Cronus: Everlasting Privacy with Audit and Cast

Thomas Haines*

Norwegian University of Science and Technology
thomas.haines@ntnu.no

Abstract. We present a new online voting scheme with everlasting privacy and cast-as-intended verifiability. We follow the so called "audit-and-cast" paradigm where the voter audits the ballot before casting it. To mitigate the ability of this information to harm the voter's privacy, we provide measures for avoiding coercion by allowing **any** party to create fake proofs for the content of **any** vote. We propose an efficient implementation and formally verify its security properties.

1 Introduction

This work focuses on simultaneously achieving two properties, everlasting privacy and cast-as-intended verification—along with end-to-end verifiability—in an efficient manner. Everlasting privacy is achieved because the public information in the scheme is unconditionally hiding. Further, our scheme provides timely and accountable cast-as-intended verification which the voter confirms before the ballot is accepted; this results in a simple vote ceremony which is dispute free. This prevents the issue found in many other schemes, where disgruntled voters can cast aspersions which cannot be resolved. While we do not claim that our scheme provides coercion resistance, we do provide to all parties the ability to create fake proofs of all votes. This means interested parties can run counter-coercion strategies on behalf of voters.

2 Related work

Electronic voting schemes have been the subject of much cryptographic study since the 1980s [3,7,8,9,11,12,24]. During this extensive study of verifiable electronic voting schemes many subtle issues have been detected. One such issue is receipt-freeness [4] and another is cast-as-intended verifiability. Unfortunately the former significantly complicates the latter. Prominent verifiable electronic voting schemes

* The author acknowledges support from the Luxembourg National Research Fund (FNR) and the Research Council of Norway for the joint project SURCVS. Part of this work was completed while the author was working at Polyas GmbH.

for remote voting like Helios [2] use Benaloh challenges [32] to achieve cast-as-intended verification without generating receipts, but this mechanism—as currently implemented—is (often) unusable [21]. Compounding the usability issues with some cast-as-intended mechanisms are the further issues of accountability and dispute management. We desire that electronic voting schemes provide strong evidence of misconduct, but the definitions of cast-as-intended verification require only that a voter can detect misconduct. While this issue is dealt with in some papers [10,30], in many deployed examples and papers [19,29,31] the definitions do not imply strong evidence. This lack of strong evidence means that if honest voters make mistakes in verification, or disgruntled voters desire to cast aspersions on the security of the election, there is no way to tell the difference between these scenarios and an attack. To provide strong evidence the definition needs to be strengthened to something closer to accountability [22].

Many voting protocols have been proposed with everlasting privacy and others have configurations which achieve this property. Everlasting privacy was proposed as an extension to the Helios scheme by Demirel et al. [14], extending the Split-ballot voting scheme from Moran and Naor [25]. This kind of extension reduces privacy attacks on the system (from an external adversary) to information theoretic security rather than computational. Hence, no future breakthrough in computation power, mathematics, or large scale quantum computers will put the voters' privacy at risk to the public. While there are schemes which provide information theoretic maximal privacy[1] these are impractical for most real elections. Demirel et al's scheme and many others, including our work here, have at least one authority against which privacy holds only computationally. Another set of schemes use some form of anonymous signature (ring, group, linkable) and an anonymous channel [23] which achieves everlasting privacy cleanly, but the existence of such a channel is problematic to realise [34].

Our work is similar to both Selene [29] and Guasch et al's work on "How to Challenge and Cast Your e-Vote" [17] in that we make use of a voter controlled trapdoor which allows the voter to be given a proof about which they can then equivocate to possible coercers. Our work provides stronger cast-as-intended verifiability than either of the above by avoiding dispute issues in the cast-as-intended mechanism. We achieve this by getting the voter to confirm that they are happy with the cast-as-intended check before submitting. Selene has a significantly different user experience where the voter checks that their vote was included after the election is complete, whereas Guasch et al's work has a similar voter experience to ours. In addition, while there seems to be no insurmountable barrier to updating either of the above schemes to have everlasting privacy, neither scheme currently has this property.

[1] That is, the adversary learns nothing more about the honest voters' input than it learned from the result and its own input.

3 Building Blocks

3.1 Definitions

We begin by stating the standard definitions.

Definition 1. *Encryption Scheme: An encryption scheme Π_ϵ consists of a triple of efficient algorithms $(Gen_\epsilon, Enc_\epsilon, Dec_\epsilon)$ such that:*

$Gen_\epsilon(1^n)$**:** *Given the security parameter 1^n, output a pair of public and secret keys (pk, sk). The public key specifies a message space M_ϵ.*

$Enc_\epsilon(pk, m)$**:** *Given a public key and a message m, return $\perp$ if $m \notin M_\epsilon$, otherwise output a ciphertext c. The explicit notation is $Enc_\epsilon(pk, m; r)$ for the randomness used in the encryption.*

$Dec_\epsilon(sk, c)$**:** *Using the private key sk, output a decryption d of the ciphertext c.*

We require an encryption system to be correct and at least preserve indistinguishability of messages under chosen plaintext attacks.

Definition 2. *Trapdoor Commitment Scheme: A trapdoor commitment scheme Π_c is a tuple of efficient algorithms $(Gen_c, GenTK_c, Com_c, Verify_c, Fake_c)$ such that:*

$Gen_c(1^n)$**:** *Given the security parameter 1^n, output a public commitment key ck and a proof that ck was generated without a known trapdoor. The commitment key specifies a message space M_c.*

$GenTK_c(1^n)$**:** *Given the security parameter 1^n, output a public commitment key ck and trapdoor key tk. The commitment key specifies a message space M_c.*

$Com_c(ck, m)$**:** *Given a commitment key and a message m, return $\perp$ if $m \notin M_c$, otherwise output a pair (c, d) made of a commitment and an opening. The explicit notation is $Enc_\epsilon(ck, m; r)$ for the randomness used in the commitment.*

$Verify_c(ck, c, d, m)$**:** *Given a commitment key, commitment, opening and message, return either 1 or 0 if these inputs are consistent or not.*

$Fake_c(ck, tk, c, d, m, m')$**:** *Given a commitment key, trapdoor key, commitment, opening and messages m and m' returns a new opening d'.*

Clearly the commitment scheme should be correct and it should not be possible to open the same commitment to two different messages without knowledge of the trapdoor key. In addition, our scheme requires that the commitments be statistically hiding.

Definition 3. *Signature Scheme: A signature scheme Π_s is a triple of efficient algorithms $(Gen_s, Sign_s, Verify_s)$ such that :*

$Gen_s(1^n)$**:** *Given a security parameter 1^n, output a pair of public and secret keys (pk, sk). The public key specifies a message space M_s.*

$Sign_s(sk, m)$**:** *Given a secret key and a message m, return $\perp$ if $m \notin M_s$, otherwise output a signature s.*

$Verify_s(pk, m, s)$**:** *Return either 1 if the signature verifies for the public key or 0 if not.*

A signature scheme should be correct and prevent even existential forgeries.

3.2 Specifics

We take as our primary building blocks the Pedersen commitment scheme [28] and Paillier encryption [27]. These have been very commonly used in e-voting with everlasting privacy, going back to Moran's seminal work [25], and were also used in [14] and [13]. In essence, one commits to the vote in a Pedersen commitment and then encrypts the opening under Paillier encryption. The unconditionally hiding Pedersen commitments can ensure the integrity of the election on the public board without revealing any information about the vote. In addition, we make use of verifiable mixnets and sigma protocols.

We note that Paillier has somewhat of a bad reputation in the e-voting community for complicating secure threshold key generation. While this was certainly a significant concern when Paillier was first proposed, subsequent research has changed the situation [26]; of particular note in addressing this concern is Hazay et al's [20] recent work. At present, the remaining reason not to prefer Paillier encryption is computational costs which we would argue are insufficiently problematic to justify the procedural issues incurred by most alternatives.

We note in passing that it is possible to adapt the very elegant PPATC scheme of Cuvelier et al. [13] for use with our scheme as a replacement for Paillier encryption and Pedersen commitment. The resulting scheme proceeds identically to how we describe, but instead of Pedersen commitments, Abe et al's [1] commitment scheme is used—once with known trapdoors and once without. This works because while PPATC is not additively homomorphic, an analogous sigma protocol to the "Sigma protocol for consistent commitments"—which we will define below—is available. We detail this sigma protocol in appendix A.

Paillier encryption consists of a triple of efficient algorithms $(Gen_{Pa}, Enc_{Pa}, Dec_{Pa})$ such that:

$Gen_{Pa}(1^n)$**:** Choose two n-bit primes, p and q, and compute the public modulus $N := pq$ for the Paillier encryption. The corresponding secret key $\lambda := \lambda(N)$ is the number $lcm(\phi(p), \phi(q)) = \phi(N)/2$. Output (N, λ).

$Enc_{Pa}(N, m)$**:** For $m \in \mathbb{Z}_N$ choose $r \in_R \mathbb{Z}_N^*$ and return $= (N+1)^m r^N \bmod N^2$.

$Dec_{Pa}(\lambda, c)$**:** For $c \in \mathbb{Z}_{N^2}$ return $([c^\lambda \bmod N^2] - 1)/N)[\lambda^{-1} * x \bmod N] \in \mathbb{Z}_N$.

Pedersen's commitment scheme consists of a tuple of efficient algorithms $(Gen_{Pe}, GenTK_{PE}\ Com_{Pe}, Verify_{Pe}, Fake_{Pe})$ such that:

$Gen_{Pe}(1^n)$**:** Choose a group G of order o, $2^{n-1} < o < 2^n$, in which the discrete log problem is hard. Select two generators g and h in a verifiably random way such that no non-trivial information about the relationship between is revealed, for instance as described in [16]; return $ck := (G, o, g, h)$. We also allow G to be given explicitly in which case the notation is $Gen_{Pe}(G)$.

$GenTK_{PE}(1^n)$**:** Choose a group G of order o approximately equal to 1^n in which the discrete log problem is hard. Select a random generator g and random element of $\mathbb{Z}_o$ x and set $h = g^x$; return $ck := (G, o, g, h)$ and $tk := x$. We also allow G to be given explicitly in which case the notation is $Gen_{Pe}(G)$.

$Com_{Pe}(ck, m)$**:** Given a commitment key and message $m \in \mathbb{Z}_o$ choose $r \in_R \mathbb{Z}_o$ and return $(g^r h^m, r)$.

$Verify_{Pe}(ck, c, d, m)$**:** Given a commitment key, commitment c, opening d and message m return 1 if $c = g^d h^m$ else 0.

$Fake_{Pe}(ck, tk, c, d, m, m')$**:** Recall that $tk = x$ where $h = g^x$, return $(r' := r + xm - xm')$.

In our scheme we use Pedersen commitments in two separate places (with different commitment keys). First, we use them to achieve everlasting privacy while preserving integrity. In this use, the ability to implant a trapdoor is undesirable, so the setup should produce two generators which are verifiably drawn uniformly and independently at random. They should be drawn in such a way that no more information about the relationship between them is revealed than can be easily computed given only the two elements. However, we also use the Pedersen commitments to allow the voters to verify that their ballot is cast as intended but later equivocate about the proof, in which case we make deliberate use of the trapdoors. **We emphasise that since these two different places have two different sets of keys no contradiction occurs by having one trapdoored and the other not.**

We can exploit the homomorphic properties of the above primitives to create efficient Zero Knowledge Proofs (ZKP) of correct encryption. Specifically given a Pedersen commitment of the vote c, Paillier ciphertexts opening the commitment c_1 and c_2, and an additional verification Pedersen commitment c_v, we can prove that the ciphertexts do encrypt an opening to the commitment and that the additional verification commitment refers to the same vote. The sigma protocol for correct encryption first appeared in [13].

Sigma protocol for correct encryption Given a $ck, pk, (c, c_1, c_2)$ we show that we know (v, r, r_1, r_2) such that $c = Com_{Pe}(ck, v; r)$, $c_1 = Enc_{Pa}(N, r; r_1)$, and $c_2 = Enc_{Pa}(N, v; r_2)$.

1. Prover chooses (v', r', r'_1, r'_2) at random and computes $c' = Com_{Pe}(ck, v'; r')$, $c'_1 = Enc_{Pa}(pk, r'; r'_1)$, and $c'_2 = Enc_{Pa}(pk, v'; r'_2)$ and returns (c', c'_1, c'_2).
2. Verifier sends a challenge e chosen at random in $\mathbb{Z}_N$.
3. Prover computes $t_1 := v' + ev$, $t_2 := r' + er$, $t_3 := r'_1 {r'_1}^e$, and $t_4 := r'_2 {r'_2}^e$ and sends these to the verifier.

4. The verifier accepts if $c'c^e = Com_{Pe}(ck, t_1; t_2)$ and $c'_1 c_1^e = Enc_{Pa}(pk, t_2; t_3)$ and $c'_2 c_2^e = Enc_{Pa}(pk, t_1; t_4)$.

Sigma protocol for consistent commitments Given a $ck, ck_v, (c, c_v)$ we show that we know (v, r, r_v) such that $c = Com_{Pe}(ck, v; r)$, and $c_v = Com_{Pe}(ck_v, v; r_v)$.

1. Prover chooses (v', r', r'_v) at random and computes $c' = Com_{Pe}(ck, v'; r')$, and $c'_v = Com_{Pe}(ck_v, v'; r'_v)$ and returns (c', c'_v).
2. Verifier sends a challenge e chosen at random in $\mathbb{Z}_N$.
3. Prover computes $t_1 := v' + ev$, $t_2 := r' + er$, and $t_3 := r'_v + er_v$ and sends these to the verifier.
4. The verifier accepts if $c'c^e = Com_{Pe}(ck, t_1; t_2)$ and $c'_v c_v^e = Com_{Pe}(ck_v, t_1; t_3)$.

We can make the sigma protocol for correct encryption non-interactive by applying the Fiat-Shamir transform, and we will in future refer to **CEProve** and **CEVerify** for the non-interactive prover and verifier functions respectively. We do the same for the consistent commitment sigma protocol and refer to **CCProve** and **CCVerify**.

The Sigma protocol for correct encryption above can be modified to prove correct re-encryption of a tuple of a Pedersen commitment and two Paillier ciphertexts. Given the sigma protocols for correct re-encryption we can apply Wikström's general result from [33] to construct a mixnet. An optimised variant of Wikström's mixnet for shuffling Pedersen and Paillier together recentely appeared in [18].

We will refer to the mixnet for the Pedersen commitment and Paillier ciphertexts together as **Mix** and its verification algorithm as **MixVerify**; similarly we denote the mixnet for commitments alone as **Mix'** and its verification algorithm as **MixVerify'**. We denote **MixSimulate** and **MixSimulate'** the simulators for the mixnets which are obtained by reprogramming the random oracle.

It is a trivial equivalence for the authorities to check that the commitments are shuffled according to the same permutation and updated by the same randomness factors on both boards. We note that in this case, the structure of Wikström's proof also works for a significantly optimised variant of the mixnet from the general result, as is also true in the case of ElGamal for instance; we omit the details.

4 Cronus E-Voting Scheme

We now describe the scheme which uses two bulletin boards BB and sBB; the first is a public board and the second can only be seen by the authorities. Since BB is public information all algorithms are assumed to have access to all its contents. For simplicity we describe it with a single key holder although threshold key generation is available:

Setup(1^n) runs $Gen_{Pa}(1^n)$ and receives (N, λ), chooses k such that $kN+1$ is prime, denoting the subgroup of order N in $\mathbb{Z}_{kN+1}$ as G and runs $Gen_{Pe}(G)$ and receives $(g,\ h)$, and sets the election public key $pk = (N, G, g, h)$ and $sk = (\lambda)$. Then it generates the empty list of credentials **ID** and posts pk and **ID** BB.

Register(1^n, id) runs $GenTK_{Pe}(G)$ and $Gen_s(1^n)$ and sets $pk_{id} = (ck, pk_s)$ and $sk_{id} = (tk, sk_s)$, and posts (id, pk_{id}) to BB unless id already appears on the board in which case it aborts.

CreateVote(v, id) retrieves $(pk = (N, G, g, h), pk_{id} = (ck, pk_s))$ from BB. Then, it runs $Com_{Pe}((g, h), v)$ and receives (c, r); it then chooses r_1 and r_2 at random in $\mathbb{Z}_N^*$, then runs $Enc_{Pa}(N, r; r_1)$ and $Enc_{Pa}(N, v; r_2)$ and receives c_1 and c_2. It then runs $Com_{Pe}(ck, v)$ and receives (c_v, r_v). The voting device calls **CEProve**$(c, c_1, c_2)(v, r, r_1, r_2)$ and **CCProve**$(c, c_v)(v, r, r_v)$ and receives π_{CE} and π_{CC}. The ballot b is then $(c, c_1, c_2, c_v, \pi_{CE}, \pi_{CC})$. The device also outputs (v, r_v) which will be used to audit the ballot.

CastBallot(b, sk_{id}, id) runs $Sign_{sk_{id}}(b)$ and receives ζ. The authenticated ballot is then $b_a = (id, b, \zeta)$.

ProcessBallot(b_a) parses b_a as (id, b, ζ) and b as $(c, c_1, c_2, c_v, \pi_{CE}, \pi_{CC})$. It checks that $Verify_s(pk_{id}, b, \zeta) = 1$ and that **CEVerify**$(c, c_1, c_2, \pi_{CE}) = 1$ and that **CCVerify**$(c, c_v, \pi_{CC}) = 1$. If tk_{id} is not present BB then posts (id, c, c_v, π_{CC}) to the public bulletin BB and (id, c_1, c_2, π_{CE}) to the private bulletin board sBB.

AuditVote(id, v, r_v). On receiving (id, v, r_v), the audit device accesses the bulletin board to retrieve (c, c_v, π_{CC}). It checks that **CEVerify**$(c, c_v, \pi_{CC}) = 1$ and that $Verify_{Pe}(ck_{id}, c_v, v, r_v) = 1$. If these checks pass it returns 1, otherwise 0.

ConfirmBallot$(id, sk_{id}, (v, r_v))$ parses sk_{id} as (tk, sk_s) and retrieves pk_{id} parsing it as (ck, pk_s). It then checks that tk is valid for ck and that $Verify_{Pe}(ck, c_v, v, r_v) = 1$, if so it chooses v' at random from the set of valid votes and posts (tk, v', r'_v) to the BB where $r'_v = Fake_{Pe}(ck, tk, c_v, r_v, v, v')$, otherwise it returns 0.

Tally$(sk,\ sBB)$

Filter First the authorities filter out the ballots which were not confirmed.

Parallel shuffle They then take the set of commitments on the public board and the set of commitments taken with the ciphertexts on the private board. They then re-randomise and shuffle these sets using **Mix'** and **Mix** respectively, checking that the commitments on the two boards match at each step. We denote by Φ_1 and Φ_2 the Non-Interactive Zero Knowledge Proof (NIZKP) proofs of correct mixing for the public and private board respectively. The output set of commitments along with intermediary values and Φ_1 are posted to the public board BB and the output set of ciphertexts and Φ_2 are posted to the secret board sBB.

Decryption The authorities then jointly decrypt the set of ciphertexts of the form (c'_1, c'_2) to recover (r', v) which are then posted to the public board.

VerifyTally(pk) to verify that a set of votes $(v_1, ..., v_n)$ are the correct result, an auditor retrieves the input commitments, output commitments, intermediary values, openings and Φ_i for the public board. It then runs **Mix'** and checks that $Verify_{Pe}(ck, c', v, r') = 1$ for all commitments c' and openings r'. If all checks pass it returns 1, else 0.

4.1 Election Flow

The election protocol has the following participants: *Election Authorities*, *Registrars*, and *Voters* whom we assume have access to a *Voting device* and *Audit device*.

Setup Phase Before an election, the set of election authorities set up the public parameters as defined by **Setup**, which they publish to the public bulletin board BB—and also define the voting options and the tally function. The voters are registered by the registrars, using the **Register** function, and the voters pk_{id} are posted to the BB.

Vote Casting Phase During the vote casting stage, voters access their voting devices and cast their ballots using **CreateVote** and **CastBallot**, and receive v, r_v. They then provide v, r_v to the audit device which checks the ballot using **AuditBallot**. If the ballot audit is successful, the voter then confirms their ballot using **Confirm Ballot**. The voter should check that both their voting device and audit device agree that the ballot has been confirmed.

Tallying Phase After voting is over, the authorities run **Tally** which runs the mixnets and tallies the votes.

Audit Phase After the election, any party can check the signatures on the submitted ballots to see that ballots were collected as cast and no ballots from ineligible voters are present. They can also run **VerifyTally** to check that the ballots were counted as collected.

The voter's experience in the most straightforward instantiation of the scheme is of authenticating and voting on one device, which then displays a QR code. The voter then scans this with a second device which confirms that the encrypted ballot encodes the voter's choice. The voter then confirms their ballot.

5 Security Definitions and Analysis

In defining Ballot Privacy, we follow the Ballot PRIVacy (BPRIV) definitions of Bernhard et al. [5] which we have slightly modified, while our cast-as-intended definition is similar to Escala et al. [15,17]. We also briefly discuss why everlasting privacy holds and why the encryption is Non-Malleable and indistinguishable under a Chosen Plaintext Attack (NM-CPA).

5.1 Ballot privacy

The intuition for the formal definition of privacy is that privacy should hold even against insiders up to and including any subset of the authorities, less than the threshold that can recover the key—that threshold can trivially break privacy by decrypting the ciphertexts next to the voter id. Formally we define ballot privacy by the adversary advantage in the following experiment. Note that in our privacy definitions we prepend the bulletin board on which the algorithm is running to its list of inputs when necessary for clarity.

$\mathbf{Exp}^{bpriv,\beta}_{\mathcal{B},\mathcal{V}}(\lambda)$
  $(pk, sk) \leftarrow \mathbf{Setup}(1^\lambda)$
$\mathcal{O}voteLR(id, v_0, v_1)$
  $sk_{id} = \mathbf{Register}(1^n, id)$
  Let $(b_0, (v_0, r_0)) = \mathbf{CastBallot}(\mathbf{VoteCreate}(v_0, id), sk_{id}, id)$ and
    $(b_1, (v_1, r_1)) = \mathbf{CastBallot}(\mathbf{VoteCreate}(v_1, id), sk_{id}, id)$
  If $\mathbf{ProcessBallot}(b_\beta) = 0$ return 0.
  If $\mathbf{ConfirmBallot}(id, sk_{id}, (v_\beta, r_\beta)) = 0$ return 0.
$\mathcal{O}cast(id, b, (v, r))$
  $sk_{id} = \mathbf{Register}(1^n, id)$
  If $\mathbf{ProcessBallot}(b) = 0$ return 0.
  If $\mathbf{ConfirmBallot}(id, sk_{id}, (v, r)) = 0$ return 0.
$\mathcal{O}board()$
  return (BB_β, sBB_β)
$\mathcal{O}tally()$ for $\beta = 0$
  $(r, \phi) \leftarrow \mathbf{Tally}(BB_0, sBB_0, sk)$
$\mathcal{O}tally()$ for $\beta = 1$
  $(r, \phi) \leftarrow \mathbf{Tally}(BB_0, sBB_0, sk)$
  $\phi' \leftarrow \mathbf{SimProof}(BB_1, r)$
  $return(r, \phi')$

Definition 4. *BPRIV Consider a voting scheme* $\mathcal{V} = (\textbf{\textit{Setup}}, \textbf{\textit{Register}}, \textbf{\textit{CreateVote}}, \textbf{\textit{CastBallot}}, \textbf{\textit{ProcessBallot}}, \textbf{\textit{AuditVote}}, \textbf{\textit{ConfirmBallot}}, \textbf{\textit{Tally}}, \textbf{\textit{VerifyTally}})$ *for a set* I *of voter identities for a result function* p*. We say the scheme has ballot privacy if there exists an algorithm* ***SimProof*** *such that no efficient adversary can distinguish between games* $\textbf{\textit{Exp}}^{bpriv,0}_{\mathcal{B},\mathcal{V}}(\lambda)$ *and* $\textbf{\textit{Exp}}^{bpriv,1}_{\mathcal{B},\mathcal{V}}(\lambda)$ *defined by the oracles above, that is for any efficient algorithm* $\mathcal{A}$

$$|Pr[\textbf{\textit{Exp}}^{bpriv,0}_{\mathcal{B},\mathcal{V}}(\lambda) = 1] - Pr[\textbf{\textit{Exp}}^{bpriv,1}_{\mathcal{B},\mathcal{V}}(\lambda)]|$$

is negligible in λ*.*

Due to similarities between between Helios and our scheme the proof of ballot privacy is similar. However, there are significant differences; we have a public and a confidential bulletin board and a two-stage ballot casting process. For the

purpose of proving ballot privacy we assume casting and confirming a ballot is an atomic process, since the adversary has less information to attack privacy when his process aborts rather then completes this does not enable any attacks. We define **BB** as the union of BB and sBB so that it contains entries of the form $(id, b = (c, c_1, c_2, c_v))$.

Recall the observation that since the visible bulletin board is built through the $\mathcal{O}voteLR(id, v_0, v_1)$ and $\mathcal{O}cast(id, b, (v, r))$ queries, our BPRIV reduction can associate, to any entry (id_i, b_i) in the visible bulletin board, a tuple $(id_i, b_i^0, b_i^1, v_i^0, v_i^1)$.

[**Game** G_{-1}] Let G_{-1} be the BRPIV game corresponding to Experiment $\mathbf{Exp}^{bpriv,0}_{\mathcal{B},\mathcal{V}^{Cronus}}$. The BPRIV adversary $\mathcal{A}$ sees the ballot box $\mathbf{BB}_0$ and an oracle $\mathcal{O}tally()$ faithfully answered.

[**Game** G_0] Let G_0 be the same as Game G_{-1} except the tallying proof ϕ' is produced by **MixSimulate** and **MixSimulate'** by reprogramming the Random Oracle G. Due to the zero-knowledge property of the NIZKP proof associated with the mixnet, the distinguishing probability is negligibly close between the two games. From now on the proof is always simulated.

[**Game** $G_{0,i}$] is obtained from Game $G_{0,i-1}$ by taking two possible actions depending on the contents of the tuple $(id_i, b_i^0, b_i^1, v_i^0, v_i^1)$: if $b_i^0 = b_i^1$ do nothing, else $b_i^0 \neq b_i^1$ replace the i-th entry (id_i, b_i^0) in BB_0 with (id_i, b_i^1). By the NM-CPA property of our construction, the distinguishing probability of the adversary is negligibly close to $G_{0,i-1}$.

[**Game** G_1] Let G_1 be Game $G_{0,n}$. The view of the adversary in Game G_1 corresponds to the view of the BPRIC adversary with $\beta = 1$. Cronus is thus BPRIV private.

5.2 Cast-as-Intended Verifiability

The challenger $\mathcal{C}$ calls $\mathbf{Setup}(1^\lambda)$ and provides (N, G, g, h) to the adversary. For a list of voters ID and Bulletin boards BB, sBB, it provides the following oracles.

$\mathcal{O}registerHonest(id)$
: $\mathcal{A}$ provides $id \notin ID$. The challenger $\mathcal{C}$ calls $\mathbf{Register}(1^\lambda, id)$ adding (id, pk_{id}) to ID.

$\mathcal{O}registerCorrrupt(id, pk_{id})$
: $\mathcal{A}$ provides $id \notin ID$. The challenger $\mathcal{C}$ adds (id, pk_{id}) to ID.

$\mathcal{O}cast(id, b)$
: $\mathcal{C}$ returns $\mathbf{CastBallot}(b, sk_{id}, id)$.

$\mathcal{O}process(b_a)$

$\mathcal{C}$ returns **ProcessBallot**(b_a).

$\mathcal{O}confirmHonest(id, v, r_v)$

If **AuditVote**(id, v, r_v) returns 1 then return sk_{id} and run **ConfirmBallot**$(id, sk_{id}, (v, r_v))$.

$\mathcal{O}confirmCorrupt(id, b_a)$

return sk_{id}.

Definition 5. *Cast-as-Intended Verification: Consider a voting scheme* $\mathcal{V} =$ (***Setup***, ***Register***, ***CreateVote***, ***CastBallot***, ***ProcessBallot***, ***AuditVote***, ***ConfirmBallot***, ***Tally***, ***VerifyTally***) *for a set* ID *of voter identities and a result function* p*. We say the scheme has cast-as-intended verifiability if there exists no efficient adversary that can win the following game with greater than negligible probability.*

The adversary wins if there exists a confirmed ballot $(id, c, c_v) \in BB$ and $(id, c_1, c_2) \in sBB$ such that the following conditions hold:

- $id \in ID$ and id was not the input to $\mathcal{O}registerCorrupt$.
- $Dec_{Pa}(sk, c_2) \neq v$ where v was the voting option submitted by the adversary to $\mathcal{O}confirmHonest$.

Provided that Cronus is instantiated with a secure signature scheme $(Gen_s, Sign_s, Verify_s)$, sound Pedersen commitment scheme $(Gen_{Pe}, GenTK_{Pe}, Com_{Pe}, Verify_{Pe}, Fake_{Pe})$, and sound NIZKP schemes (**CEProve**, **CEVerify**),(**CCProve**, **CCVerify**)—and one of the devices is honest—no such efficient adversary exists.

First note that all entries on the board, confirmed or otherwise, were placed there by the $\mathcal{O}process$ oracle. This means that the adversary has shown that it knows an opening v, r, r_1, r_2 such that $c = Com_{Pe}(ck, v; r)$, $c_1 = Enc_{Pa}(N, r; r_1)$ and $c_2 = Enc_{Pa}(N, v; r_2)$; and, that it knows an opening v', r', r_v such that $c = Com_{Pe}(ck, v'; r')$ and $c_v = Com_{Pe}(ck, v'; r_v)$. By the binding property of the Pedersen commitments $v = v'$ and $r = r'$.

Secondly, observe that all entries on the board not violating the first condition were confirmed there through the $\mathcal{O}confirmHonest$ oracle. This oracle checks that the adversary can open the commitment c_v to v before it confirms the ballots. Since the Pedersen commitments are binding, this v must be the only opening which the adversary knows; and, hence, the same v which it showed to be equal to contents of the ciphertext c_2 when the ballot was processed. Therefore, by the soundness of **CEProve** and **CCProve** and the binding property of the commitment scheme, the vote v submitted to $\mathcal{O}confirmHonest$ is the vote which c_2 decrypts to.

We have just shown that the first device is unable to submit a different vote without being detected, but what if the second device is corrupt? If both devices are corrupt then the voter has no integrity guarantees. However, it is also clear

that if the audit device is corrupt but the voting device is honest the ballot is sent correctly and the voting device will honestly tell the voter when their ballot is confirmed. We note that if one device complains that the other has misbehaved, diagnosing which device is actually malicious is non-trivial.

5.3 Strong Consistency and Strong Correctness

We define the following extraction and verification algorithms consistent with Tally and ProcessBallot:

1. Extract$((c, c_1, c_2, c_v, \pi_{CE}, \pi_{CC}), sk)$ first verifies the proofs π_{CE} and π_{CC} and if either fails returns $\perp$. Otherwise decrypts c_2 and return the result.
2. ValidInd(b) checks that the ballot is valid and the proofs are correct. That is, given a ballot $b = (c, c_1, c_2, c_v, \pi_{CE}, \pi_{CC})$ it checks that both π_{CE} and $\pi_C C$ verify.

Definition 6. *Strong Consistency: A voting protocol has strong consistency if the following hold:*

- *For all pk from* ***Setup*** *for any voter v, for any voter identity id and $pk_i d$ for* ***Register****(id), Extract(****CreateVote****(v,id)) = v.*
- ***ProcessBallot****$(b_a) = 1$,* ***AuditBallot****$(id, v, r_v) = 1$,* ***ConfirmBallot****$(id, sk_i d, (v, r_v)) =$ 1 implies that $ValidInd(b) = 1$.*
- *Where the adversary's chance of winning the following game is negligible,*
  Setup phase *The challenger runs* ***Setup****(1^n) to generate pk and sk, which it gives to $\mathcal{A}$.*
  Bulletin Board *$\mathcal{A}$ submits a bulletin board BB and sBB.*
  Counting phase *The challenger runs* ***Tally****(sk) and obtains the set of output votes r and tally proof ϕ_1.*
  Output *The adversary $\mathcal{A}$ wins if $r \neq Extract(BB, sk)$, where Extract is applied to each confirmed ballot on the bulletin board.*

The first property follows trivially from the correctness of Paillier encryption. The second property follows because **ProcessBallot** checks the same proofs as ValidInd. The last property follows trivially because the definition of extract and tally are the same up to mixing.

Definition 7. *Strong Correctness: A voting protocol has strong correctness if given pk =* ***Setup****(1^n), for any efficient algorithm $\mathcal{A}$, the following probability*

$$Pr[(id, v, BB) \leftarrow \mathcal{A}(pk); \textbf{\textit{Register}}(1^n, id); \textbf{\textit{CreateVote}}(v, id) = b, (v, r_v);$$
$$\textbf{\textit{CastBallot}}(b, sk_{id}, id) = b_a : \textbf{\textit{ProcessBallot}}(b_a) = 0 \vee \textbf{\textit{AuditVote}}(id, v, r_v) = 0$$
$$\vee \textbf{\textit{ConfirmBallot}}(id, sk_{id}, (v, r_v) = 0]$$

is negligible in λ, where id does not appear on BB.

By definition of **ProcessBallot**, **AuditVote** and **ConfirmBallot** the above definition implies that the following events must occur except with negligible probability. The signature produced by **CastBallot** must verify. The sigma protocol transcripts produced by **CreateVote** must verify. The tk_{id} must not already appear on the bulletin board. The commitment c_v must open to v, r_v. All of these events are implied by the correctness of the relevant primitives with the exception of tk_{id} not appearing on the bulletin board; for this, we require that id does not already appear on the BB as produced by the adversary.

5.4 Everlasting Privacy

Everlasting privacy is fairly straightforward. The public bulletin board BB contains the following information for each confirmed ballot at the end of the election $(id, pk_{id} = (ck, pk_s), c, c_v, \pi_{CC})$. The commitments c and c_v are statistically hiding and hence leak negligible information about the vote regardless of adversary's computational power. The proof π_{CC} is honest verifier zero knowledge and also leaks negligible information about the vote regardless of adversary computational power. Since no non-negligible information is leaked about the vote based on the public information, we conclude that the scheme has everlasting privacy.

5.5 Encryption

It is known that an Indistinguishable under Chosen Plaintext Attack (IND-CPA) secure cryptosystem plus a Simulation Sound Extractable Proof of Knowledge (SSE-PoK) is Non-Malleable under Chosen Plaintext Attack (NM-CPA) [6]. It also known that applying the strong Fiat-Shamir transform to sigma protocols yields an SSE-PoK. We therefore have that our construction is also NM-CPA secure provided that Paillier-Encryption is IND-CPA secure. It is also necessary to prevent simple duplication of ballots; this can be done by filtering duplicates or including the voter id in the input to the hash function challenge generator in the non-interactive version.

6 Practical Realisation

In practice the voter needs to have access to the keys in some way. One option is that the voter has a trusted authentication device which handles the signing and trapdoor for them; however, this reduces the practicality of the scheme. Another option is to provide the voter with a confirmation code and distribute the trapdoor among the set of tellers; the set of tellers would release the trapdoor when the voter confirms. Interestingly, this realisation does not affect the integrity of the scheme—unless the first device colludes with the tellers—but has a mild impact on privacy.

7 Conclusion

We have presented a straightforward and effective scheme with cast-as-intended verifiability, everlasting privacy, and universal verifiability. The scheme avoids many of the issues hitherto present in similar schemes through careful use of checks and zero-knowledge proofs. The construction as we present it relies on Pedersen commitments and Paillier encryption; however, it is equally possible to instantiate over elliptic curves of prime order with bilinear pairings.

References

1. Masayuki Abe, Kristiyan Haralambiev, and Miyako Ohkubo. Group to group commitments do not shrink. In *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 301–317. Springer, 2012.
2. Ben Adida. Helios: Web-based open-audit voting. In Paul C. van Oorschot, editor, *USENIX Security Symposium*, pages 335–348. USENIX Association, 2008.
3. Josh C Benaloh and Moti Yung. Distributing the power of a government to enhance the privacy of voters. In *Proceedings of the fifth annual ACM symposium on Principles of distributed computing*, pages 52–62. ACM, 1986.
4. Josh Cohen Benaloh and Dwight Tuinstra. Receipt-free secret-ballot elections (extended abstract). In Frank Thomson Leighton and Michael T. Goodrich, editors, *STOC*, pages 544–553. ACM, 1994.
5. David Bernhard, Véronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. Sok: A comprehensive analysis of game-based ballot privacy definitions. In *IEEE Symposium on Security and Privacy*, pages 499–516. IEEE Computer Society, 2015.
6. David Bernhard, Olivier Pereira, and Bogdan Warinschi. How not to prove yourself: Pitfalls of the fiat-shamir heuristic and applications to helios. In *ASIACRYPT*, volume 7658 of *Lecture Notes in Computer Science*, pages 626–643. Springer, 2012.
7. David Chaum. Untraceable mail, return addresses and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.
8. David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, 1985.
9. David Chaum. Elections with unconditionally-secret ballots and disruption equivalent to breaking rsa. In Christoph G. Günther, editor, *EUROCRYPT*, volume 330 of *Lecture Notes in Computer Science*, pages 177–182. Springer, 1988.
10. David Chaum, Richard Carback, Jeremy Clark, Aleksander Essex, Stefan Popoveniuc, Ronald L. Rivest, Peter Y. A. Ryan, Emily Shen, and Alan T. Sherman. Scantegrity ii: End-to-end verifiability for optical scan election systems using invisible ink confirmation codes. In *EVT*. USENIX Association, 2008.
11. Josh D Cohen. *Improving privacy in cryptographic elections*. Citeseer, 1986.
12. Josh D Cohen and Michael J Fischer. A robust and verifiable cryptographically secure election scheme. In *FOCS*, volume 85, pages 372–382, 1985.
13. Edouard Cuvelier, Olivier Pereira, and Thomas Peters. Election verifiability or ballot privacy: Do we need to choose? In *European Symposium on Research in Computer Security*, pages 481–498. Springer, 2013.

14. Denise Demirel, Jeroen Van De Graaf, and Roberto Araújo. Improving helios with everlasting privacy towards the public. In *Proceedings of the 2012 international conference on Electronic Voting Technology/Workshop on Trustworthy Elections*, pages 8–8. USENIX Association, 2012.
15. Alex Escala, Sandra Guasch, Javier Herranz, and Paz Morillo. Universal cast-as-intended verifiability. In Jeremy Clark, Sarah Meiklejohn, Peter Y. A. Ryan, Dan S. Wallach, Michael Brenner, and Kurt Rohloff, editors, *Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers*, volume 9604 of *Lecture Notes in Computer Science*, pages 233–250. Springer, 2016.
16. PUB FIPS. 186-4: Federal information processing standards publication. digital signature standard (dss). *Information Technology Laboratory, National Institute of Standards and Technology (NIST), Gaithersburg, MD*, pages 20899–8900, 2013.
17. Sandra Guasch and Paz Morillo. How to challenge and cast your e-vote. In Jens Grossklags and Bart Preneel, editors, *Financial Cryptography and Data Security - 20th International Conference, FC 2016, Christ Church, Barbados, February 22-26, 2016, Revised Selected Papers*, volume 9603 of *Lecture Notes in Computer Science*, pages 130–145. Springer, 2016.
18. Thomas Haines and Clementine Gritti. Improvements in everlasting privacy: Efficient and secure zero knowledge proofs. Cryptology ePrint Archive, Report 2019/901, 2019. https://eprint.iacr.org/2019/901.
19. J Alex Halderman and Vanessa Teague. The new south wales ivote system: Security failures and verification flaws in a live online election. In *International Conference on E-Voting and Identity*, pages 35–53. Springer, 2015.
20. Carmit Hazay, Gert Læssøe Mikkelsen, Tal Rabin, Tomas Toft, and Angelo Agatino Nicolosi. Efficient RSA key generation and threshold paillier in the two-party setting. *J. Cryptology*, 32(2):265–323, 2019.
21. Fatih Karayumak, Maina M. Olembo, Michaela Kauer, and Melanie Volkamer. Usability analysis of helios - an open source verifiable remote electronic voting system. In Hovav Shacham and Vanessa Teague, editors, *2011 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections, EVT/WOTE '11, San Francisco, CA, USA, August 8-9, 2011*. USENIX Association, 2011.
22. Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Accountability: definition and relationship to verifiability. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010*, pages 526–535. ACM, 2010.
23. Philipp Locher, Rolf Haenni, and Reto E Koenig. Coercion-resistant internet voting with everlasting privacy. In *International Conference on Financial Cryptography and Data Security*, pages 161–175. Springer, 2016.
24. M Merrit. Cryptographic protocols. Ph.D. Thesis, 1983.
25. Tal Moran and Moni Naor. Split-ballot voting: Everlasting privacy with distributed trust. *ACM Trans. Inf. Syst. Secur.*, 13(2), 2010.
26. Takashi Nishide and Kouichi Sakurai. Distributed paillier cryptosystem without trusted dealer. In *WISA*, volume 6513 of *Lecture Notes in Computer Science*, pages 44–60. Springer, 2010.

27. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238. Springer, 1999.
28. Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, 1991.
29. Peter Y. A. Ryan, Peter B. Rønne, and Vincenzo Iovino. Selene: Voting with transparent verifiability and coercion-mitigation. In *Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers*, pages 176–192, 2016.
30. P.Y.A Ryan. A variant of the Chaum voter-verifiable scheme. In *Proceedings of the 2005 workshop on Issues in the theory of security*, pages 81–88. ACM, 2005.
31. Drew Springall, Travis Finkenauer, Zakir Durumeric, Jason Kitcat, Harri Hursti, Margaret MacAlpine, and J. Alex Halderman. Security analysis of the estonian internet voting system. In *ACM Conference on Computer and Communications Security*, pages 703–715. ACM, 2014.
32. Dan S. Wallach and Ronald L. Rivest, editors. *Simple Verifiable Elections*. USENIX Association, 2006.
33. Douglas Wikström. A commitment-consistent proof of a shuffle. In *Australasian Conference on Information Security and Privacy ACISP*, volume 5594 of *Lecture Notes in Computer Science*, pages 407–421. Springer, 2009.
34. Nan Yang and Jeremy Clark. Practical governmental voting with unconditional integrity and privacy. In Michael Brenner, Kurt Rohloff, Joseph Bonneau, Andrew Miller, Peter Y. A. Ryan, Vanessa Teague, Andrea Bracciali, Massimiliano Sala, Federico Pintore, and Markus Jakobsson, editors, *Financial Cryptography and Data Security - FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017, Revised Selected Papers*, volume 10323 of *Lecture Notes in Computer Science*, pages 434–449. Springer, 2017.

A Sigma protocol for consistent Abe commitments

We present a sigma protocol which shows that the prover can open two of Abe et al's [1] commitments to the same message. Recall that Abe et al's commitments are defined over an elliptic curve coupled with a bilinear pairing; we denote the groups of the curve as $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$. Given two generators for $\mathbb{G}_1$ denoted G_0, G_1 and a generator for $\mathbb{G}_2$ denoted H, a commitment to a message m using randomness r, r' is a tuple $(H^{r_1}m, G_0^r G_1^{r_1})$.

Sigma protocol for consistent commitments Given a $\mathbb{G}_1, \mathbb{G}_2, G_0, G_1, H, (c_1, c_2)$, (c_1', c_2') the prover shows that they know (r, r', r_1, r_1') such that $c_1/c_1' = H^r/H^{r'}$, $c_2 = G_0^r G_1^{r_1}$, and $c_2' = G_0^{r'} G_1^{r_1'}$.

1. Prover chooses (s, s', s_1, s_1') at random and computes $com_1 = H^s/H^{s'}$, $com_2 = G_0^s G_1^{s_1}$, and $com_3 = G_0^{s'} G_1^{s_1'}$ and returns (com_1, com_2, com_3).

2. Verifier sends a challenge e chosen at random in $\mathbb{Z}_N$.
3. Prover computes $t_1 := s+er$, $t_2 := s'+er'$, $t_3 := s_1+er_1$, and $t_4 := s'_1+er'_1$ and sends these to the verifier.
4. The verifier accepts if $com_1(c_1/c'_1)^e = H^{t_1}/H^{t_2}$ and $com_2 c_2^e = G_0^{t_1} G_1^{t_3}$ and $com_3 {c'}_2^e = G_0^{t_2} G_1^{t_4}$.

The proof is straightforward and we omit it due to lack of space.