Machine-Checked Proofs of Accountability: How to sElect who is to Blame

Constantin Catalin Dragan, François Dupressoir, Kristian Gjøsteen, Thomas Haines, Peter Rønne, Morten Rotvold Solberg

Machine-Checked Proofs of Accountability: How to sElect who is to Blame

Constantin Cătălin Drăgan[1], François Dupressoir[2], Kristian Gjøsteen[3], Thomas Haines[4], Peter B. Rønne[5], and Morten Rotvold Solberg[3]

[1] University of Surrey, Guildford, United Kingdom c.dragan@surrey.ac.uk
[2] University of Bristol, Bristol, United Kingdom f.dupressoir@bristol.ac.uk
[3] Norwegian University of Science and Technology, Trondheim, Norway
{kristian.gjosteen,mosolb}@ntnu.no
[4] Australian National University, Canberra, Australia thomas.haines@anu.edu.au
[5] CNRS, LORIA, Université de Lorraine, Nancy, France peter.roenne@loria.fr

Abstract. Accountability is a critical requirement of any deployed voting system as it allows unequivocal identification of misbehaving parties, including authorities. In this paper, we propose the first game-based definition of accountability and demonstrate its usefulness by applying it to the sElect voting system (Küsters *et al.*, 2016) – a voting system that relies on no other cryptographic primitives than digital signatures and public key encryption.
We strengthen our contribution by proving accountability for sElect in the EasyCrypt proof assistant. As part of this, we identify a few errors in the proof for sElect as presented by Küsters *et al.* (2016) for their definition of accountability.
Finally, we reinforce the known relation between accountability and verifiability, and show that it is still maintained by our new game-based definition of accountability.

1 Introduction

A system is accountable if, when something goes wrong, it is possible to judge who is responsible based on evidence provided by the system participants. For a voting system, this means that if we do not accept the outcome of an election, the honest parties should be able to produce evidence that pinpoints who is to blame, in the sense that they have not followed the protocol. This is in principle trivial for some voting systems, such as the Helios voting system where each party proves their correct behaviour using zero knowledge arguments. This is, however, not trivial for every reasonable voting system, in particular voting systems with complex ballot submission procedures, such as the Swiss Post voting system [21]; in the Swiss Post case the system involves a complicated protocol between half a dozen participants to decide if a ballot was cast by a valid voter and well-formed and hence should be counted.

The sElect voting system [17] is an interesting case for accountability. Unlike Helios, the system does not use any advanced cryptography, relying entirely on

secure public key encryption. The system uses nested public key encryption to allow a very simple mixnet decryption. The voter creates a nested encryption of their ballot and a random check value, each layer encrypted with a mix server public key. Each mix server decrypts one layer of encryption, sorting the result lexicographically to effect mixing. The last mix server simply outputs decrypted ballots, together with the voter-specific check value. Voters verify that their ballot is included in the count by checking that the ballot appears together with the voter's check value.

Informally, the sElect system is accountable because voters can reveal the randomness used in the nested encryption, thereby enabling tracing of the encrypted ballot through the mixnet, which will pinpoint which mix server did not correctly decrypt.

Accountability might seem to be a fairly simple notion, but it is technically difficult to find a definition that both captures accountability and is easy to work with. This can be seen from the fact that no definition of accountability seems to have been broadly accepted in the community. Also, when Küsters *et al.* [17] apply the definition from [18] to sElect, there are a number of errors in the result they claim; we will discuss these in greater length in Sec. 1.2. These errors suggest that the existing accountability definitions are hard to work with. In other words, there is a need for a workable general definition of accountability.

The simplicity of sElect comes at a cost, which is that the system is only private for voters that accept the election outcome. This problem can be mitigated using the final cryptosystem trick from [12]; with this trick, "a sender first encrypts her message under the "final" public key and uses this encrypted message as an input to the protocol as described so far. This innermost encryption layer is jointly decrypted only if the protocol does not abort. If the protocol does abort, only the encrypted values are revealed and privacy is protected by the final layer of encryption." However, using this mitigation in sElect would require the voters' devices to check the mix before the result is decrypted which substantially complicates the protocol and delays the tally result, which would be unacceptable in most cases.

Privacy is of course essential for voting systems, but we note that we are not studying privacy in this paper, only accountability, since the privacy of sElect is well-understood.

1.1 Our Contribution

This paper contains two main contributions: The first game-based definition of accountability, and a proof of accountability for the sElect [17] voting system. A variant of the latter proof has been formalised in the EasyCrypt [3] proof assistant.

This game-based definition is significant because this style of definitions are often easier to understand and work with. For security proofs, ease of understanding and use is a significant factor in getting things right and later verifying that things are indeed correct. Further, it allows us to use existing tools for game-based proofs, specifically EasyCrypt, to formally verify security.

The accountability proof for sElect is significant, first because it demonstrates that our new definition of accountability works. Second, the sElect voting system is interesting because it is so simple, requiring no other primitives than digital signatures and public key encryption. Proving security properties for interesting voting systems is intrinsically interesting.

As we have seen, informal arguments sometimes contain errors. A proof formalised in EasyCrypt is significant, in that it ensures that we have no errors in arguments, making the overall security proof easier to verify.

In addition to the main contribution, we also make the relation between verifiability and accountability precise, in the sense that accountability implies verifiability (when suitably defined). This is a significant result, suggesting that future system designers should focus on achieving accountability.

1.2 Related Work

To the best of our knowledge, no game-based definition of accountability has been proposed earlier. However, several definitions of accountability (for general security protocols, not only electronic voting protocols) have been proposed in the symbolic model. Bruni *et al.* [5] propose a general definition amenable to automated verification. Künnemann *et al.* [16] give a definition of accountability in the decentralised-adversary setting, in which single protocol parties can choose to deviate from the protocol, while Künnemann *et al.* [15] give a definition in the single-adversary setting, where all deviating parties are controlled by a single, centralised adversary. Morio & Künnemann [19] combine the definition from [15] with the notion of case tests to extend the definition's applicability to protocols with an unbounded number of participants. Furthermore Küsters *et al.* [18] put forward quantitative measures of accountability both in the symbolic and computational model. Similar for all these definitions is that they clearly distinguish between *dishonest* parties and *misbehaving* parties. Even though a party is dishonest (controlled by an adversary), it does not necessarily deviate from the protocol and cause a violation of the security goal. In such cases, the party is not misbehaving and should not be held accountable for anything.

While no game-based definition of accountability has been proposed, game-based definitions for other voting-related security properties do exist in the literature. Some of these definitions have also been formalised in the proof assistant EasyCrypt [3], with related machine-checked proofs for a variety of voting protocols. Cortier *et al.* [6] formalise a game-based definition of ballot-privacy called BPRIV [4] in EasyCrypt and give a machine-checked proof that Labelled-MiniVoting [6] and several hundred variants of Helios [2] satisfy this notion of ballot privacy. Cortier *et al.* [7] build on work from [6] and also formalise a game-based definition of verifiability in EasyCrypt, in addition to giving a machine-checked proof that Belenios [9] is ballot-private and verifiable. Drăgan *et al.* [10] formalise the mb-BPRIV ballot privacy definition [8] in EasyCrypt and give a machine-checked proof that Labelled-MiniVoting and Belenios satisfy this definition. They also propose a new game-based ballot privacy definition called du-mb-BPRIV, which is applicable to schemes where voter verification

can or must happen after the election result has been computed, and give a machine-checked proof that Labelled-MiniVoting, Belenios and Selene [20] all satisfy this definition.

Problems in the Küsters et al. [17] Accountability Proof. In carefully analysing sElect we became aware of two errors in the Accountability theorem which we detail below; to our knowledge these errors have not previously been documented in the literature. There is a significant complexity in the parameters used in Theorem 3 (Accountability) in the full version of sElect [17], but fortunately this is largely orthogonal to the points we need to discuss.

Ballot Stuffing The goal for which accountability is proven (see Definition 1 in [17]) somewhat implicitly requires that the multiset containing the election result contains at most n elements, where n is the number of voters. However, no argument is made in the proof that the judge will hold anyone accountable if there are more than n ballots. Both the pen-and-paper description and the implementation of sElect omit any checks which would catch the addition of ballots by the mix servers, and it seems that the authentication server could also stuff ballots though this would be more involved. As significant as this vulnerability is, it is easy to fix and we have done so in the version of sElect we prove accountability for.

Honest Nonce Collision A described above, the goal the theorem aims for uses multisets and hence if multiple honest voters vote for the same choice we expect to see at least that many copies of the choice in the output; this is somewhat complicated in sElect by the augmentation of voter choices with nonces. The mechanism which sElect uses to detect ballots being removed relies on the plaintext encrypted by the honest voters being unique; however, this does not happen when the nonces and choices of the honest voters collide. The chance of such collision should appear in the security bound of accountability for sElect unless it is explicitly negligible in the security parameter. Strangely, sElect will drop these votes even with no adversarial involvement since the protocol specifies that the final mix server (like all others) should filter its output for duplicates. We note that the probability of collisions does appear in the verifiability theorem and proof.

2 Game Based Accountability

In this section we present our game based definition of accountability for electronic voting protocols, and we start by presenting the parties and their roles.

2.1 Parties

We consider the following parties and their role in the election process.

Voting Authority VA that sets up the election process, generates public parameters, defines voter eligibility, etc. The election secret keys are managed by separate parties, called decryption and mixnet authorities.

Decryption and Mixnet Authorities $\mathsf{MS}_i(\mathsf{msk}_i, \mathsf{mpk}_i)$ that manage together the decryption process, and each party has been allocated a part of the decryption key/election secret key. This is typically done by decryption or re-encryption together with shuffling of ballots/votes to break the link between recorded ballots and the votes.

Authentication Server $\mathsf{AS}(\mathsf{ask}, \mathsf{apk})$ issues confirmation tokens that ballots were recorded as cast, typically under the form of signatures.

Judge J assigns blame to misbehaving parties based on publicly available data and voter reported evidence. We model this by having an algorithm Judge.

Voters id_i that cast their vote v_i. The voting process is facilitated by a voting supporting device VSD that builds ballots for the user and then casts them.

Bulletin board BB stores publicly verifiable information relevant to an election, e.g. ballots, mixnet outcomes, and election outcome. The bulletin board may be divided into subcomponents such as a list of submitted ballots or the election outcome.

2.2 Voting System

The election process is defined by the following tuple of algorithms.

$\mathsf{Setup}()$**:** This algorithm produces the public election data pd and the secret election data sd. This is done by interaction between VA, $\mathsf{MS}_0, \ldots, \mathsf{MS}_k$, and potentially AS.

$\mathsf{Vote}(\mathsf{pd}, v)$**:** This algorithm builds the ballot b based on the vote v and public data pd. Additionally, it produces the internal state of the voter, state, to facilitate the verification process later.

$\mathsf{ASCreate}(\mathsf{ask}, b)$**:** This algorithm produces a token σ that the ballot b has been received and accepted by the authentication server $\mathsf{AS}(\mathsf{ask}, \mathsf{apk})$.

$\mathsf{ASVerify}(\mathsf{pd}, b, \sigma)$**:** This algorithm verifies if the token σ is valid for the ballot b and public data pd.

$\mathsf{Tally}(\mathsf{sd}, \mathsf{BB})$**:** This algorithm models the sequence of calls to the mixnet and decryption authorities to produce the election outcome.

$\mathsf{VSDVerify}((\mathsf{state}, b, \sigma), \mathsf{pd}, \mathsf{BB})$**:** Checks if the system has followed the required processes for this user's vote and ballot, and it outputs $\perp$ if no misbehaving party has been identified. Otherwise, it returns the misbehaving party and the corresponding evidence.

$\mathsf{Judge}(\mathsf{pd}, \mathsf{BB}, \mathsf{E})$**:** It checks that the publicly available data is valid with respect to some predefined metrics and against the list of evidence E. It returns the error symbol $\perp$ if no misbehaving party has been identified; otherwise, it outputs the misbehaving party B. As all checks can be replicated publicly, it does not need to return evidence.

The following algorithm is unbounded, but is only part of the security experiment and will not be run during an election.

$\mathsf{Bad}(\mathsf{pd}, \mathsf{BB}, \mathsf{E}, \mathsf{V})$**:** This unbounded algorithm serves to provide a ground truth of which parties misbehaved. By the requirements of our definition, it always

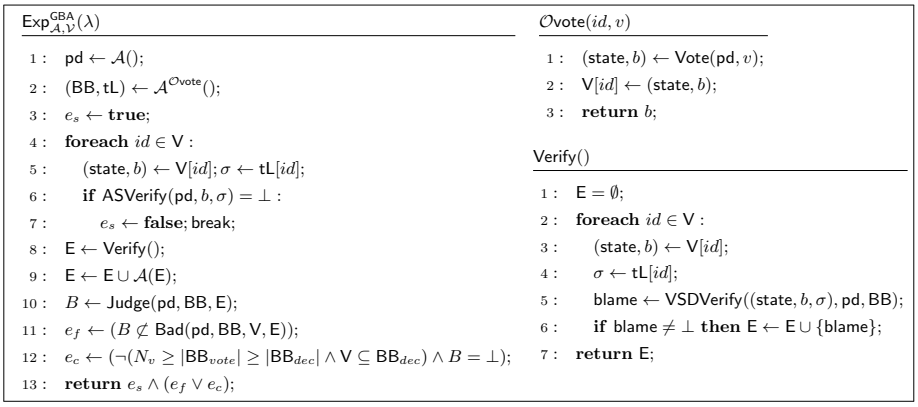

Fig. 1. The new game-based security notion for accountability. BB_{vote} and BB_{dec} denote different subcomponents of the bulletin board, respectively ballots submitted through $\mathcal{O}\mathsf{vote}$ and information produced by tallying.

blames a party when the election result does not reflect the votes of voters - given the public data pd, the bulletin board BB, the list of evidence E, and the internal state of honest voters V. Optionally, it may detect whether a party has deviated from the protocol in a way which does not change the election result. It should never blame an honest party.

2.3 Accountability

We consider that the adversary has full control over all parties introduced in Section 2.1, except the Judge. The adversary can also incorporate their own evidence to Judge. If a party deviates from the protocol steps, then that party becomes *misbehaving* and could be identified and blamed by either Judge or Bad. However, if the party follows exactly the protocol steps we call that party *behaving*, independent of them being honest or dishonest (corrupted by the adversary).

The formal accountability definition is found in Fig. 1. The first step for the adversary is to start the election process and provide the public data pd. Then, the adversary runs the voting and tally phase and commits to the current state of the bulletin board BB, together with a list of all authentication tokens tL. During the voting phase, the adversary can make use of the oracle $\mathcal{O}\mathsf{vote}$ to replicate the behavior of behaving voters and build their ballot b and internal state state.

To capture the natural behavior of behaving and honest voters that would check their tokens and complain before the tally is provided, we incorporate an automatic lose condition for the adversary if any of the provided tokens for those voters cannot be verified by ASVerify; this approach is similar to that taken by Küsters *et al.* [17] in their (non-game based) accountability proof of sElect.

Verification is done as a two-stage process, first by collecting evidence E from all honest voters (those that used $\mathcal{O}\mathsf{vote}$ and whose internal states are stored in

V) and from the adversary $\mathcal{A}(\mathsf{E})$; and secondary by calling Judge to check the public data together with that evidence. The Judge is responsible for providing either a misbehaving party B if there is enough evidence to do so, or $\perp$ if nothing could be detected. The adversary wins if one of the following happens:

Fairness: Judge wrongly blames a party B when it did not misbehave. This is checked by running the Bad algorithm to identify all misbehaving parties in the system and check whether B has been included, or

Completeness: the result is not consistent with the honest votes but no one is blamed. This is done by Judge producing $\perp$ when an honest voter's ballot was dropped, or there are more submitted ballots than there are voters, or more ballots in the election outcome than the number of ballots that were cast in the first place.

Definition 1 (Game-Based Accountability). *Let $\mathcal{V}$ be a voting system as defined in this section. We say that $\mathcal{V}$ satisfies* GBA *if for any efficient adversary $\mathcal{A}$ their advantage is negligible in λ:*

$$\mathsf{Adv}^{\mathsf{gba}}_{\mathcal{A},\mathcal{V}}(\lambda) = \Pr\left[\mathsf{Exp}^{\mathsf{GBA}}_{\mathcal{A},\mathcal{V}}(\lambda) = 1\right].$$

Our adversary winning conditions aligns our definition with the one from Küsters *et al.* [17], such that any voting system that satisfies our accountability definition will also satisfy the one by Küsters *et al.* [17] (with the goal used for sElect), with some possible caveats about the casting of schemes between the two definitions. We expand on this in Section 3.3

3 sElect

In this section we introduce sElect [17] using the format of Section 2; we focus on the elements with are important for accountability and omit some orthogonal details. The formal description is in Figure 2. We denote by $\mathsf{BB}_{vote}, \mathsf{BB}_{mix}$ and BB_{dec} the different subcomponents of BB: respectively the submitted ballots, data produced by the mixnet and the election outcome, which is a list of plaintext votes.

3.1 Cryptographic primitives

The voting system sElect relies on two basic cryptographic primitives: an IND-CCA2 encryption scheme $E = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ and an EU-CMA signature scheme $S = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{SigVerif})$. To make the encryption scheme compatible with decryption mixnets it needs to allow nested encryptions. Typically, this is done through a hybrid cryptosystem [1], by combining hybrid ElGamal and AES in a suitable mode such that each encryption contains an AES encryption of the message under a random AES key and an ElGamal encryption of the AES key.

As part of the formalisation for the shuffling done by the mixnet servers $\mathsf{MS}_0, \ldots, \mathsf{MS}_k$, we consider the operators lex for sorting a list in lexicographic order, and undup for removing duplicates. We additionally have that the authentication authority AS runs S.

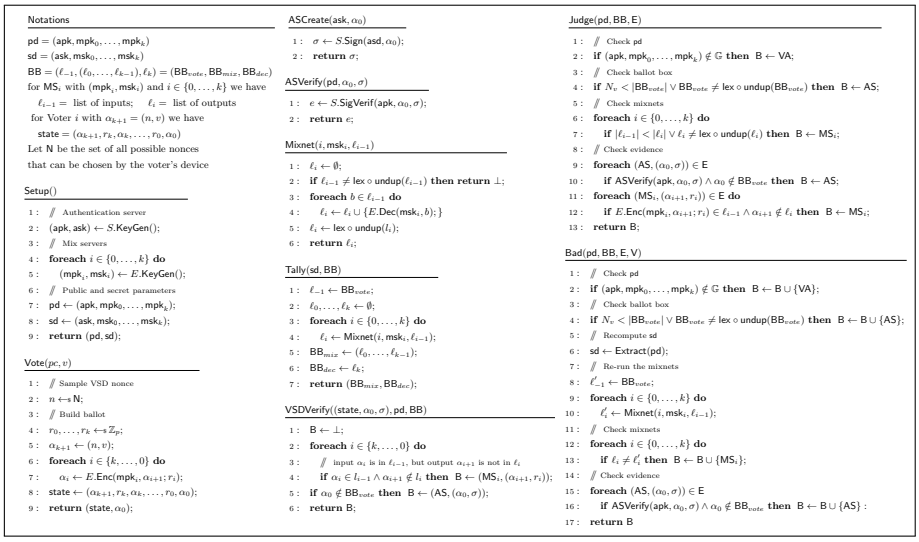

Fig. 2. Algorithms defining the sElect voting scheme with an IND-CCA2 secure public key encryption system $E = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ and an EU-CMA secure signature scheme $S = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{SigVerif})$.

3.2 sElect Algorithms

$\mathsf{Setup}()$**:** The authentication server key pair $(\mathsf{apk}, \mathsf{ask})$ is generated by $S.\mathsf{KeyGen}$, and the mixnet servers key pairs $(\mathsf{mpk}_i, \mathsf{msk}_i)$ are computed by $E.\mathsf{KeyGen}$. The algorithm returns the public data $\mathsf{pd} = (\mathsf{apk}, \mathsf{mpk}_0, \ldots, \mathsf{mpk}_k)$ and secret data $\mathsf{sd} = (\mathsf{ask}, \mathsf{msk}_0, \ldots, \mathsf{msk}_k)$.

$\mathsf{Vote}(\mathsf{pd}, v)$**:** The algorithm samples a supporting device verification code n such that it can be used later by the voter to ensure their vote was counted. sElect also considers a short voter verification code n_{voter} that has no security assumptions (for accountability); we have included that code together with the voter's candidate choices c as part of the vote $v = (n_{voter}, c)$. The algorithm sets $\alpha_{k+1} = (n, v)$ and uses a series of encryptions $\alpha_i \leftarrow \mathsf{Enc}(\mathsf{mpk}_i, \alpha_{i+1}, r_i)$ to build the ballot α_0 and internal state $\mathsf{state} = (\alpha_{k+1}, \alpha_k, r_k, \ldots, \alpha_0, r_0)$, given some random coins $r_0, \ldots, r_k \in \mathbb{Z}_p$.

$\mathsf{ASCreate}(\mathsf{asd}, \alpha_0)$**:** It returns a signature σ by calling $S.\mathsf{Sign}$ over the ballot α_0.

$\mathsf{ASVerify}(\mathsf{pd}, \alpha_0, \sigma)$**:** This algorithm calls $S.\mathsf{SigVerif}$ to check if the signature σ is valid for the ballot α_0.

$\mathsf{Tally}(\mathsf{sd}, \mathsf{BB})$**:** Given the ballot box $\ell_{-1} = \mathsf{BB}_{vote}$, this algorithm runs each mixnet MS_i over ℓ_{i-1} to produce ℓ_i, for $i \in \{0, \ldots, k\}$. Each mixnet MS_i, ensures first that the inputs are in lexicographic order and contain no duplicates, before decrypting all ciphertexts received as inputs, and finally outputting them in lexicographic order and without duplicates. The last mixnet server produces the election outcome $\mathsf{BB}_{dec} = \ell_k$. The algorithm returns the mixing info $\mathsf{BB}_{mix} = (\ell_0, \ldots, \ell_{k-1})$ and election outcome BB_{dec}.

$\mathsf{VSDVerify}((\mathsf{state}, \alpha_0, \sigma), \mathsf{pd}, \mathsf{BB})$**:** Voters check the output of each mixnet server by using their internal state $(\alpha_{k+1}, \alpha_k, r_k, \ldots, \alpha_0, r_0)$. The voter blames a mixnet server MS_i if they see that their ciphertext α_{i+1} is in the input list of that server, but the ciphertext α_i is not in the output list. Recall that α_i has been created by encrypting α_{i+1} under that server's public key mpk_i: $\alpha_i \leftarrow \mathsf{Enc}(\mathsf{mpk}_i, \alpha_{i+1}, r_i)$; thus, (α_{i+1}, r_i) can be used as evidence of misbehaviour of MS_i. The voter also checks that their ballot α_0 has been included in the ballot box BB_{vote}, and blames the authentication server AS if that has not happened, using the signature σ the user received during voting as evidence. This step can be done at any point in the election if one considers an ideal bulletin board, or at the end of an election under weaker trust assumptions over the bulletin board [11].

$\mathsf{Judge}(\mathsf{pd}, \mathsf{BB}, \mathsf{E})$**:** This algorithm does an initial round of checks over the public data before evaluating the collected evidence E. The verification of public data consists of

- Ensuring that the public data is valid - that is, the public keys are group elements. If this is not true, then the voting authority VA is blamed as it allowed the election to run.
- Checking that the size of the ballot box does not exceed the number of voters and that the ballot box has been ordered lexicographically and duplicates have been removed. Otherwise, the authentication server AS is blamed.
- Checking that each mixnet server output is in lexicographic order and has no duplicates, and that the size of the output list does not exceed the size of the input list. If these properties do not hold for mixnet server MS_i then the algorithm blames this mixnet server.

Once all the public data has been verified, the algorithm looks at the evidence collected by voters from their $\mathsf{VSDVerify}$ algorithm:

- Evidence (α_0, σ) against AS. If the evidence contains a valid signature σ for a ballot α_0 not in the ballot box, then the authentication server AS is blamed.
- Evidence (α_{i+1}, r_i) against MS_i. If the evidence shows that $\alpha_i \leftarrow \mathsf{Enc}(\mathsf{mpk}_i, \alpha_{i+1}, r_i)$ is in the input list of this server, but α_{i+1} is not in the output, then this mixnet server is blamed.

$\mathsf{Bad}(\mathsf{pd}, \mathsf{BB}, \mathsf{E}, \mathsf{V})$**:** This algorithm uses a computationally unbounded algorithm $\mathsf{sd} \leftarrow \mathsf{Extract}(\mathsf{pd})$ to obtain the secret keys of all authorities sd from their public data pd; similar to the vote extraction algorithm from [13] and [14]. $\mathsf{Extract}$ will never fail to return something as it will see any bitstring in the public data as a group element. However, it may not produce meaningful data or the real secret keys if these do not exist.

Bad uses the secret data from $\mathsf{Extract}$ to re-run the election tally and perform all verification steps to identify misbehaving parties. It looks at the validity of the public data pd and ballot box BB_{vote} using the same methods employed by Judge. Then, it re-creates for each mixnet server MS_i its estimated output ℓ_i' and blames that party if their estimated output ℓ_i' is different from the declared output ℓ_i. This type of check already includes the checks on the

evidence submitted by voters against mixnet servers. Finally, it performs the checks on the evidence against the authentication server AS.

3.3 EasyCrypt Proof

Informally, we prove that the probability that the adversary is able to produce valid public data, a valid bulletin board and valid signatures, while at the same time violating either fairness or completeness, is negligible. We assume throughout the proof that the public key encryption scheme used to encrypt and decrypt ballots is perfectly correct, i.e. if we let $E = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be the (IND-CCA2 secure) PKE used in sElect, then we assume that for all key pairs $(\mathsf{pk}, \mathsf{sk})$ output by KeyGen and for all plaintexts m in the message space, we have $\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = m$. As we assume that sElect is implemented with hybrid encryption of ElGamal and AES (cf. Section 3.1), this assumption holds. Under this assumption, the probability that the adversary violates the fairness aspect of accountability is in fact 0. The probability that the adversary violates the completeness aspect of accountability, is related to whether or not *nonce collisions* occur, i.e. whether or not the devices of two or more honest voters sample the same nonce.

Theorem 1. *Let* $\mathsf{sElect}(E, S)$ *be defined as in Figure 2 for an IND-CCA2 encryption scheme* E *and an EU-CMA signature scheme* S*. Then, for all PPT adversaries* $\mathcal{A}$ *against* GBA*, we have*

$$\mathsf{Adv}^{\mathsf{gba}}_{\mathcal{A},\mathsf{sElect}}(\lambda) \leq \Pr[Col]\,,$$

where Col is the event that a collision occurs in the nonces chosen by the voters' devices.

The proof sketch can be found in App. A.

Differences between our paper proof and EasyCrypt *proof.* The main difference in the above proof and the proof formalised in EasyCrypt[1] is that in EasyCrypt we let the adversary choose both the plaintext vote and the verification nonce and compress this into a single plaintext. Under the assumption that the choices made by the adversary are unique, this allows us to use sets rather than multisets in EasyCrypt which is technically easier. In some sense this is however also a stronger result than above since it proves accountability even in the case where the nonces are adversarially chosen, but still unique. What is not proven in EasyCrypt is the probability of a collision happening, but for uniform distributions of nonces this is the well-known birthday paradox which is not interesting for the present paper to verify in EasyCrypt. Finally, keeping a general collision probability for the full plaintext consisting of device-generated nonce, voter-chosen nonce and plaintext vote is more general, and cannot be assumed to be uniformly random in practice, but can be bounded by the birthday probability on the device-generated nonces.

[1] The EasyCrypt code can be accessed from `https://github.com/mortensol/acc-select`

4 Relation to the Küsters et al. definition

In this section we relate the above presented definition of accountability, GBA, to the one by Küsters et al. [18], which we denote Acc_{KTV}. More precisely, we sketch a proof that for the class of voting schemes expressible in our definition, if they satisfy GBA for a certain definition of Bad then they must be accountable under Acc_{KTV} with a standard goal.

Consider a voting scheme as defined in Section 2, consisting of a voting authority VA, decryption authorities DA, mixnet authorities MS, authentication server AS, voters id_i with voter supporting devices VSD_i, and bulletin board BB. We assume there are authenticated channels from the VSDs to the AS. We assume that each VSD has one authenticated and one anonymous channel to the BB. We assume that all communication is authenticated with signatures with the exception of the anonymous channel and for simplicity omit the description of this occurring from the exposition below.

4.1 Modeling

A voting scheme of this kind can be modeled in the framework of [18] in a straightforward way as a protocol $\mathcal{P}(n, m, q, \mu, p^{verif}_{voter}, p^{verif}_{abst})$. We refer to [18] for the notation used. We denote by n the number of voters and supporting devices, by m the number of mix servers, by q the number of decryption servers. By μ we denote the probability distribution on the set of candidates/choices, including abstention. We denote by p^{verif}_{voter} and p^{verif}_{abst} the probability that the voting voter will verify and an absenting voter will verify respectively.[2]

We define Φ_k as the accountability property consisting of the constraints:

$$\chi_i \rightarrow dis(id_i) \vee dis(AS), \quad \chi'_i \rightarrow dis(id_i) \vee dis(AS)$$
$$\neg\gamma_k \wedge \neg\chi \rightarrow dis(\mathsf{VA}) | dis(\mathsf{AS}) | dis(\mathsf{DA}_i)^q_{i=1} | dis(\mathsf{MS}_j)^m_{j=1}$$

where

γ_k contains all runs of the protocol where at most n votes are in the result and where at most k of the honest votes are not included in the result. See [17] for a formal definition and discussion of this goal.

χ_i contains all the runs of $\mathcal{P}$ where the voter i complains they did not get a receipt.

χ'_i contains all the runs of $\mathcal{P}$ where the voter i complains they did not vote but a vote was cast on their behalf.

χ contains the union of all runs in χ_i and χ'_i for all $i \in [1, ..., n]$

[2] Absenting voters verify that their identifier is not included on the list published by the AS.

4.2 Result

Let Bad be defined as follows: Bad returns all parties whose output is not in the co-domain of the honest algorithms. When parties are called multiple times on different algorithms and pass states, we take the co-domain over all possible states consistent with their early public output.

Let the Judge_{KTV} algorithm for Acc_{KTV} in [18] be constructed as follows:

(J1) first it runs Judge (from our definition) and if this outputs blame, then Judge_{KTV} blames the party returned by Judge.

(J2) If no valid complaints were made by the voters causing blame, the judge checks the complaints posted by the voters. If there is any such complaint then Judge_{KTV} blames (disjunctively) both the party accused and the voter accusing.

Definition 2 (Voter Verification Correct). *For a scheme π we say that it is voter verification correct if for all runs of the protocol the party blamed by* VSDVerify *is in the set output by* Bad *or it blames the* AS *after receiving an invalid confirmation.*

Theorem 2 (GBA implies Acc_{KTV}). *Let the judge Judge_{KTV} and algorithm* Bad *be defined as above. Then for any scheme which has* GBA *and voter verification correctness, Judge_{KTV} ensures $\left(\Phi_k, \delta^k(p^{verif}_{voter}, p^{verif}_{abst})\right)$-accountability for $\mathcal{P}(n, m, q, \mu, p^{verif}_{voter}, p^{verif}_{abst})$ where*

$$\delta^k(p^{verif}_{voter}, p^{verif}_{abst}) = (1 - \min(p^{verif}_{voter}, p^{verif}_{abst}))^{k+1}.$$

Due to space constraints, we detail this in App. B. The proof relies on analysing fairness and completeness for the two definitions.

5 Verifiability

In this section we show that our definition of accountability implies verifiability; a relation already shown in the framework of Küsters et al [18]. To prove this implication here, we introduce a new game-based definition of verifiability, that we formalize via the experiment $\mathsf{Exp}^{Ver}_{\mathcal{A}}(\lambda)$ in Fig. 3.[3] Our definition of verifiability ensures individual verifiability and no ballot stuffing during tally, and is appropriate for lightweight voting systems like sElect. Our definition is modular, and can be enhanced to model stronger notions of verifiability (e.g., universal verifiability or no ballot stuffing at submission time); however, to achieve them voting systems will require heavier cryptographic primitives, likes zero-knowledge proofs for correct tallying or shuffling.

We consider $\mathcal{I}$ the set of eligible voter IDs, and we introduce algorithm VoterVerif that enables voters to verify their vote. We keep track of voters that successfully verified using the set Checked and we raise the flag Complain when

[3] In the game, we use the notation "Require" for **if** $\cdots$ **else return** $\perp$.

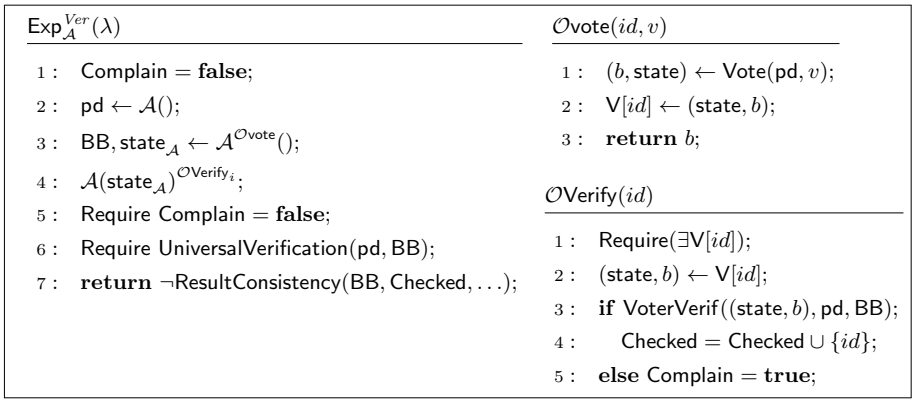

Fig. 3. Verifiability assuming uncorrupted vote-casting.

verification fails. The adversary can choose which voters verify via the oracle $\mathcal{O}\mathsf{Verify}(id)$. Additionally, the adversary uses the vote oracle $\mathcal{O}\mathsf{Vote}$ to model honest voters (re-)casting their votes; we focus on the last vote counts policy, but this can easily be generalized for any policies.

The adversary also controls the bulletin board BB, however anyone can perform $\mathsf{UniversalVerification}(\mathsf{pd}, \mathsf{BB})$ to universally verify this state. We further use $\mathsf{ResultConsistency}(\mathsf{BB}, \mathsf{Checked}, \ldots)$ to model the consistency relations on the bulletin board, and can depend on the different subcomponents of BB: list of submitted ballots $\mathsf{BB}|_{submit}$, the election result $\mathsf{BB}|_{res}$ and extra info $\mathsf{BB}|_{extra}$.

Consider the election result function $\rho : \mathsf{Cand}^* \mapsto \mathsf{Res}$ as a symmetric function from the set of plaintext votes, chosen from the space of candidates Cand, to a given result set Res. Using $\mathsf{V}[S]$ the corresponding list of plaintext votes from the vote oracle, we model

- Individual Verifiability: Intuitively this should ensure that the verified votes are all included in the tally. Using the verification oracles $\mathcal{O}\mathsf{Verify}_i, i = 1, \ldots, k$ we denote the successful verifiers $\mathsf{Checked}$. The constraint from $\mathsf{ResultConsistency}$ is $\exists v_1, \ldots, v_i \in \mathsf{Cand}, i + |\mathsf{Checked}| \leq |\mathcal{I}|$:

$$\rho(v_1, \ldots, v_i, \mathsf{V}[\mathsf{Checked}]) = \mathsf{BB}|_{res}$$

  where we have slightly abused notation for readability. We have included a constraint on the number of malicious votes since if the result function allows cancelling votes the inclusion of the honest votes would make little sense if the adversary can add malicious votes arbitrarily.
- No Ballot Stuffing at Tally Time: $|\mathcal{I}| \geq |\mathsf{BB}|_{submit}|$ and $\exists i \leq |\mathsf{BB}|_{submit}|$ $\exists v_1, \ldots, v_i \in \mathsf{Cand} : \rho(v_1, \ldots, v_i) = \mathsf{BB}|_{res}$, i.e. there is at most as many submitted ballots as eligible voters and the result is consistent with a number of votes that is less than or equal to the submitted ballots.

In the case of schemes where all the decrypted votes are displayed individually in $\mathsf{BB}|_{res}$, especially this holds for the mixnet-tally schemes, the slightly stronger

statement can be made that

$$|\mathcal{I}| \geq |\mathsf{BB}|_{submit}| \geq |\mathsf{BB}|_{res}| \wedge \mathsf{V}[\mathsf{Checked}] \subseteq_{ms} \mathsf{BB}|_{res}\,, \tag{1}$$

where we use $\mathsf{V}[\mathsf{Checked}]$ and $\mathsf{BB}|_{res}$ as multisets.

We define verifiability given a chosen ResultConsistency if any efficient adversary has negligible advantage in $\mathsf{Exp}^{Ver}_{\mathcal{A}}(\lambda)$. In particular, we define verifiability for voting systems with the result being the plaintext votes as:

Definition 3. *We say that a voting system $\mathcal{V}$, with result function being the set of votes, satisfies individual verifiability and no ballot stuffing at tally time if for any efficient adversary $\mathcal{A}$ their advantage $\mathsf{Adv}^{ver}_{\mathcal{A},\mathcal{V}}(\lambda) = \mathsf{Exp}^{Ver}_{\mathcal{A},\mathcal{V}}(\lambda)$ is negligible in λ, where ResultConsistency checks Eq. 1.*

We note that there are some verifiability properties that sElect does not fulfill but could be easily captured by the ResultConsistency or separate games, namely

- Tally Uniqueness: The adversary cannot produce two boards both satisfying UniversalVerification and individual verifications but with different tally results and having the same submitted ballots $\mathsf{BB}|_{submit}$.
- Universal Verifiability: Here ResultConsistency requires that the result is the same as the result from votes extracted from the valid ballots in $\mathsf{BB}|_{submit}$ given only that the board satisfies $\mathsf{UniversalVerification}(\mathsf{pd}, \mathsf{BB})$.

5.1 Accountability implies Verifiability

We will now prove that the GBA accountability definition implies verifiability for individual verifiability and no ballot stuffing as defined in Def. 3. However, in order to do so, we need to relate the Judge and the VSDVerify algorithms used in $\mathsf{Exp}^{\mathsf{GBA}}_{\mathcal{A},\mathcal{V}}(\lambda)$ with the algorithms UniversalVerification and VoterVerif used in $\mathsf{Exp}^{Ver}_{\mathcal{A},\mathcal{V}}(\lambda)$. Especially, the verifiability definition does not consider the authentication server AS and its signatures, since it is not relevant for defining verifiability. To this end we make the following definition for a voting system $\mathcal{V}$ fitting both the accountability and the verifiability framework:

Definition 4. *We call a voting system $\mathcal{V}$* accountability-verifiability-correct *if the signature part for AS is an independent part that can be removed to give a reduced system valid for the verifiability framework, or correspondingly added. Further, the Judge will never output blame if all verification checks by the verifying voters using VSDVerify does not output blame and $\mathsf{UniversalVerification} = \top$. Further, $\mathsf{VSDVerify}((\mathsf{state}, b, \sigma), \mathsf{pd}, \mathsf{BB}_{vote}, \mathsf{BB}_{mix}, \mathsf{BB}_{dec})$ will not output blame if $\mathsf{ASVerify}(\mathsf{pd}, b, \sigma) = \top$ and $\mathsf{VoterVerif}((\mathsf{state}, b), \mathsf{pd}, \mathsf{BB}) = \top$.*

Theorem 3. *Given an accountability-verifiability-correct voting system, then accountability as defined in Def. 1 implies individual verifiability and no ballot stuffing at tally time as defined in Def. 3 assuming the AS signature scheme is perfectly correct and we have a constant number of voters. More precisely for any efficient adversary $\mathcal{A}$ against $\mathsf{Exp}^{Ver}_{\mathcal{A},\mathcal{V}_r}(\lambda)$ with advantage $\mathsf{Adv}^{ver}_{\mathcal{A},\mathcal{V}_r}(\lambda)$ in the reduced system $\mathcal{V}_r$ without signatures, we can construct an adversary $\mathcal{B}$ against $\mathsf{Exp}^{\mathsf{GBA}}_{\mathcal{A},\mathcal{V}}(\lambda)$ with advantage at least $\frac{1}{2^{|\mathcal{I}|}}\mathsf{Adv}^{ver}_{\mathcal{A},\mathcal{V}_r}(\lambda)$.*

Due to space constraints the full proof is in App. C.

It follows from Thm. 1 and Thm. 3 that sElect fulfills individual verifiability and no ballot stuffing at tally time as defined in Def. 3.

6 Concluding Remarks

We study notions of accountability for electronic voting, and produce the first *game-based* notion of accountability for mix-based electronic voting schemes. We relate our notion to Küsters et al's quantitative notion, arguing that they coincide at the extremes of the parameter range.

We demonstrate the value of such a game-based notion by formalising it in EasyCrypt, and produce a machine-checked proof of accountability—as we define it—for Küsters et al.'s sElect protocol, discussing issues with previous accountability results for sElect as we go. Finally, we use our new game-based definition of accountability to study the relationship between accountability, verifiability, demonstrating in particular that accountability implies verifiability.

Generalisation beyond sElect We framed our discussions, and our definitions, with sElect. However, our definitions would also somewhat trivially apply to other voting schemes. In particular, as mentioned in the introduction, any scheme making judicious use of sound zero-knowledge proofs for verifiability can be trivially argued to be accountable: an adversary who is able to break accountability with sound zero-knowledge proofs does so either by breaking soundness of the zero-knowledge proof systems, or by breaking accountability of a scheme in which verification for the zero-knowledge proofs is idealised to reject any proof that was not produced as is by the prover—relying then only on the correctness of the encryption scheme as in our sElect proof. Although this argument is easy to make on paper, formalising it in EasyCrypt on existing formal definitions for Helios (for example) would involve effort incommensurate to its scientific value as part of this specific paper.

Beyond Accountability Capturing accountability as a game-based notion is not just useful to allow a more precise analysis of accountability. By doing so, we hope to open the way to the study of privacy and security properties of voting schemes *with dispute resolution*. Formally taking into account dispute resolution requires a precise understanding of the individual and overall guarantees offered by verifiability in terms of the accuracy of the election result.

Acknowledgment T. Haines is the recipient of an Australian Research Council Australian Discovery Early Career Award (project number DE220100595). C. C. Drăgan is supported by EPSRC grant EP/W032473/1 (AP4L), EU Horizon grants 101069688(CONNECT) and 101070627 (REWIRE). P. Rønne received funding from the France 2030 program managed by the French National Research Agency under grant agreement No. ANR-22-PECY-0006.

References

1. Abdalla, M., Bellare, M., Rogaway, P.: DHIES: An encryption scheme based on the Diffie-Hellman problem. Contributions to IEEE P1363a (Sep 1998)
2. Adida, B.: Helios: Web-based open-audit voting. In: van Oorschot, P.C. (ed.) USENIX Security 2008. pp. 335–348. USENIX Association (Jul / Aug 2008)
3. Barthe, G., Dupressoir, F., Grégoire, B., Kunz, C., Schmidt, B., Strub, P.Y.: EasyCrypt: A Tutorial, pp. 146–166. Springer International Publishing, Cham (2014). https://doi.org/10.1007/978-3-319-10082-1˙6, `https://doi.org/10.1007/978-3-319-10082-1_6`
4. Bernhard, D., Cortier, V., Galindo, D., Pereira, O., Warinschi, B.: A comprehensive analysis of game-based ballot privacy definitions. Cryptology ePrint Archive, Report 2015/255 (2015), `https://eprint.iacr.org/2015/255`
5. Bruni, A., Giustolisi, R., Schürmann, C.: Automated analysis of accountability. In: Nguyen, P.Q., Zhou, J. (eds.) ISC 2017. LNCS, vol. 10599, pp. 417–434. Springer, Heidelberg (Nov 2017)
6. Cortier, V., Dragan, C.C., Dupressoir, F., Schmidt, B., Strub, P.Y., Warinschi, B.: Machine-checked proofs of privacy for electronic voting protocols. In: 2017 IEEE Symposium on Security and Privacy. pp. 993–1008. IEEE Computer Society Press (May 2017). https://doi.org/10.1109/SP.2017.28
7. Cortier, V., Dragan, C.C., Dupressoir, F., Warinschi, B.: Machine-checked proofs for electronic voting: Privacy and verifiability for belenios. In: Chong, S., Delaune, S. (eds.) CSF 2018 Computer Security Foundations Symposium. pp. 298–312. IEEE Computer Society Press (2018). https://doi.org/10.1109/CSF.2018.00029
8. Cortier, V., Lallemand, J., Warinschi, B.: Fifty shades of ballot privacy: Privacy against a malicious board. In: Jia, L., Küsters, R. (eds.) CSF 2020 Computer Security Foundations Symposium. pp. 17–32. IEEE Computer Society Press (2020). https://doi.org/10.1109/CSF49147.2020.00010
9. Cortier, V., Gaudry, P., Glondu, S.: Belenios: A Simple Private and Verifiable Electronic Voting System, pp. 214–238 (04 2019). https://doi.org/10.1007/978-3-030-19052-1˙14
10. Drăgan, C.C., Dupressoir, F., Estaji, E., Gjøsteen, K., Haines, T., Ryan, P.Y., Rønne, P.B., Solberg, M.R.: Machine-checked proofs of privacy against malicious boards for selene & co. In: 2022 IEEE 35th Computer Security Foundations Symposium (CSF). pp. 335–347 (2022). https://doi.org/10.1109/CSF54842.2022.9919663
11. Hirschi, L., Schmid, L., Basin, D.A.: Fixing the achilles heel of E-voting: The bulletin board. In: Küsters, R., Naumann, D. (eds.) CSF 2021 Computer Security Foundations Symposium. pp. 1–17. IEEE Computer Society Press (2021). https://doi.org/10.1109/CSF51468.2021.00016
12. Khazaei, S., Moran, T., Wikström, D.: A mix-net from any CCA2 secure cryptosystem. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 607–625. Springer, Heidelberg (Dec 2012). https://doi.org/10.1007/978-3-642-34961-4˙37
13. Kiayias, A., Zacharias, T., Zhang, B.: End-to-end verifiable elections in the standard model. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part II. LNCS, vol. 9057, pp. 468–498. Springer, Heidelberg (Apr 2015). https://doi.org/10.1007/978-3-662-46803-6˙16
14. Kiayias, A., Zacharias, T., Zhang, B.: Ceremonies for end-to-end verifiable elections. In: Fehr, S. (ed.) PKC 2017, Part II. LNCS, vol. 10175, pp. 305–334. Springer, Heidelberg (Mar 2017). https://doi.org/10.1007/978-3-662-54388-7˙11

15. Künnemann, R., Esiyok, I., Backes, M.: Automated verification of accountability in security protocols. In: Delaune, S., Jia, L. (eds.) CSF 2019 Computer Security Foundations Symposium. pp. 397–413. IEEE Computer Society Press (2019). https://doi.org/10.1109/CSF.2019.00034
16. Künnemann, R., Garg, D., Backes, M.: Accountability in the decentralised-adversary setting. In: Küsters, R., Naumann, D. (eds.) CSF 2021 Computer Security Foundations Symposium. pp. 1–16. IEEE Computer Society Press (2021). https://doi.org/10.1109/CSF51468.2021.00007
17. Küsters, R., Müller, J., Scapin, E., Truderung, T.: sElect: A lightweight verifiable remote voting system. In: Hicks, M., Köpf, B. (eds.) CSF 2016 Computer Security Foundations Symposium. pp. 341–354. IEEE Computer Society Press (2016). https://doi.org/10.1109/CSF.2016.31
18. Küsters, R., Truderung, T., Vogt, A.: Accountability: definition and relationship to verifiability. In: Al-Shaer, E., Keromytis, A.D., Shmatikov, V. (eds.) ACM CCS 2010. pp. 526–535. ACM Press (Oct 2010). https://doi.org/10.1145/1866307.1866366
19. Morio, K., Künnemann, R.: Verifying accountability for unbounded sets of participants. In: Küsters, R., Naumann, D. (eds.) CSF 2021 Computer Security Foundations Symposium. pp. 1–16. IEEE Computer Society Press (2021). https://doi.org/10.1109/CSF51468.2021.00032
20. Ryan, P.Y.A., Rønne, P.B., Iovino, V.: Selene: Voting with transparent verifiability and coercion-mitigation. In: Clark, J., Meiklejohn, S., Ryan, P.Y.A., Wallach, D.S., Brenner, M., Rohloff, K. (eds.) FC 2016 Workshops. LNCS, vol. 9604, pp. 176–192. Springer, Heidelberg (Feb 2016). https://doi.org/10.1007/978-3-662-53357-4˙12
21. SwissPost: Swiss post voting system. https://gitlab.com/swisspost-evoting (2022)

Appendix A Sketch of Proof of Theorem 1

We now sketch the proof of Theorem 1. We begin by defining two new games: a fairness game G_f and a completeness game G_c. These games are almost identical to the original security game, with the exception that in G_f, we remove the variable e_c from the experiment and only consider the fairness aspect of accountability, while in G_c, we remove the variable e_f and only consider the completeness aspect of accountability. Let E_f resp. E_c be the event that the game G_f resp. G_c returns 1. It is straightforward to see that $\Pr\left[\mathsf{Exp}^{\mathsf{GBA}}_{\mathcal{A},\mathsf{sElect}}(\lambda) = 1\right] \leq \Pr[E_f] + \Pr[E_c]$. Thus, the adversary has two possibilities to win. Either Judge has blamed an innocent party, or it has blamed no one, but the result is inconsistent with the honest votes. We analyze the fairness and completeness aspects separately, and argue that the adversary has zero probability of winning the fairness game and negligible probability of winning the completeness game.

We begin with fairness. We will consider each way in which the judge may blame a party and show that it will never blame a party that did not misbehave. Recall the various checks performed by the judge: The judge first checks that the public keys used by the authentication server and the mix servers are valid. If not, it will blame the voting authority as it allowed the election to run with invalid keys. Note that Bad will also blame the voting authority if the keys are invalid (but not otherwise), meaning that the voting authority will only be

blamed by the judge if it indeed misbehaved. As invalid keys will result in the voting authority being blamed by both the judge and by Bad, we assume for the remainder of the proof that all the public keys are valid.

The judge then checks that the bulletin board BB_{vote} is valid, i.e. that it contains at most N_v elements and that its contents are in lexicographic order and duplicate-free. If this check fails, the judge blames the authentication server. If this is the case, the authentication server will also be blamed by Bad, ensuring that if AS is blamed for producing an invalid board, it must indeed have misbehaved. Next, the judge checks that the output of each mix server contains at most as many elements as in its received input and that the output of each mix server is duplicate free and in lexicographic order. Since an honest mixer filters out duplicates and sorts the output list, it will always pass this check.

The judge then checks, for all ciphertext and signature pairs in the evidence list whether or not there is a ballot with a valid signature that is not present on BB_{vote}. Since an honest authentication server only authenticates the first ballot from each voter, and posts all these on the bulletin board, it will always pass this check. Note that if a dishonest voter blames the authentication server with valid evidence, the Bad algorithm will also blame the authentication server, and thus the judge will not blame the authentication server unless it is also blamed by Bad. Finally, Judge checks, for any triple $(\mathsf{mpk}_i, \alpha_{i+1}, r_i)$, whether or not $\mathsf{Enc}(\mathsf{mpk}_i, \alpha_{i+1}; r_i)$ is in the input to the ith mix server, but α_{i+1} is not in its output. Since the encryption system is correct, $\mathsf{Enc}(\mathsf{mpk}_i, \alpha_{i+1}; r_i)$ will decrypt to α_{i+1} and since an honest mix server does not remove any ciphertexts other than duplicates, it will always pass this check. In summary, Judge will never blame an honestly behaving party, and thus, the adversary has zero probability of winning the fairness game.

We now move on to completeness and bound the adversarial advantage in the completeness game, i.e. that if extra ballots are added or honest voters' ballots are dropped, the judge will, with overwhelming probability, hold someone accountable. Fairness ensures that the blamed party actually misbehaved.

We begin with the first criterion for completeness, i.e. that the number of ballots on BB_{vote} is not greater than the number of eligible voters. This follows from the second check of the Judge algorithm, where it checks if the bulletin board is valid. The second criterion (that the number of votes on BB_{dec} is not greater than the number of cast ballots) follows from the judge checking that the output of each mix server contains at most as many elements as its input.

Now consider the criterion that says that all honest votes are in the multiset of votes output by the last mix server (i.e. BB_{dec}). Every honest voter checks, using $\mathsf{VSDVerify}$, that their ballot appears in BB_{vote}. If not there, they output the token σ given to them by the authentication server. This, in turn, causes the authentication server to be blamed by the judge. If AS was not blamed, we know that all honest ballots were present in BB_{vote}. If any honest ballot is dropped by one of the mix servers, this will be detected by $\mathsf{VSDVerify}$, which will output some evidence that this mix server misbehaved, which in turn causes this mix server to be blamed by the judge.

Now, the adversary has one possibility of winning the completeness game, namely if two (or more) voters have cast the same vote, and their sampled nonces happen to be equal. In this case, the adversary may drop all but one of these ballots without it being detected. To analyze this situation, we slightly modify the completeness game. We call the new game G'_c and let E'_c be the probability that G'_c returns 1. The difference from G_c to G'_c is that in G'_c, we keep track of the nonces that are sampled when the adversary calls the vote oracle, and only sample new nonces from the set of nonces that have not been used earlier. The two games are equivalent unless there is a collision in the first game, hence $|\Pr[E_c] - \Pr[E'_c]| \leq \Pr[Col]$.

In G'_c, as there are no collisions in the nonces, any ballot that is dropped by the adversary will be detected by $\mathsf{VSDVerify}$, which in turns causes the judge to blame the misbehaving party. In other words, in G'_c, the adversary will have zero probability of winning, so $\Pr[E'_c] = 0$. Thus, the probability that the adversary wins the completeness game is bounded by $\Pr[Col]$. As the adversary has zero probability of winning the fairness game, and the probability of winning the accountability game is bounded by the sum of winning the fairness game and the completeness game, we arrive at the conclusion of Theorem 1 that the advantage is bounded by the collision probability. By the birthday paradox the collission probability is bounded by $\frac{q_v(q_v-1)}{2\cdot|\mathsf{N}|}$, where q_v is the number of queries vote oracle queries and N is the nonce space.

Appendix B Proof for Theorem 2

The proof of the theorem follows from analyzing Fairness and Completeness.

Lemma 1 (Fairness). *The judge J is computationally fair in $\mathcal{P}(n, m, \mu, p^{verif}_{voter}, p^{verif}_{abst})$.*

Proof. The proof is essentially the same as for sElect in [17] for the voting phase but relies on GBA in the mixing and decryption phases.

Consider what happens if the voter makes a complaint and the judge blames both the party accused and the voter (J2). Since the bulletin board is honest and the channel is authenticated the voter must really have made the complaint. There are two cases. If the voter is dishonest the verdict is clearly true. If the voter is honest, the correctness of the verdict follows from the voter verification correctness of the protocol either because the person it blamed has misbehaved or because the authentication server did not send a valid confirmation. (J2) covers both the case where the voter's ballot is dropped and when it is added.

Case (J1) is covered by GBA. Since the scheme has GBA it follows that the adversary cannot make either of these conditions trigger when the party ran its honest program, otherwise GBA would not hold.

Lemma 2 (Completeness). *For every instance π of $\mathcal{P}(n, m, \mu, p^{verif}_{voter}, p^{verif}_{abst})$, we have*

$$Pr\left[\pi(1^l) \to \neg(J : \Phi_k)\right] \leq \delta^k(p^{verif}_{voter}, p^{verif}_{abst}) = \left(1 - \min\left(p^{verif}_{voter}, p^{verif}_{abst}\right)\right)^{k+1}$$

with overwhelming probability as a function of l.

Again, the proof is essentially the same as for sElect in the voting phase but relies on GBA in the mixing and decryption phases. We need to show that the following probabilities are bounded for every i: a) $Pr\big[\pi(1^l) \rightarrow (\chi_i \wedge \neg dis(v_i) \wedge \neg dis(AS))\big]$, b) $Pr\big[\pi(1^l) \rightarrow (\chi'_i \wedge \neg dis(v_i) \wedge \neg dis(AS))\big]$, c) $Pr\big[\pi(1^l) \rightarrow (\neg\gamma_k \wedge \neg\chi \rightarrow dis(\mathsf{VA})|dis(\mathsf{AS})|dis(\mathsf{DA}_i)_{i=1}^{q}|dis(\mathsf{MS}_j)_{j=1}^{m})\big]$. The first two probabilities are equal to zero as noted in the sElect proof [17]. The last probability is δ^k bounded by the completeness component of GBA. This is immediate when p_{voter}^{verif} is equal to one since our definition assumes all honest voters vote and verify; when p_{voter}^{verif} is lower this is more complicated and requires guessing ahead of time which voters will verify. This can be achieved using standard techniques from complexity leveraging.

Appendix C Proof for Theorem 3

Proof. Consider an adversary $\mathcal{A}$ against $\mathsf{Exp}^{Ver}_{\mathcal{A},\mathcal{V}_r}(\lambda)$. We start by running $\mathcal{A}$ getting the output pd which we use for $\mathcal{B}$ in addition to an honestly generated signing keypair for AS. We then make a random guess about which voters $\mathcal{A}$ is going to ask to verify. The probability of guessing correctly is at least $1/2^{|\mathcal{I}|}$. Now, we keep running $\mathcal{A}$ to choose honestly cast votes and creating the bulletin board BB. Every time the vote oracle is called and we guessed the voter is going to verify, we let $\mathcal{B}$ query the same and forward the output to $\mathcal{A}$. If we guessed that the voter is not going to verify, we simply honestly generate the ballot and send it to $\mathcal{A}$ without $\mathcal{B}$ querying the vote oracle. We use the board BB output by $\mathcal{A}$ in addition to honestly generated signatures for AS. Since the signature scheme is perfectly correct, the signatures will verify in lines 4-7 of $\mathsf{Exp}^{\mathsf{GBA}}_{\mathcal{A},\mathcal{V}}(\lambda)$.

We now run $\mathcal{O}\mathsf{Verify}$ for $\mathcal{B}$ which will call verification for all voters used in the oracle calls in $\mathsf{Exp}^{\mathsf{GBA}}_{\mathcal{A},\mathcal{V}}(\lambda)$. We can use the outputs to $\mathcal{A}$'s calls to the verification oracle. Here we assume that we guessed the verifiers correctly and, further, in this case the two sets of verifying voters will be the same in the two experiments. For the sake of the proof, we will abort if they do not match, hence the degradation factor in the advantage. Now with probability $\frac{1}{2^{|\mathcal{I}|}}\mathsf{Adv}^{ver}_{\mathcal{A},\mathcal{V}_r}(\lambda)$ in $\mathsf{Exp}^{Ver}_{\mathcal{A},\mathcal{V}_r}(\lambda)$ we will have no complaints from the individual verification, the universal verification will be successful and we have $\neg(|\mathcal{I}| \geq |\mathsf{BB}|_{submit}| \geq |\mathsf{BB}|_{res}| \wedge \mathsf{V}[\mathsf{Checked}] \subseteq_{ms} \mathsf{BB}|_{res})$. Using that the scheme is accountability-verifiability-correct in $\mathsf{Exp}^{\mathsf{GBA}}_{\mathcal{A},\mathcal{V}}(\lambda)$ all individual verification will also produce no blame since the signatures will verify by perfect correctness, and, finally, again by accountability-verifiability-correctness and successful universal verification, no blame will be output by Judge, i.e. $|B| = 0$. Since the votes from the verifying voters, $\mathsf{V}[\mathsf{Checked}]$, in $\mathsf{Exp}^{\mathsf{GBA}}_{\mathcal{A},\mathcal{V}}(\lambda)$ exactly corresponds to the votes from the oracle vote calls in $\mathsf{Exp}^{\mathsf{GBA}}_{\mathcal{A},\mathcal{V}}(\lambda)$ and $|\mathsf{BB}|_{submit}| = |\mathsf{BB}_{vote}|$ and $\mathsf{BB}|_{res} = \mathsf{BB}_{dec}$ we exactly get the winning condition $(\neg(n \geq |\mathsf{BB}_{vote}| \geq |\mathsf{BB}_{dec}| \wedge V \subseteq \mathsf{BB}_{dec}) \wedge B = \perp)$ in $\mathsf{Exp}^{\mathsf{GBA}}_{\mathcal{A},\mathcal{V}}(\lambda)$.