number of voters and vote in their place. We call a tally correct when, regardless of the behavior of corrupted parties, the last ballot, and only the last ballot, of each voter is counted.

See Figure 2 for Experiment $\mathbf{Exp}^{corr}_{ES,\mathcal{A}}$. A voting protocol is *correct* if $\mathbf{Succ}^{corr}_{ES,\mathcal{A}}(k_1,k_2,k_3,n_V,n_C)$ is negligible in k_1,k_2,k_3 for any adversary $\mathcal{A}$.

Experiment $\mathbf{Exp}^{corr}_{ES,\mathcal{A}}(k_1, k_2, k_3, n_V, n_C)$

1 $\{(sk_i,pk_i) \leftarrow \texttt{register}(SK_{\mathcal{R}}, i, k_1)\}_{i=1}^{n_V}$;
2 $V \leftarrow \mathcal{A}(\{pk_i\}_{i_1}^{n_V}$, "choose controlled voter set");
3 $\{\beta_{i,j}\}_{i\notin V} \leftarrow \mathcal{A}$("choose votes from uncontrolled voters");
4 $\mathcal{BB} \Leftarrow \{\texttt{vote}(sk_i,PK_{\mathcal{T}},\mathcal{BB},n_C,\beta_{i,0},k_2)\}_{i\notin V}$;
5 $(\mathbf{X},P) \leftarrow \texttt{tally}(SK_{\mathcal{T}},\mathcal{BB},n_C,\{pk_i\}_{i_1}^{n_V},k_3)$;
6 $\mathcal{BB} \Leftarrow \mathcal{A}$("cast ballots", $\mathcal{BB}$)
7 $(\mathbf{X}',P') \leftarrow \texttt{tally}(SK_{\mathcal{T}},\mathcal{BB},n_C,\{pk_i\}_{i_1}^{n_V},k_3)$;
8 **if** $\texttt{verify}(PK_{\mathcal{T}},\mathcal{BB},n_C,\mathbf{X}',P') = 1$ **and** $(\{\beta_{i,0} \not\subset \langle\mathbf{X}'\rangle$ **or** $|\langle\mathbf{X}'\rangle| - |\langle\mathbf{X}\rangle| > |V|)$ **then**
output 1;
else
output 0;

Fig. 2: Experiment $\mathbf{Exp}^{corr}_{ES,\mathcal{A}}$. A tallying scheme is correct if the last, and only the last, vote of legitimate voters is counted.

3.2. Verifiability

We adopt Juels et al.'s notion of verifiability. See Figure 3 for Experiment $\mathbf{Exp}^{ver}_{ES,\mathcal{A}}$. A voting protocol is *verifiable* if $\mathbf{Succ}^{ver}_{ES,\mathcal{A}}(k_1,k_2,k_3,n_V,n_C)$ is negligible in k_1,k_2,k_3 for any adversary $\mathcal{A}$.

Experiment $\mathbf{Exp}^{ver}_{ES,\mathcal{A}}(k_1, k_2, k_3, n_V, n_C)$

$\{(sk_i, pk_i) \leftarrow \texttt{register}(SK_{\mathcal{R}}, i, k_1)\}_{i=1}^{n_V}$;
$(\mathcal{BB}, \mathbf{X}, P) \leftarrow \mathcal{A}(SK_{\mathcal{T}}, \{(sk_i, pk_i)\}_{i=1}^{n_V}$, "forge election");
$(\mathbf{X}', P') \leftarrow \texttt{tally}(SK_{\mathcal{T}}, \mathcal{BB}, n_C, \{pk_i\}_{i_1}^{n_V}, k_3)$;
if $\mathbf{X} \neq \mathbf{X}'$ **and** $\texttt{verify}(PK_{\mathcal{T}}, \mathcal{BB}, n_C, \mathbf{X}, P) = 1$ **then**
  output 1;
else
  output 0;

Fig. 3: Experiment $\mathbf{Exp}^{ver}_{ES,\mathcal{A}}$. A tallying scheme is verifiable if $\mathcal{A}$ is not able to state a wrong tally with a correct proof.

3.3. Deniability of Revoting

Similarly to how Juels, Catalano, and Jakobsson model coercion-resistance, we model the deniability of revoting as two experiments following the "real-world-ideal-world" paradigm: Even in a "flawless" voting scheme some information leaks—the total number of cast ballots, the tally, and possibly even who participated in the vote. When measuring the advantage of the adversary, we seek to factor out "unavoidable" leaks of information. To this end we define an *ideal* experiment ($\mathbf{Exp}^{revoting\text{-}c\text{-}resist\text{-}ideal}_{ES,\mathcal{A},H}$) that captures all such leaks. The *real* experiment ($\mathbf{Exp}^{revoting\text{-}c\text{-}resist}_{ES,\mathcal{A},H}$) captures the concrete voting protocol. We then examine the difference of the probability of success of an adversary in the ideal world versus an adversary in the real protocol.

In the "real world" experiment, an election is carried out as specified by the protocol. The adversary can corrupt and thus completely control a set of voters. We model this by handing the secret keys of corrupted voters to the adversary. Uncorrupted voters cast their ballots according to the distribution D_{n_U,n_C}. Further, the adversary may select one (uncorrupted) voter as his coercive target. As in the original definition by Juels et al., an extension to a simultaneous coercion of more than one voter is straightforward. The voter does not carry out an evasion strategy in our experiment, i.e. the adversary receives the coerced voter's secret key and may cast votes in the voter's stead. (Note that we actually assume the adversary *cannot* learn the coerced voter's secret key in reality (see Section 2.3). We model time-limited, perfect coercion by giving the adversary access to the secret key, and not accepting further output later.) Then, a bit b is flipped. If $b = 0$, the coerced voter evades coercion by revoting. If $b = 1$, the voter submits to the coercion, i.e., does nothing. After all votes are cast, a tally and corresponding proofs are computed and handed to the adversary. The adversary then guesses b. See Figure 4 for Experiment $\mathbf{Exp}^{revoting\text{-}c\text{-}resist}_{ES,\mathcal{A},H}$.

In the ideal experiment, we make use of a function `ideal-tally`, which represents an ideal tallying process. Its working depends on the challenge bit b. If $b = 0$ (voter evades coercion) it counts the coercive target voter's vote. Otherwise (voter submits to coercion) it counts the adversary's choice for the coercive target voter's vote. Further, the ideal adversary is not supplied with the contents of the bulletin board, but only its length l, as well as the election result $\mathbf{X}$. If the coercive target voter evades coercion, the reported length of the bulletin board is increased by the length of one ballot. (Note that the overall number of revotes can always be inferred if the number of cast ballots and the number of counted ballots is visible.) See Figure 5 for Experiment $\mathbf{Exp}^{revoting\text{-}c\text{-}resist\text{-}ideal}_{ES,\mathcal{A},H}$.

Experiment $\mathbf{Exp}^{\textit{revoting-c-resist}}_{ES,\mathcal{A},H}(k_1, k_2, k_3, n_V, n_A, n_C)$

$V \leftarrow \mathcal{A}$(voter names, "control voters");
$\{(sk_i, pk_i) \leftarrow \texttt{register}(SK_{\mathcal{R}}, i, k_1)\}_{i=1}^{n_V}$;
$(j, \beta) \leftarrow \mathcal{A}(\{sk_i\}_{i\in V}$, "set target voter and vote");
if $|V| \neq n_A$ **or** $j \notin \{1,2,\ldots,n_v\} - V$ **or** $\beta \notin \{1,2,\ldots,n_C\} \cup \emptyset$ **then**
  output 0;
$\mathcal{BB} \Leftarrow \texttt{vote}(\{sk_i\}_{i\notin V}, PK_{\mathcal{T}}, n_C, D_{n_U,n_C}, k_2)$;
$\mathcal{BB} \Leftarrow \mathcal{A}(sk_j, \mathcal{BB}$, "cast ballots");
$b \in_U \{0,1\}$;
if $b = 0$ **then**
  $\mathcal{BB} \Leftarrow \texttt{vote}(\{sk_j\}, PK_{\mathcal{T}}, n_C, D_{n_U,n_C}, k_2)$;
$(\mathbf{X}, P) \leftarrow \texttt{tally}(SK_{\mathcal{T}}, \mathcal{BB}, n_C, \{pk_i\}_{i_1}^{n_V}, k_3)$;
$b' \leftarrow \mathcal{A}(\mathbf{X}, P, \mathcal{BB}$, "guess b");
if $b' = b$ **then**
  output 1;
else
  output 0;

Fig. 4: Experiment $\mathbf{Exp}^{\textit{revoting-c-resist}}_{ES,\mathcal{A},H}$. This experiment describes the possibilities of an adversary $\mathcal{A}$ in the "real world", including the tally result.

Experiment $\mathbf{Exp}^{\textit{revoting-c-resist-ideal}}_{ES,\mathcal{A}',H}(k_1, k_2, k_3, n_V, n_A, n_C)$

$V \leftarrow \mathcal{A}'$(voter names, "control voters");
$\{(sk_i, pk_i) \leftarrow \texttt{register}(SK_{\mathcal{R}}, i, k_1)\}_{i=1}^{n_V}$;
$(j, \beta) \leftarrow \mathcal{A}'$("set target voter and vote");
if $|V| \neq n_A$ **or** $j \notin \{1,2,\ldots,n_v\} - V$ **or** $\beta \notin \{1,2,\ldots,n_C\} \cup \emptyset$ **then**
  output 0;
$b \in_U \{0,1\}$;
$\mathcal{BB} \Leftarrow \texttt{vote}(\{sk_i\}_{i\notin V}, PK_{\mathcal{T}}, n_C, D_{n_U,n_C}, k_2)$;
$\mathcal{BB} \Leftarrow \mathcal{A}'(sk_j, |\mathcal{BB}|$, "cast ballots");
$l \Leftarrow |\mathcal{BB}|$;
if $b = 0$ **then**
  $l \Leftarrow |l| + 1$;
$(\mathbf{X}, P) \leftarrow \texttt{ideal-tally}(SK_{\mathcal{T}}, \mathcal{BB}, n_C, \{pk_i\}_{i_1}^{n_V}, k_3, b)$;
$b' \leftarrow \mathcal{A}'(\mathbf{X}, l$, "guess b");
if $b' = b$ **then**
  output 1;
else
  output 0;

Fig. 5: Experiment $\mathbf{Exp}^{\textit{revoting-c-resist-ideal}}_{ES,\mathcal{A}',H}$. This experiments describes the possibilities of an adversary $\mathcal{A}'$, based on the tally result. The success of $\mathcal{A}'$ normalizes the success of $\mathcal{A}$.

We define the *advantage* of the adversary $\mathcal{A}$ as $\mathbf{Adv}^{\textit{revoting}}_{\mathcal{A}}(k_1,k_2,k_3,n_V,n_A,n_C) :=$

$$\mathbf{Succ}^{\textit{revoting-c-resist}}_{ES,\mathcal{A},H}(k_1,k_2,k_3,n_V,n_A,n_C) - \mathbf{Succ}^{\textit{revoting-c-resist-ideal}}_{ES,\mathcal{A},H}(k_1,k_2,k_3,n_V,n_A,n_C).$$

A voting scheme features *deniable revoting* if $\mathbf{Adv}^{\text{revoting}}_{\mathcal{A}}$ is negligible in k_1,k_2,k_3 for any adversary.

4. A COERCION-RESISTANT VOTING PROTOCOL WITH DENIABLE REVOTING

In this section, we first introduce several building blocks used in our protocol. We then describe our construction and prove its security according to the definitions introduced in the previous section.

4.1. Black-box Ideal Functionalities

We use several primitives as blackboxes and assume the existence of an efficient realization. Though we suggest efficient instantiations where possible, we focus on our general approach for achieving deniable revoting. We see our work as a proof of concept.

4.1.1. Verifiable Secret Shuffle. A verifiable shuffle computes a function $\mathsf{Shuffle}(\mathsf{C}) \mapsto (\mathsf{C}', \mathsf{P})$, which gets as input a list $C := [c_1, \ldots, c_n]$ of randomizable ciphertexts. Its output consists of a list $C' := [c'_{\pi(1)}, \ldots, c'_{\pi(n)}]$, where π is a random permutation and $c'_{\pi(i)}$ is a re-encryption of c_i for $i = 1, \ldots, n$, and a proof P of correctness of the shuffle.

We assume our shuffles are *secret* and *verifiable*: secrecy implies it is infeasible to link an input ciphertext c_i to its corresponding output ciphertext $c'_{\pi(i)}$ better than guessing, and verifiability requires a proof P that C is indeed a permutation (with re-encryption) of C'.

We further assume our shuffles are *distributable* among several servers, and speak of a *mix-net* if a shuffle is distributed. Examples for verifiable secret shuffles and mix-nets are [Neff 2001; Sako and Kilian 1995; Groth 2002; Khazaei et al. 2012; Abe and Hoshino 2001; Golle et al. 2004].

4.1.2. EUF-CMA Secure Digital Signatures. In our voting scheme, voters use unforgeable signatures for proving their eligibility. A signature scheme $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ is called *existentially unforgeable against adaptive chosen message attacks (EUF-CMA)* [Goldwasser et al. 1988], if $Pr[(vk, sk) \leftarrow \mathsf{KeyGen}(1_k), (\sigma, m) \leftarrow \mathcal{A}^{O_{Sign(.)}}(vk) : \mathsf{Verify}(vk, \sigma, m) = 1]$ is negligible in security parameter k, where $O_{Sign(.)}$ is a signature oracle which on input of a message m_i ouputs a valid signature $\sigma_i = \mathsf{Sign}(sk, m_i)$ with $\mathsf{Verify}(vk, \sigma_i, m_i) = 1$, and m has never been queried to $O_{Sign(.)}$. We require our signatures to allow for an efficient zero-knowledge proof of signature knowledge.

4.1.3. Non-Interactive Zero-Knowledge Proofs (of Knowledge). We make use of Non-Interactive Zero-Knowledge Proofs (NIZK) and non-Interactive Zero-Knowledge Proofs of Knowledge (NIZKPoK) in our construction. NIZK and NIZKPoK exist for arbitrary NP statements. Correct decryption can be proven by using the Chaum-Pedersen-Protocol [Chaum and Pedersen 1993] as a proof of logarithm equivalence. Logarithm knowledge can be proven with the Schorr-protocol [Schnorr 1991]. A proof of signature knowledge was introduced by Camenisch et al. [Camenisch and Stadler 1997]. Structure-preserving signatures [Abe et al. 2010] with Groth-Sahai proofs [Groth and Sahai 2008] are an alternative that does not rely on a random oracle.

4.2. Building Blocks

4.2.1. Modified Elgamal Encryption (m-Elgamal). M-Elgamal is a modification of the Elgamal encryption scheme [Elgamal 1985], used by Juels et al. [Juels et al. 2010]. Given a multiplicative cyclic group G of prime order p in which the decisional Diffie-Hellman problem is hard, and generators $g_1, g_2 \in G$, for key generation choose random $x_1, x_2 \in \mathbb{Z}_p$, and output secret key (x_1, x_2) and public key $y := g_1^{x_1} g_2^{x_2}$. To encrypt a message m with public key y, choose $r \in \mathbb{Z}_p$ at random, and output the ciphertext $c = \mathsf{Enc}(m, y) := (g_1^r, g_2^r, y^r m)$. To decrypt a ciphertext $c = (A, B, C)$ with secret key (x_1, x_2), compute $m = \mathsf{Dec}(c, (x_1, x_2)) := A^{-x_1} B^{-x_2} C$. To simplify notation, we write $\mathsf{Enc}(m)$ or $\mathsf{Dec}(c)$ if the keys are clear from context. Decryption can be distributed using the construction of Cramer et al. [Cramer et al. 1997]. Ciphertexts created with m-Elgamal are multiplicatively homomorphic and randomizable by multiplying with an encryption of 1. A proof of encryption of a certain plaintext can be achieved by publication of the encryption randomness followed by randomization. Correct decryption can be proven with the Chaum-Pedersen-protocol [Chaum and Pedersen 1993].

4.2.2. Plaintext Equality Test. Plaintext equality tests (PET) were introduced by Juels et al. [Jakobsson and Juels 2000; Juels et al. 2010]. They decide whether two ciphertexts contain the same plaintext. We denote this by a function $\mathsf{Pet}(c_1, c_2, t) \mapsto (b, \Pi)$ which gets as input two ciphertexts c_1 and c_2 and a decryption trapdoor t. Its output is a bit b with $b = 1$ if c_1 and c_2 contain the same plaintext, and $b = 0$ otherwise, as well as a proof Π of correctness. In our protocol, we perform computations on encrypted PET results. We define *encrypted plaintext equality tests (EPETs)* denoted by a function $\mathsf{Epet}(c_1, c_2) \mapsto (c, \Pi_c)$ which outputs an encryption of 1 if c_1 and c_2 contain the same plaintext, and an encryption of a random number, if the plaintexts are different, as well as a proof Π_c of correctness. EPETs can be computed without a trapdoor. We require PETs to be distributable.

(Encrypted) Plaintext Equality Test for Homomorphic Encryption. Juels et al. [Jakobsson and Juels 2000] introduced a plaintext equality test for Elgamal encryption. We generalize their result to a PET for multiplicatively homomorphic encryption which can also be used as an EPET. Instantiating it with Elgamal or m-Elgamal allows a distributed execution of the PET.

Let *Enc* denote a multiplicatively homomorphic encryption function, i.e., $Enc(m_1) \cdot Enc(m_2) = Enc(m_1 \cdot m_2)$, and *Dec* its decryption function. We can perform a PET on two ciphertexts $c_1 = Enc(m_1)$ and $c_2 = Enc(m_2)$ for plaintexts m_1 and m_2 as follows:

Algorithm $\mathsf{Epet}(\mathsf{c_1}, \mathsf{c_2})$: On input of two ciphertexts c_1 and c_2, choose r at random and compute $c_{diff} := (c_1/c_2)^r$ with a NIZKPoK Π_c of an r such that $c_{diff} = (c_1/c_2)^r$, for example by using the Chaum-Pedersen-protocol [Chaum and Pedersen 1993]. Exploiting the homomorphic property of the encryption, we have $c_{diff} = (c_1/c_2)^r = Enc((m_1/m_2)^r) = 1^r = 1$ if $m_1 = m_2$ (or $r \equiv 0 \mod Ord(G)$, see below). By outputting c_{diff} and the proof Π_c, this scheme can be used as an EPET.

Algorithm $\mathsf{Pet}(\mathsf{c_1}, \mathsf{c_2}, \mathsf{t})$: To make a PET out of the EPET, first compute $(c_{diff}, \Pi_c) := \mathsf{Epet}(c_1, c_2)$, then decrypt c_{diff} using decryption key t, with a proof Π_d of correct decryption. Output 1 if $Dec(c_{diff}) = 1$, and 0 otherwise, as well as the proof $\Pi := (\Pi_c, \Pi_d)$. If $m_1 \neq m_2$, c_{diff} is a random number because of the mask r, and can be revealed in order to prove correctness of the result. Also, if it is not clear from the form of c_{diff} that $r \neq 0 \mod \mathrm{Ord}(G)$, r can be opened if $Dec(c_{diff}) = 1$.

4.3. Our Construction

We divide elections into five phases: setup, registration, setup publication, voting, and tallying.

(1) Setup. The tellers create a joint public encryption key $PK_{\mathcal{T}}$ and a shared secret decryption key sk_T for threshold decryption for the homomorphic encryption scheme m-Elgamal introduced in Section 4.2.1. Let $\mathsf{Enc}(.)$ denote its encryption function.

(2) Registration. Each voter V_j obtains their credential for voting, i.e., a key pair (pk_j, sk_j) for an EUF-CMA secure signature scheme (see Section 4.1.2). We assume that an efficient non-interactive zero-knowledge proof of knowledge of a signature exists for this signature scheme. The public verification key obtained here also acts as identification of the voter. For each voter, an entry $(ID, \mathsf{Enc}(pk_j))$ is stored in a List L_0, where ID encodes the voter's name. The encryption $\mathsf{Enc}(pk_j)$ is computed using the teller's public key $PK_{\mathcal{T}}$.

(3) Setup publication. A candidate-slate $\mathcal{C} = (c_1, \ldots, c_l)$ is published on the public bulletin board. It contains the electable candidates $c_1, \ldots, c_l$. The list L_0 is published and serves as a list of eligible voters and their encrypted public verification keys. The public key $PK_{\mathcal{T}}$ of the tellers is published as well.

(4) Voting. Voters create ballots and submit them to the bulletin board. A ballot b_i is a three-tuple as explained below. Encrypted parts are encryptions under the public key of the tellers:

$$b_i = (\mathsf{Enc}(\beta_i), \mathsf{Enc}(pk_i), \mathrm{ts}_i),\ NIZKPoK$$

Here, $\mathsf{Enc}(\beta_i)$ is an encryption of the choice $\beta_i \in \mathcal{C}$. We assume this for the simplicity of our description. To support arbitrary choices like single transferable vote (STV), we can alternatively let the choice be a function of the candidate slate and the voter's intention. Furthermore, $\mathsf{Enc}(pk_i)$ is an encryption of the voter's public key pk_i, and ts_i is a timestamp (see Section 2.3) created right before the ballot is cast. The timestamp is not encrypted. In addition to the ballot itself,

the voter submits a non-interactive zero-knowledge proof of knowledge *NIZKPoK* to prove that the ballot has been cast with her consent. The NIZKPoK proves knowledge of a signature of $(\mathsf{Enc}(\beta_i), \mathsf{Enc}(pk_i), \mathrm{ts}_i)$ w.r.t. the public key encrypted in the ballot. (Directly attaching the signature itself would reveal how many votes a single voter has cast.) Stated more formally, the voter proves knowledge of a signature σ_i with

$$\mathtt{verify}(\sigma_i, (\mathsf{Enc}(\beta_i), \mathsf{Enc}(pk_i), \mathrm{ts}_i), pk_i') = 1 \ \wedge \ pk_i' = pk_i.$$

(5) Tallying. Before the ballots can be tallied, all superseded ballots have to be sorted out. This procedure is described in detail in the next section. After all ballots with invalid proofs of signature knowledge, all superseded ballots, and all ballots with invalid credentials have been omitted, the remaining ballots can be tallied with standard techniques, for example by a decryption mix.

4.4. Sorting Out Superseded Ballots

In this section, we describe how to mark all superseded ballots. A ballot is superseded when a more recent ballot of the same voter—that is, a ballot with a newer timestamp and matching voter credential—is posted to the bulletin board. Our method protects the privacy of the voter and is also publicly verifiable.

In four steps, the tallying authority computes a shuffled list of all valid ballots without their timestamps. They will be re-encrypted and in a random order. The resulting list will only contain the last votes cast by eligible voters. W.l.o.g. we assume all submitted choices are valid.

(1) **Encrypted Tests Whether Two Ballots Are From The Same Voter** The procedure starts off with a list of all "well-formed" ballots on the bulletin board, i.e. all ballots with a valid NIZKPoK as described above. After it has been checked, the NIZKPoK is omitted in further steps. The list is sorted according to the timestamp in ascending order. For each ballot b_i, the tallying authority tests if the encrypted public key pk_i of b_i matches the public key pk_j of any newer ballot b_j (see Section 4.4.1): for each ballot b_j ($i < j < n$), the tallying authority performs distributed EPETs to obtain $(d_{i,j}, \Pi_{i,j}) := \mathsf{Epet}(Enc(pk_i), Enc(pk_j))$. The encrypted differences $d_{i,j}$ and the proofs $\Pi_{i,j}$ of correctness of the EPET-result are published. ($d_{i,j} = \mathsf{Enc}(1)$ iff $pk_i = pk_j$.)

(2) **Marking Ballots As Superseded** The tallying authority performs a verifiable conversion on all computed differences. If $d_{i,j}$ is an encryption of 1, replace it by an encryption of a random number. Else, replace it with an encryption of 1. Differences are detached from their corresponding ballots before conversion, and later sorted back. The details of this step are described in Section 4.4.2. For each ballot, the tallying authority aggregates all converted $d'_{i,j}$ by multiplying them, exploiting the homomorphic property of the encryption: $o_i := \prod_j d'_{i,j}$

(3) **Omit Superseded Ballots** The tallying authority jointly compares o_i with $\mathsf{Enc}(1)$ for each ballot b_i. It omits all ballots b_i with $\mathsf{Epet}(\mathsf{o_i}, \mathsf{Enc}(1)) \neq 1$ from future computations. (Those are the ballots that have been superseded by a more recent one of the same voter.) The last ballot is never superseded.

(4) **Omit Ballots With Invalid Credentials** Before tallying, the tallying authority checks the voter credentials by verifying that each $\mathsf{Enc}(pk_i)$ has a corresponding entry in the published list of the encrypted public keys of eligible voters. Similarly to the technique of Juels et al. [Juels et al. 2010], the tallying authority shuffles the encrypted public keys of the list L_0, and performs a PET with the encrypted public key of each ballot.

We now describe the four steps in detail.

4.4.1. Encrypted Tests Whether Two Ballots Are From The Same Voter. In the first step of the process, for each pair of ballots (b_i, b_j) with $j > i$, the tallying authority applies EPETs (see Section 4.2.2) to the encrypted public keys of the voter. While, technically, this step can be performed during the tally phase, we propose to directly perform it during the casting phase:

When a ballot $b_i = ((\mathsf{Enc}(\beta_i), \mathsf{Enc}(pk_i), \mathrm{ts}_i)$ is cast, the tallying authority checks its NIZKPoK and discards the ballot if the proof is invalid, i.e. the ballot is marked as invalid and not considered

in further computations, but remains on the bulletin board for public verification. Otherwise, the tallying authority jointly runs an EPET on the encrypted public key of b_i and those of all already cast ballots b_j, to obtain values $(d_{i,j}, \Pi_{i,j}) = \mathsf{Epet}(\mathsf{Enc}(pk_i), \mathsf{Enc}(pk_j))$. All encrypted differences $d_{i,j}$ are stored alongside b_j as indicated in Figure 6, defining a list L with entries $L[i] = (b_i, (d_{i,i+1}, \ldots, d_{i,n}))$.

After the casting phase, if n well-formed ballots have been cast, b_i has $n-i$ associated differences.

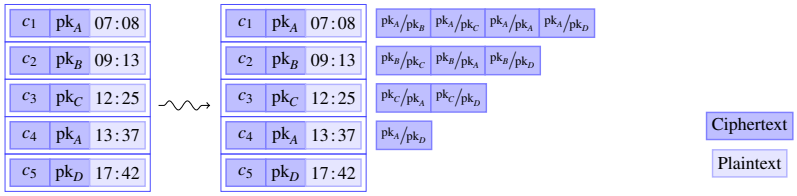

Fig. 6: Encrypted Plaintext Equality Tests (EPETs) are performed for each pair of ballots (b_i, b_j) with $j > i$ by the tallying authority: If $pk_i = pk_j$ the result is an encrypted 1, otherwise it is an encrypted random value. We denote an encrypted value by a box with a dark background and an unencrypted value by a box with a lighter background. In this example, the third fraction in the first row $(\mathrm{pk}_A/\mathrm{pk}_A)^r$ divides identical credentials, hence ballot 4 supersedes ballot 1.

4.4.2. Marking Ballots as Superseded. Before computing the supersede mark o_i for each ballot, the differences $d_{i,j}$ computed during the voting phase are converted as indicated by the mapping

$$\mathsf{Enc}(x) \mapsto \begin{cases} \mathsf{Enc}(r) & \text{if } x = 1 \\ \mathsf{Enc}(1) & \text{if } x \neq 1 \end{cases}$$

The value r can be any fixed value other than 1 (2, for example) or a number drawn at random.

To compute the mapping, the tallying authority creates a shuffled and randomized list of all encrypted differences. To this end, each entry $d_{i,j}$ is associated an encrypted tag $\mathsf{Enc}(\mathrm{ts}_i)$ (see Figure 7).

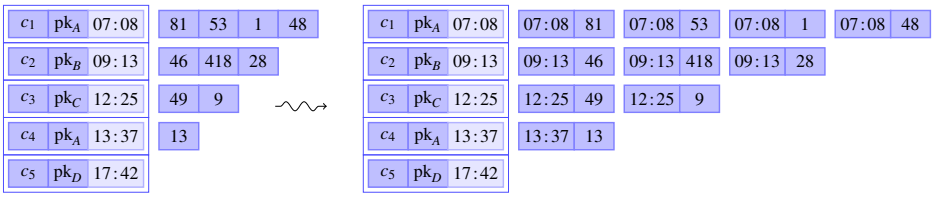

Fig. 7: The fractions from the previous step are complemented by the encrypted timestamp from the ballot, forming tuples $(\mathsf{Enc}(ts_i), d_{i,j})$. For the sake of clarity, we evaluated the fractions from Figure 6 to exemplary values. The superseded first ballot thus has a "1" associated with it.

The list of all tuples $(\mathsf{Enc}(\mathrm{ts}_i), d_{i,j})$ is then re-randomized and reordered, using a verifiable secret shuffle. After shuffling, the differences are converted (see Figure 8). To convert $d_{i,j}$ to $d'_{i,j}$ we perform a multi-party computation that involves a "coordinator" and the voting authority. In this step, the authority's task is to decrypt each $d_{i,j}$ it receives from the coordinator and return either an encryption of a 1 or of another (random or fixed) number, according to the map above. The task of the coordinator is to send fake differences or real ones to the authority. He has to make sure that the authority does not learn which conversion is real. Therefore, each element is randomized before and after each conversion by the coordinator. The randomized version of the output of each "real" conversion is fed to the authority to be converted a second time and get a fake difference. The second conversion is necessary to hide how many of the $d_{i,j}$ contain a 1. All (real and fake) differences are sent to the authority in random order. The coordinator and the authority must not collude.

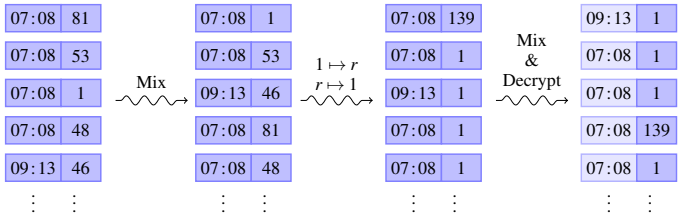

Fig. 8: Converting $d_{i,j}$ to $d'_{i,j}$: if $\mathsf{Dec}(d_{i,j}) = 1$ replace $d_{i,j}$ by $\mathsf{Enc}(r)$ ($r \neq 1$) and by $\mathsf{Enc}(1)$ else. The special "superseded" value 1 has been converted to an arbitrary number, while all other values are mapped to 1.

Irrespective of how the entries $d_{i,j}$ are converted to $d'_{i,j}$, their correctness can easily be proven: either $\mathsf{Pet}(d_{i,j} \cdot d'_{i,j},\ d_{i,j}) = 1$ or $\mathsf{Pet}(d_{i,j} \cdot d'_{i,j},\ d'_{i,j}) = 1$, never both. To prove the correctness of the inversion without revealing the values, we use a verifiable shuffle on $(d_{i,j},\ d'_{i,j})$ to obtain (a,b). We then check whether $\mathsf{Pet}(a \cdot b,\ a) = 1$ or $\mathsf{Pet}(a \cdot b,\ b) = 1$ (exclusively). See Appendix A for a compact description of the procedure.

After a second shuffling step, the tags are decrypted, and the encrypted, converted differences are sorted back to their ballot of origin (see step one in Figure 9). All steps can be verified by the public. Then the tallying authority computes the homomorphic product over all the associated marks of each ballot (see step two in Figure 9): for each i compute

$$o_i := \prod_{j=\min(i+1,n)}^{n} d'_{i,j}.$$

Observe that if $\prod d'_{i,j}$ contains only encryptions of 1, it is itself an encryption of a 1, whereas if it has a factor $\neq 1$, it is itself an encryption of a number $\neq 1$ with overwhelming probability.

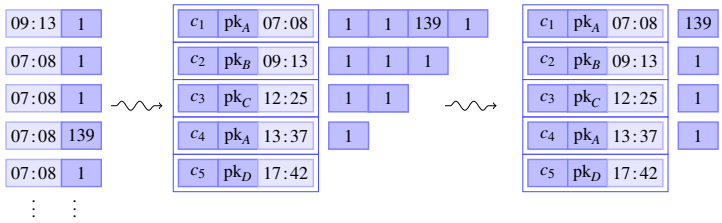

Fig. 9: The encrypted, converted differences are sorted back to their ballot of origin. Afterwards o_i is computed as the homomorphic product over all the associated marks of ballot b_i. Note that we arranged the $d'_{i,j}$ values in the same order as their preimages. In a practical realization of our scheme this would only coincidentally be the case. Ballots now have an encrypted "tag" that tells whether they are superseded by a more recent ballot.

4.4.3. Omit Superseded Ballots. Before any further processing, particularly before checking if $o_i = Enc(1)$, all ballots are weeded and shuffled. Only the encrypted choice of the voter, his encrypted public key, and the mark o_i are kept. Therefore, the tallying authority forms a list L_W with entries $L_W[i] := (b'_i, o_i)$, where $b'_i := (\mathsf{Enc}(\beta_i), \mathsf{Enc}(pk_i))$ if $b_i = (\mathsf{Enc}(\beta_i), \mathsf{Enc}(pk_i), \mathrm{ts}_i)$. The tallying authority computes and publishes $(L'_W, \Pi) := \mathsf{Shuffle}(\mathsf{L_W})$ and then jointly compares o_i with $\mathsf{Enc}(1)$ in L'_W using a PET, and publishes all proofs. Only ballots b_i with $o_i = \mathsf{Enc}(1)$ and the last ballot are kept, the others are marked as discarded.

4.4.4. Omit Ballots With Invalid Credentials. Finally, the validity of the public keys is checked. Note that, at this point, only one ballot per credential remains. Equally to the method of Juels et al., the tallying authority shuffles the encrypted public keys of L_0 and performs a joint PET with the

encrypted public key of each ballot. Ballots with valid public keys are retained. The encrypted choices of the retained ballots can then be mixed and opened for tallying.

4.5. Proving the Security of our Scheme

In this section, we prove deniability of revoting in our protocol. Correctness and verifiability are addressed in Appendix B and C, respectively.

To prove the deniability of revoting, we give a reduction of the privacy of our scheme to the Decisional Diffie-Hellman (DDH) problem. More concretely, given any (successful) adversary $\mathcal{A}$ against $\mathbf{Exp}^{\textit{revoting-c-resist}}_{ES,\mathcal{A},H}$, we construct a simulator $\mathcal{S}$ which is, with non-negligible probability, able to decide for a given quadruple (g, g^a, g^b, g^c) whether $c = ab$ in the cyclic group $G = \langle g \rangle$. As in the definition, we consider only one coerced voter. An extension to the case of multiple coerced voters is straightforward—we just have accordingly many parallel instances of the DDH problem. Our proof is similar to that of Juels et al. [Juels et al. 2005]. We give a reduction from any adversary that uses the published ballots to break the incoercibility of the scheme to an adversary against the DDH assumption. To this end, we simulate the execution of experiment $\mathbf{Exp}^{\textit{revoting-c-resist}}_{ES,\mathcal{A},H}$.

The DDH experiment $\mathbf{Exp}^{\mathsf{ddh}}_{G}$ secretly draws a bit d, if $d = 1$ sets $g^c = g^{ab}$, and to a random element $g^c \in G$ otherwise. Given a DDH challenge (g, g^a, g^b, g^c), $\mathcal{S}$ deducts two public keys. In the setup publication phase, $\mathcal{S}$ sends the first one to the adversary, while it uses the second one during the rest of the protocol. $\mathcal{S}$ deducts the following two m-Elgamal public keys:

$$(g_1 := g,\ g_2 := g^a,\ y_g := g_1^{x_1} g_2^{x_2} = g^{x_1} g^{a x_2}) \text{ and}$$

$$(h_1 := g^b,\ h_2 := g^c,\ y_h := h_1^{x_1} h_2^{x_2} = g^{b x_1} g^{c x_2}).$$

Recall that, to encrypt m using the Modified Elgamal Encryption (see Section 4.2.1) using public key (g_1, g_2, y), one chooses a random r and outputs $(g_1^r, g_2^r, y^r m)$. When $c = ab$ ($d = 1$ in the surrounding experiment), for any m there are r, r' such that $(g_1^{r'}, g_2^{r'}, y_g^{r'} m) = (h_1^r, h_2^r, y_h^r m)$. ($r' = br$, concretely.) Therefore, ciphertexts created using the above public keys have the same distribution.

When, on the other hand, $c \neq ab$ ($d = 0$), m is perfectly hidden in the encryption. A message m, encrypted with the second public key and decrypted with the first (given to $\mathcal{A}$), yields:

$$(h_1^r = g^{br} = g_1^{br}, h_2^r = g^{cr} = g_2^{(c/a)r},$$

$$y_h^r m = g^{rbx_1} g^{rcx_2} m = g^{rbx_1} g^{abrx_2} g^{(c-ab)rx_2} m = y_g^{br} g^{(c-ab)rx_2} m)$$

In $\mathbf{Exp}^{\textit{revoting-c-resist}}_{ES,\mathcal{A},H}$ the choices of the voters as well as their public keys are perfectly hidden in the m-Elgamal Encryption when $d = 0$. In this case the adversary's capabilities are reduced to those in experiment $\mathbf{Exp}^{\textit{revoting-c-resist-ideal}}_{ES,\mathcal{A},H}$. To justify this assertion, we must show how the adversary's input can be faked in the $d = 0$ case with only the information from $\mathbf{Exp}^{\textit{revoting-c-resist-ideal}}_{ES,\mathcal{A},H}$ available. It is then obvious that the adversary cannot gain any additional advantage from the simulated tallying process.

In the ideal experiment $\mathbf{Exp}^{\textit{revoting-c-resist-ideal}}_{ES,\mathcal{A},H}$ the adversary learns the total number of cast ballots and the number of valid ballots. The individual m-Elgamal encryptions the simulator outputs to the adversary hide the plaintext perfectly when $d = 0$. In this case, they are not distinguishable from any random group element and thus can be faked by random output. We must also be able to fake the quantity and structure of the data the adversary learns during the simulation. These can be determined from the difference of cast and valid ballots, and vice versa.

We state the simulation of $\mathbf{Exp}^{\textit{revoting-c-resist}}_{ES,\mathcal{A},H}$ and argue that its perfect. The simulator $\mathcal{S}$ receives a $W \in D_{n_U, n_C}$ and a challenge (g, g^a, g^b, g^c) with $c = ab$ if the random bit $d = 1$ or $c \neq ab$ if $d = 0$.

Setup. The simulator $\mathcal{S}$ chooses the secret key $SK_{\mathcal{T}} := (x_1, x_2)$, uniformly and at random. $\mathcal{S}$ also calculates the public key $PK_{\mathcal{T}} := (g^{x_1}, g^{x_2}, h = (g_1^{x_1} g_2^{x_2}) \mod p$ and sets $C = \{c_i\}_{i=1}^{n_C}$.

Registration. $\mathcal{S}$ simulates the registrar $\mathcal{R}$: $\mathcal{S}$ generates the voter credentials: $\{(pk_i, sk_i)\}_{i=1}^{n_V}$
Setup publication. $\mathcal{S}$ publishes the public key of the tally $PK_{\mathcal{T}}$, the set of candidates C, and $L_0 = \{$"voter's name i", $\mathsf{Enc}(pk_i)\}_{i=1}^{n_V}$.
Adversarial corruption. The adversary $\mathcal{A}$ selects n_A voters to corrupt. Let V denote the group of corrupted voters. He also selects a voter j for coercion. He determines the choices β for the corrupted and the coerced voters. The simulation is terminated if the selection is invalid.
Coin flip. The bit b is chosen uniformly at random: $b \in_U \{0,\ 1\}$
Honest voter simulation. For each honest voter, the simulator creates the ballots with the proofs:

$$A_0 := \{(\mathsf{Enc}(\beta_i), \mathsf{Enc}(pk_i), \mathrm{ts}_i),\ NIZKPoK\}$$

$$= \{b_i := ((h_1^{r_i}, h_2^{r_i}, h_1^{r_i x_1} h_2^{r_i x_2} c_j), (h_1^{k_i}, h_2^{k_i}, h_1^{k_i x_1} h_2^{k_i x_2} pk_i), ts), NIZKPoK\}_{i=1}^{n}$$

Thereby all r_i and k_i are chosen at random in Z_q. The choices are determined in W. For creating the proofs the simulator $\mathcal{S}$ uses the voter's secret keys.
Adversarial ballot posting. The adversary calculates the ballots for each voter $v \in V$ and the voter j in the same way. We call the set of these ballots B_0. $\mathcal{A}$ posts B_0.
Tallying simulation. $\mathcal{S}$ simulates a honest tallying authority, whereby it uses the secret key $SK_{\mathcal{T}}$ from the setup step. Since the correctness of every step from each authority can be verified, any modification from an adversary can be ignored.

Proof checking. The proof of each ballot in A_0 and B_0 is checked. Let E_1 be all ballots with valid proofs.
Creating $d_{i,j}$. The simulator $\mathcal{S}$ creates the $d_{i,j}$ by performing the necessary EPETs. $\mathcal{S}$ also creates the proofs honestly.
Converting $d_{i,j}$. $\mathcal{S}$ performs the protocol honestly and provides the proofs. During this process, it uses the secret key $SK_{\mathcal{T}}$ to decrypt the EPETs.
Creating o_i. $\mathcal{S}$ accumulates the $d_{i,j}$ as scheduled.
Creating final ballot list. For each ballot in E_1 create a cut list as described in the protocol. $\mathcal{S}$ creates a shuffle and the according proofs. Denote the result as E_2. $\mathcal{S}$ creates list of tuples, using the secret key: $E_3 := \{(\mathsf{Enc}(\beta_i), \mathsf{Enc}(pk_i))\} : \mathsf{Pet}(o_i, \mathsf{Enc}(1)) = 1$
Checking Voter Credentials. $\mathcal{S}$ shuffles the second components of L_0 into a new list L_1: Let L_0' be the list of all encrypted public keys in L_0. $L_1 := \mathsf{Shuffle}(\mathsf{L}_0')$. For each ballot in E_3, $\mathcal{S}$ performs PETs with the second part of each entry. $\mathcal{S}$ uses the secret key for decryption. Let E_4 denote the list of all encrypted choices from the ballots with a according item in L_1.
Choice decryption. The decryption of the valid choices (E_4) is done by $\mathcal{S}$ with the secret key.
Adversarial guess. The adversary outputs a guess bit b'. $\mathcal{S}$ itself returns $d' := (b' \stackrel{?}{=} b)$ as his guess bit, i.e. whether the adversary correctly guessed b.

Since the simulator executes the protocol as specified, for $d = 1$ the simulation is indistinguishable to $\mathcal{A}$ from a real protocol. Let $\mathcal{V}$ denote the view of the adversary. Thus, we have

$$\Pr[S = 1 \mid d = 1] = Pr[\mathbf{Exp}_{ES,\mathcal{A},H}^{revoting\text{-}c\text{-}resist}(\mathcal{V}) = 1] = \mathbf{Succ}_{ES,\mathcal{A},H}^{revoting\text{-}c\text{-}resist}(\mathcal{V}).$$

On the other hand, as we argued above, the adversary in the simulation does not have any advantage to the adversary in $\mathbf{Exp}_{ES,\mathcal{A},H}^{revoting\text{-}c\text{-}resist\text{-}ideal}(\mathcal{V})$ if $d = 0$. Thus,

$$\Pr[S = 1 \mid d = 0] = Pr[\mathbf{Exp}_{ES,\mathcal{A},H}^{revoting\text{-}c\text{-}resist\text{-}ideal}(\mathcal{V}) = 1] = \mathbf{Succ}_{ES,\mathcal{A},H}^{revoting\text{-}c\text{-}resist\text{-}ideal}(\mathcal{V}).$$

Concluding the argument, we have

$$\begin{aligned}\mathbf{Adv}_{\mathcal{S}}^{\mathsf{ddh}} &= \Pr[S = 1 \mid d = 1] - Pr[S = 1 \mid d = 0] \\ &= \mathbf{Succ}_{ES,\mathcal{A},H}^{revoting\text{-}c\text{-}resist}(\mathcal{V}) - \mathbf{Succ}_{ES,\mathcal{A},H}^{revoting\text{-}c\text{-}resist\text{-}ideal}(\mathcal{V}) \\ &= \mathbf{Adv}_{\mathcal{A}}^{revoting}.\end{aligned}$$

□

REFERENCES

Masayuki Abe, Kristiyan Haralambiev, and Miyako Ohkubo. 2010. Signing on Elements in Bilinear Groups for Modular Protocol Design. Cryptology ePrint Archive, Report 2010/133. (2010). http://eprint.iacr.org/.

Masayuki Abe and Fumitaka Hoshino. 2001. Remarks on Mix-Network Based on Permutation Networks. In *Public Key Cryptography (Lecture Notes in Computer Science)*, Kwangjo Kim (Ed.), Vol. 1992. Springer, 317–324.

Ben Adida, Olivier De Marneffe, Olivier Pereira, and Jean-Jacques Quisquater. 2009. Electing a University President Using Open-audit Voting: Analysis of Real-world Use of Helios. In *Proceedings of the 2009 Conference on Electronic Voting Technology/Workshop on Trustworthy Elections (EVT/WOTE'09)*. USENIX Association, Berkeley, CA, USA, 10–10. http://dl.acm.org/citation.cfm?id=1855491.1855501

Roberto Araújo, Sébastien Foulle, and Jacques Traoré. 2010. A Practical and Secure Coercion-Resistant Scheme for Internet Voting. In *Towards Trustworthy Elections*, David Chaum, Markus Jakobsson, RonaldL. Rivest, PeterY.A. Ryan, Josh Benaloh, Miroslaw Kutylowski, and Ben Adida (Eds.). Lecture Notes in Computer Science, Vol. 6000. Springer Berlin Heidelberg, 330–342. DOI: http://dx.doi.org/10.1007/978-3-642-12980-3_20

Josh Benaloh. 2013. Rethinking Voter Coercion: The Realities Imposed by Technology. *Presented as part of the USENIX Journal of Election and Technology and Systems (JETS)* (2013), 82–87.

Jan Camenisch and Markus Stadler. 1997. Efficient group signature schemes for large groups. In *Advances in Cryptology CRYPTO '97*, Jr. Kaliski, BurtonS. (Ed.). Lecture Notes in Computer Science, Vol. 1294. Springer Berlin Heidelberg, 410–424. DOI: http://dx.doi.org/10.1007/BFb0052252

Ran Canetti and Rosario Gennaro. 1996. Incoercible Multiparty Computation. Cryptology ePrint Archive, Report 1996/001. (1996). http://eprint.iacr.org/.

David Chaum and Torben P. Pedersen. 1993. Wallet Databases with Observers. In *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO '92)*. Springer-Verlag, London, UK, UK, 89–105. http://dl.acm.org/citation.cfm?id=646757.705670

Jeremy Clark and Urs Hengartner. 2011a. Selections: Internet Voting with Over-the-Shoulder Coercion-Resistance. In *Financial Cryptography (Lecture Notes in Computer Science)*, George Danezis (Ed.), Vol. 7035. Springer, 47–61.

Jeremy Clark and Urs Hengartner. 2011b. Selections: Internet Voting with Over-the-Shoulder Coercion-Resistance. Cryptology ePrint Archive, Report 2011/166. (2011). http://eprint.iacr.org/.

Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. 1997. A Secure and Optimally Efficient Multi-Authority Election Scheme. Springer-Verlag, 103–118.

Stéphanie Delaune, Steve Kremer, and Mark D. Ryan. 2009. Verifying Privacy-type Properties of Electronic Voting Protocols. *Journal of Computer Security* 17, 4 (July 2009), 435–487. DOI: http://dx.doi.org/10.3233/JCS-2009-0340

Taher Elgamal. 1985. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *Advances in Cryptology Proceedings of CRYPTO 84*, David Chaum George Robert Blakley (Ed.). Springer-Verlag, Berlin, Heidelberg, 10–18. http://www.springerlink.com/content/jl0mkpm32tn8ve3q/

Aleksander Essex, Jeremy Clark, and Urs Hengartner. 2012. Cobra: Toward Concurrent Ballot Authorization for Internet Voting. In *Proceedings of the 2012 International Conference on Electronic Voting Technology/Workshop on Trustworthy Elections (EVT/WOTE'12)*. USENIX Association, Berkeley, CA, USA, 3–3. http://dl.acm.org/citation.cfm?id=2372353.2372356

Kristian Gjøsteen. 2010. Analysis of an internet voting protocol. *IACR Cryptology ePrint Archive* 2010 (2010), 380.

Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. 1988. A Digital Signature Scheme Secure Against Adaptive Chosen-message Attacks. *SIAM J. Comput.* 17, 2 (April 1988), 281–308. DOI: http://dx.doi.org/10.1137/0217017

Philippe Golle, Markus Jakobsson, Ari Juels, and Paul Syverson. 2004. Universal Re-encryption for Mixnets. In *Topics in Cryptology – CT-RSA 2004*, Tatsuaki Okamoto (Ed.). Lecture Notes in Computer Science, Vol. 2964. Springer Berlin Heidelberg, 163–178. DOI: http://dx.doi.org/10.1007/978-3-540-24660-2_14

James Green-Armytage. 2014. Direct Voting and Proxy Voting. (2014). Available at http://inside.bard.edu/~armytage/proxy.pdf.

Gurchetan S. Grewal, Mark D. Ryan, Sergiu Bursuc, and Peter Y. A. Ryan. 2013. Caveat Coercitor: Coercion-Evidence in Electronic Voting. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy (SP '13)*. IEEE Computer Society, Washington, DC, USA, 367–381. DOI: http://dx.doi.org/10.1109/SP.2013.32

Jens Groth. 2002. A Verifiable Secret Shuffe of Homomorphic Encryptions. In *Public Key Cryptography - PKC 2003*, Yvo Desmedt (Ed.). Lecture Notes in Computer Science, Vol. 2567. Springer Berlin / Heidelberg, 145–160. http://dx.doi.org/10.1007/3-540-36288-6_11 10.1007/3-540-36288-6_11.

Jens Groth and Amit Sahai. 2008. Efficient Non-interactive Proof Systems for Bilinear Groups. In *Proceedings of the Theory and Applications of Cryptographic Techniques 27th Annual International Conference on Advances in Cryptology (EUROCRYPT'08)*. Springer-Verlag, Berlin, Heidelberg, 415–432. http://dl.acm.org/citation.cfm?id=1788414.1788438

Markus Jakobsson and Ari Juels. 2000. Mix and Match: Secure Function Evaluation via Ciphertexts. In *Advances in Cryptology ASIACRYPT 2000*, Tatsuaki Okamoto (Ed.). Lecture Notes in Computer Science, Vol. 1976. Springer Berlin Heidelberg, 162–177. DOI: http://dx.doi.org/10.1007/3-540-44448-3_13

Ari Juels, Dario Catalano, and Markus Jakobsson. 2005. Coercion-resistant electronic elections. In *WPES '05: Proceedings of the 2005 ACM workshop on Privacy in the electronic society*. ACM, New York, NY, USA, 61–70. http://markus-jakobsson.com/papers/jakobsson-wpes05.pdf

Ari Juels, Dario Catalano, and Markus Jakobsson. 2010. Coercion-Resistant Electronic Elections. In *Towards Trustworthy Elections*, David Chaum, Markus Jakobsson, RonaldL. Rivest, PeterY.A. Ryan, Josh Benaloh, Miroslaw Kutylowski, and Ben Adida (Eds.). Lecture Notes in Computer Science, Vol. 6000. Springer Berlin Heidelberg, 37–63. DOI: http://dx.doi.org/10.1007/978-3-642-12980-3_2

Shahram Khazaei, Tal Moran, and Douglas Wikström. 2012. A Mix-Net from Any CCA2 Secure Cryptosystem. In *Advances in Cryptology ASIACRYPT 2012*, Xiaoyun Wang and Kazue Sako (Eds.). Lecture Notes in Computer Science, Vol. 7658. Springer Berlin Heidelberg, 607–625. DOI: http://dx.doi.org/10.1007/978-3-642-34961-4_37

Ralf Küsters, Tomasz Truderung, and Andreas Vogt. 2012. A Game-Based Definition of Coercion-Resistance and its Applications. *Journal of Computer Security (special issue of selected CSF 2010 papers)* 20, 6/2012 (2012), 709–764.

Mirosław Kutyłowski and Filip Zagórski. 2007. Verifiable Internet Voting Solving Secure Platform Problem. In *Advances in Information and Computer Security*, Atsuko Miyaji, Hiroaki Kikuchi, and Kai Rannenberg (Eds.). Lecture Notes in Computer Science, Vol. 4752. Springer Berlin Heidelberg, 199–213. DOI: http://dx.doi.org/10.1007/978-3-540-75651-4_14

Epp Maaten. 2004. Towards Remote E-Voting: Estonian case.. In *Electronic Voting in Europe (LNI)*, Alexander Prosser and Robert Krimmer (Eds.), Vol. 47. GI, 83–100. http://dblp.uni-trier.de/db/conf/evoting/evoting2004.html#Maaten04

James C. III Miller. 1969. A program for direct and proxy voting in the legislative process. *Public Choice* 7, 1 (1969), 107–113. DOI: http://dx.doi.org/10.1007/BF01718736

Tal Moran and Moni Naor. 2006. Receipt-Free Universally-Verifiable Voting With Everlasting Privacy. In *Advances in Cryptology – CRYPTO 2006 (Lecture Notes in Computer Science)*, Cynthia Dwork (Ed.), Vol. 4117. Springer, 373–392.

Andrew C. Myers, Michael Clarkson, and Stephen Chong. 2008. Civitas: Toward a Secure Voting System. In *IEEE Symposium on Security and Privacy*. IEEE, 354–368. http://www.truststc.org/pubs/450.html

C. Andrew Neff. 2001. A verifiable secret shuffle and its application to e-voting. CCS '01 Proceedings of the 8th ACM conference on Computer and Communications Security. (2001).

Gerald V. Post. 2010. Using re-voting to reduce the threat of coercion in elections. In *Electronic Government, an International Journal (Volume 7, Number 2/2010)*. Inderscience Publishers, 168–182. https://inderscience.metapress.com/content/e841t68786434728/resource-secured/?target=fulltext.pdf

Riigikogu. 2002. Riigikogu Election Act. Riigi Teataja. (2002). https://www.riigiteataja.ee/en/eli/ee/501092014005/consolide/current.

Kazue Sako and Joe Kilian. 1995. Receipt-free mix-type voting scheme: a practical solution to the implementation of a voting booth. In *Proceedings of the 14th annual international conference on Theory and application of cryptographic techniques (EUROCRYPT'95)*. Springer-Verlag, Berlin, Heidelberg, 393–403. http://dl.acm.org/citation.cfm?id=1755009.1755052

C.P. Schnorr. 1991. Efficient signature generation by smart cards. *Journal of Cryptology* 4, 3 (1991), 161–174. DOI: http://dx.doi.org/10.1007/BF00196725

Warren D. Smith. 2005. New cryptographic election protocol with best-known theoretical properties. Frontiers in Electronic Elections (FEE 2005). (2005).

Oliver Spycher, Rolf Haenni, and Eric Dubuis. 2010. Coercion-resistant hybrid voting systems. In *Electronic Voting*. 269–282.

Dominique Unruh and Jörn Müller-Quade. 2010. Universally Composable Incoercibility. In *Crypto 2010 (LNCS)*, Vol. 6223. Springer, 411–428. Preprint on IACR ePrint 2009/520.

Melanie Volkamer and Rüdiger Grimm. 2006. Multiple Casts in Online Voting: Analyzing Chances. In *Electronic Voting 2006: 2nd International Workshop, Co-organized by Council of Europe, ESF TED, IFIP WG 8.6 and E-Voting.CC, August, 2nd - 4th, 2006 in Castle Hofen, Bregenz, Austria. (LNI)*, Robert Krimmer (Ed.), Vol. 86. GI, 97–106. http://subs.emis.de/LNI/Proceedings/Proceedings86/article4554.html

A. TAG CONVERSION

In Section 4.4 we informally describe how to create encrypted marks that tell which ballots have been superseded. Here we define the procedure more formally. The procedure starts with the list of all ballots and a corresponding list of differences next to each ballot and attaches an encrypted mark to each ballot. The encrypted mark tells whether the ballot has been superseded.

We use subroutines like a secret verifiable shuffle (see Section 4.1.1). Since each of those subroutines is verifiable, it produces a proof of correctness beside the actual result. Each proof contributes to the over-all proof of correctness. In abuse of notation and for better readability, we do not explicitly mention them.

Conversion $\mathsf{Convert}(L)$

Input. A list L with (ballot,differences)-entries $L[i] = (b_i, (d_{i,i+1}, \ldots, d_{i,n}))$

Output. A converted list L' of ballots and differences, and a list P of proofs of correctness.

Algorithm. (Each operation which can not be recomputed by everyone outputs a proof of correctness. For the sake of readability, we do not explicitly write the proof-output, but assume it implicitly appended to the list P.)

(1) Initialize list L' with $L[i] := (b_i, [])$ (the second entry will be filled later).
Initialize a look up table TS with an entry (i, ts_i) for each ballot $b_i = (\mathsf{Enc}(\beta_i), \mathsf{Enc}(pk_i), \mathrm{ts}_i)$

(2) For each ballot b_i prepare a tag $\mathsf{Enc}(\mathrm{ts}_i)$.

(3) $\forall 1 \le i \le n,\ \min(i+1, n) \le j \le n$: Save $(\mathsf{Enc}(\mathrm{ts}_i), d_{i,j}, [], [])$ to a list D. The empty entries will be filled later

(4) Shuffle D: $D' := \mathsf{Shuffle}(\mathsf{D})$.

(5) For each $d_{i,j}$ in D'
convert $d_{i,j}$ to $d'_{i,j} := \begin{cases} \mathsf{Enc}(r) & \text{if } d_{i,j} = \mathsf{Enc}(1) \\ \mathsf{Enc}(1) & \text{else} \end{cases}$ where $r \neq 1$
and store $d'_{i,j}$ in D' next to $d_{i,j}$: $(\mathsf{Enc}(\mathrm{ts}_i), d_{i,j}, d'_{i,j}, [])$.

(6) create the proof Π of correct conversion and store it in D': $(\mathsf{Enc}(\mathrm{ts}_i), d_{i,j}, d'_{i,j}, \Pi)$.
The correctness of the conversion can be verified based on D'.

(7) Shuffle D': $D'' := \mathsf{Shuffle}(\mathsf{D}')$

(8) For each $(\mathsf{Enc}(\mathrm{ts}_i), d'_{i,j}) \in D''$:
— Decrypt tag $\mathsf{Enc}(\mathrm{ts}_i)$ to ts_i with proof of correct decryption
— look up entry (i, ts_i) in TS
— append $d'_{i,j}$ to the second entry of $L'[i]$.

(9) For each ballot b_i: multiply each element $d'_{i,j}$ in the second entry of L'[i] and replace them by the result

(10) Output (L', P).

Fig. 10: Marking each ballot whether it has been superseded.

B. CORRECTNESS

Recall that a voting scheme is correct if the success probability of any adversary in Experiment $\mathbf{Exp}^{corr}_{ES,\mathcal{A}}(k_1, k_2, k_3, n_V, n_C)$ (Figure 2) is negligible in $\max(k_1, k_2, k_3)$. We show that our construction is correct by an induction over the number of ballots n on the bulletin board in Step 8 of experiment $\mathbf{Exp}^{corr}_{ES,\mathcal{A}}$. The number $n = n_{\mathcal{V}} + n_{\mathcal{A}}$ is comprised of $n_{\mathcal{V}}$, the number of ballots posted by honest voters on the bulletin board ($\mathcal{BB}$) in Step 4, and $n_{\mathcal{A}}$, the number of ballots posted on $\mathcal{BB}$ by the adversary in Step 6. Per assumption, the bulletin board is append-only.

Fix $n = 1$ as the base case, i.e. either $n_{\mathcal{A}} = 0$ or $n_{\mathcal{V}} = 0$. If $n_{\mathcal{A}} = 0$, then $\mathbf{X}' = \mathbf{X}$ and $P' = P$, and the experiment outputs 0. On the other hand, if $n_{\mathcal{V}} = 0$, then we have to show that the adversary can only cast a vote for a corrupted voter. (The adversary cannot overwrite an uncorrupted voter's vote, because there are none.) Ballots containing an invalid voter public key, i.e. a public key $pk \notin L_0$ are discarded in the tallying phase (see Section 4.4, Step 5). The probability that the adversary

generates a public key pair (sk, pk) such that sk generates a valid signature for the ballot and $pk \in L_0$ is negligible in the security parameter of the signature scheme. Thus, the experiment outputs 0 except for a negligible probability. (We stress that "malformed" ballots without a correct proof of knowledge of a signature on the ballot are discarded before the tallying phase of our protocol.)

Now assume there are $n = n_{\mathcal{V}} + n_{\mathcal{A}}$ ballots on the bulletin board in Step 8 of $\mathbf{Exp}^{corr}_{ES,\mathcal{A}}$ and the probability that the experiments outputs 1 is negligible in the security parameters for any adversary. To transfer to the $n+1$-case, we distinguish two cases to post an additional ballot on $\mathcal{BB}$.

— The additional ballot is cast in Step 4, i.e. by an uncontrolled voter. We have to show that it supersedes the most recent ballot by the same voter. This is achieved by a pairwise comparison of the public keys on the ballots (see Section 4.4).
— The additional ballot is cast in Step 6, i.e. by a controlled voter. We have to show that it does not supersede a ballot by a different voter and itself is not already superseded. By the same argument as above, only ballots that contain identical public keys are superseded during the tallying phase. (For the adversary to post a ballot containing the public key of an uncontrolled voter, he would have to forge the signature of the ballot and thus break the EUF-CMA security of the signature scheme.) Because the newest ballot has the newest timestamp per assumption, it is not superseded in this step.

This concludes the argument.

□

C. VERIFIABILITY

We argue the verifiability of our voting scheme informally. A process composed of several subroutines is verifiable if each step is verifiable by itself. Our process is composed of a small set of verifiable black-box functionalities and other verifiable buildings blocks. We describe them in Sections 4.1 and 4.2.

Each of these subroutines produces proofs of correctness along with its result. If either of these proofs is invalid, $\mathbf{Exp}^{ver}_{ES,\mathcal{A}}$ outputs 0. All these proofs are published next to the result of the subroutine on the bulletin board. The whole bulletin board with the input $\{b_i\}_{i=1}^{n}$, the final output $\mathbf{X}$, all subroutine proofs, and all interim results is the global proof P of correctness of the tally. We point out that each interim result is published except for the secret keys and some randomness which is necessary to protect the privacy of the voter. The input of each subroutine is also marked down on the bulletin board. Sometimes the output from one routine has to be reshaped to match the input format from another one. This reshaping is always deterministic.

Several operations like the reshaping or the computation of the product over all marks $d_{i,j}$ of each ballot b_i can be recomputed by anyone. Therefore no explicit proof is necessary for these steps. All operations which are not recomputable by the public are accompanied by proofs of correctness.

In conclusion, our tallying process is verifiable because intermediate values are public, and all subroutines are publicly verifiable.