BeleniosRF: A Strongly Receipt-Free Electronic Voting Scheme

Véronique Cortier[1] Georg Fuchsbauer[2,*] David Galindo[3]

[1] CNRS/LORIA, France
[2] IST Austria
[3] SCYTL Secure Electronic Voting, Spain

Abstract

We propose a new voting scheme, BeleniosRF, that offers both strong receipt-freeness and end-to-end verifiability. It is strongly receipt-free in the sense that even dishonest voters cannot prove how they voted. We give a game-based definition capturing this property, inspired by and improving the original receipt-freeness definition by Benaloh and Tuinstra. Built upon the Helios protocol, BeleniosRF inherits from its simplicity.

1 Introduction

Electronic voting protocols should achieve two antagonistic security goals: privacy and verifiability. Additionally, they must be practical, from a usability, operational and efficiency point of view. Privacy can be expressed via several, increasingly demanding security properties.

- Basic *ballot privacy* guarantees that no one can learn how a voter voted.
- *Receipt-freeness* ensures that a voter cannot prove to anyone how he voted. While privacy protects honest voters, receipt-freeness aims at protecting vote privacy even when voters willingly provide information to an attacker.
- *Coercion-resistance* should allow an honest voter to cast his vote even if he is, during some time, fully under the control of an attacker. Coercion-resistance typically requires revoting.

To our knowledge, Civitas [JCJ05,CCM08] is the only scheme that achieves both verifiability and coercion-resistance, without requiring a great deal of interaction between the ballot box or the election authorities and the voter (such as [BT94]). While the scheme is a foundational work, it seems difficult to use it in large-scale elections for two main reasons. First, the tally phase requires $O(n^2)$ operations where n is the number of received ballots, which opens the way to denial-of-service attacks. Second, to achieve verifiability, a voter should both be able to recognize his ballot and lie about it. This either requires cryptographic skills or a heavy infrastructure (e.g. in person registration).

In contrast, Helios [Adi08,AdMPQ09] is a scheme that "only" achieves privacy and verifiability. Helios is based on a classical voting system proposed by Cramer, Gennaro and Schoenmakers [CGS97] with variants proposed by Benaloh [Ben07]. It has been used in several real-world elections such as that of the president of the University of Louvain-la-Neuve in Belgium. It is also used by the International Association for Cryptographic Research (IACR) for its elections since 2011 [IAC]. As emphasized by its authors, Helios should only be used in low-coercion

[*]Supported by the European Research Council, ERC Starting Grant (259668-PSPC).

environments. Indeed, a voter may easily reveal how he voted by exhibiting the randomness used by his computer to compute his ballot.

Our contribution. Building upon a recent variant of Helios, called Belenios [CGGI14], and a new malleable signature scheme that we introduce here (and which could be of independent interest), we propose a receipt-free version of Helios, which we call BeleniosRF.

Blazy *et al.* [BFPV11] introduced a primitive called *signatures on randomizable ciphertexts*. It consists of a signature scheme and a public-key encryption scheme that is randomizable (that is, given a ciphertext, anyone can create a fresh ciphertext of the same plaintext—without knowing it). The primitive provides an additional functionality: given a signature on a ciphertext, anyone can randomize the ciphertext and *adapt* the signature to the new ciphertext; that is, produce a signature that is valid on the new ciphertext; and all that *without* knowing neither decryption key nor signing key nor the plaintext. On the other hand, it is infeasible to compute a signature on a ciphertext that encrypts a message of which no encryption has been signed.

We adapt this primitive to propose a non-interactive[1] receipt-free e-voting scheme, which works as follows:

- As in Belenios, each voter is provided with a signature key pair, in addition to authentication means to the ballot box (typically a login and password).
- Each voter encrypts and signs his ballot. He additionally includes a proof of knowledge to prevent ballot malleability.
- Upon receiving a ballot, the ballot box re-randomizes the ballot and adapts the corresponding signature and proof before publishing the ballot.

Receipt-freeness comes from the fact that a voter no longer has control nor knowledge of the randomness used to form the final ballot stored in the ballot box. However, she can still verify that her vote is present, as the re-randomized ciphertext comes with a signature that is valid under her verification key. As she is the only one to know the signing key, by unforgeability of the signature primitive, the vote cannot have been altered by the ballot box.

We show that our scheme BeleniosRF is *strongly receipt-free* in the sense that even a dishonest voter cannot prove how he voted. We formalize this property via a game-based definition building on the ballot privacy definition recently proposed in [BCG+15], adapting it to the case of ballot submission by dishonest voters. In doing so we give a new formulation for the receipt-freeness definition by Benaloh and Tuinstra [BT94].

Interestingly, BeleniosRF inherits the simplicity of Helios and Belenios: the voting procedure is entirely unchanged compared to Belenios. Moreover, BeleniosRF accommodates revoting as well as no revoting (note that revoting is typically forbidden in real-world elections since it represents an important change w.r.t. traditional paper-based procedures).

Related work. Another receipt-free scheme has been recently proposed at Eurocrypt'15 [KZZ15]. However, its receipt-freeness should be understood w.r.t. honest voters that are under corruption after the ballot casting phase. Under this definition for example Helios is declared receipt-free. In contrast, we achieve receipt-freeness even w.r.t. to voters that act maliciously during the voting phase (i.e. that may be willing to run a specially crafted malicious software that provides a receipt for their vote).

We have already discussed Helios and Civitas. Other well-known and deployed schemes include Prêt-à-voter [RBH+09] and Scantegrity [CEC+08]. These systems however are designed for elections with physical voting booths. The system used in Estonia [SFD+14] as well as the one deployed in Norway [AC11, Gjø13] might possibly satisfy some level of receipt-freeness, as the corresponding ballot boxes are not publicly available. But, because of this, they do not achieve universal verifiability (in contrast to BeleniosRF).

[1]After a successful authentication between the voter and the ballot box, ballot casting is non-interactive.

Our scheme builds on previous work by Blazy et al. [BFPV11], who introduce the primitive of signatures on randomizable ciphertexts. The authors sketch a receipt-free e-voting scheme similar to ours, although no definition nor security reduction for receipt-freeness is given. In Section 5 we present a ballot-replay attack against their e-voting protocol, effectively showing that it does not provide ballot privacy. Our proposal for ballot casting is about $k+3$ times more efficient than the one from [BFPV11] in terms of space and computation, where k is the bit length of the vote v to be encrypted. This benefit comes at the cost of assuming a random oracle and mixnet-based tallying.

2 Receipt-Freeness

We now formally define receipt-freeness and start by providing the general syntax of a voting system, inspired by [CGGI14, BCG+15].

2.1 Syntax of a Voting System

Election systems typically involve several entities. For the sake of simplicity we consider each entity to consist of only one individual but note that all of them could be thresholdized.

1. *Election administrator*: denoted by $\mathcal{E}$, is responsible for setting up the election; it publishes the identities id of eligible voters, the list of candidates and the result function ρ of the election (typically counting the number of votes every candidate received).
2. *Registrar*: denoted by $\mathcal{R}$, is responsible for distributing secret credentials to voters and registering the corresponding public credentials.
3. *Trustee:* denoted by $\mathcal{T}$, is in charge of tallying and publishing a final result.
4. *Voters*: the eligible voters $id_1, \ldots, id_\tau$ are participating in the election.
5. *Ballot-box manager*: denoted by $\mathcal{B}$, is responsible for processing ballots and storing valid ballots in the ballot box BB; it is also responsible for publishing PBB, the public part of the ballot box, called bulletin board.

We continue by describing the syntax for an electronic voting protocol that we will be using thorough the paper. The syntax below considers *single-pass* schemes, that is, systems where voters only have to post a single message to the board. A voting protocol is always relative to a family of result functions $\{\rho_\tau\}_{\tau\geq 1}$ for $\tau \in \mathbb{N}$, with $\rho_\tau \colon \mathbb{V}^\tau \to \mathbb{R}$, where $\mathbb{V}$ is the set of admissible votes and $\mathbb{R}$ is the result space. A voting protocol $\mathcal{V} = (\mathsf{Setup}, \mathsf{Register}, \mathsf{Vote}, \mathsf{Valid}, \mathsf{Append}, \mathsf{Publish}, \mathsf{VerifyVote}, \mathsf{Tally}, \mathsf{Verify})$ consists of nine algorithms whose syntax is as follows:

$\mathsf{Setup}(1^\lambda)$, on input a security parameter 1^λ, outputs an election public/secret pair $(\mathbf{pk}, \mathbf{sk})$, where $\mathbf{pk}$ typically contains the public key of the election and/or a list of credentials L. We assume $\mathbf{pk}$ to be an implicit input of the remaining algorithms.

$\mathsf{Register}(1^\lambda, id)$, on inputs a security parameter 1^λ and an identifier id, outputs the secret part of the credential usk_{id} and its public credential upk_{id}, which is added to the list $L = \{\mathsf{upk}_{id}\}$.

$\mathsf{Vote}(id, \mathsf{upk}, \mathsf{usk}, v)$ is used by voter id to cast his choice $v \in \mathbb{V}$. It outputs a ballot b, which may or may not include the identifier id or the public credential upk. The ballot b is sent to the bulletin board through a (possibly authenticated) channel.

$\mathsf{Valid}(\mathsf{BB}, b)$ takes as input the ballot box BB and a ballot b and checks its validity. It returns $\top$ for valid ballots and $\perp$ for invalid ones (ill-formed, containing duplicated ciphertext from the ballot box, etc.).

$\mathsf{Append}(\mathsf{BB}, b)$ updates the ballot box with the ballot b. Typically, this consists in adding b as a new entry to BB, but more involved actions might be possible (as in the case of our scheme).

$\mathsf{Publish}(\mathsf{BB})$ takes as input the ballot box BB and outputs the public view PBB of BB, called public bulletin board.

$\mathsf{VerifyVote}(\mathsf{PBB}, id, \mathsf{upk}, \mathsf{usk}, b)$ is a typically light-weight algorithm run by voters, for checking that their ballots will be included in the tally. On inputs the public board PBB, a ballot b, and the voter's identity and credentials $id, \mathsf{usk}, \mathsf{upk}$, it returns $\top$ or $\perp$.

$\mathsf{Tally}(\mathsf{BB}, \mathbf{sk})$ takes as input the ballot box BB and the secret key $\mathbf{sk}$. After some checks, it outputs the tally r, together with a proof of correct tabulation Π. Possibly, $r = \perp$, meaning the election was declared invalid.

$\mathsf{Verify}(\mathsf{PBB}, r, \Pi)$, on inputs the public bulletin board PBB and a pair (r, Π), checks whether Π is a valid proof of correct tallying for r. It returns $\top$ if so; otherwise it returns $\perp$.

The exact implementation of these algorithms of course depends on the voting protocol under consideration. In particular, the notion of public and private credentials of a voter varies a lot depending on the protocol. For example upk_{id} might be simply the identity of the voter or it may correspond to his signature-verification key.

2.2 Strong Receipt-Freeness

Intuitively, privacy ensures that an adversary cannot learn the vote of an honest voter. In contrast, receipt-freeness further guarantees that a voter cannot prove how he voted, even if he willingly provides information to the adversary. This captures the seminal intuition from Benaloh and Tuinstra [BT94]. Indeed, the latter insisted that a reasonably private electronic voting protocol shall emulate traditional voting in a voting booth: it should *allow* voters to conceal their individual votes and, at the same time, it should *prevent* them from revealing their vote. Namely, voters should not be able to give away the privacy of their vote granted by the voting protocol, even if they are willing to do so.

Building upon a definition of privacy recently introduced in [BCG+15], we simply provide the adversary with an additional oracle (called $\mathcal{O}\mathsf{receiptLR}$) that allows him to submit his own ballots on behalf of any voter (honest or dishonest). This simple formalization covers several important scenarios:

- A voter who wants to convince a vote buyer of how he voted may prepare his ballot in an arbitrary way that allows him to construct a convincing receipt (e.g., consider the scenario where the voter uses biased random coins to build his ballot and to prove how he voted [GGR09]).
- A voter that might have been corrupted before the ballot casting phase may just follow the instructions given to him by the adversary ([JCJ05]).
- A voter can record, but also forge, its interaction with the ballot box (as in [BT94]).

Formally, we consider two games, Game 0 and Game 1, defined by the oracles in Figure 1. In both games $\mathsf{BB}_0, \mathsf{BB}_1$ are ballot boxes that start out empty. Box BB_0 corresponds to the real election (that will be tallied) and BB_1 is the fake ballot box with which the adversary interacts in the second game. In Game β the adversary has indirect access to BB_β, that is, he may see the public part of the box at any time. The game provides an adversary $\mathcal{A}$ access to the oracles defined in Figure 1, which intuitively proceed as follows:

$\mathcal{O}\mathsf{init}$: This oracle generates secret and public keys for the election; the public key is returned to the adversary.

$\mathcal{O}\mathsf{board}$: This models the adversary's ability to see the publishable part of the board. The oracle returns $\mathsf{Publish}(\mathsf{BB}_\beta)$. (In many schemes, we simply have $\mathsf{Publish}(\mathsf{BB}_\beta) = \mathsf{BB}_\beta$.)

```
$\mathcal{O}\mathsf{init}$
    $(\mathbf{pk}, \mathbf{sk}) \leftarrow \mathsf{Setup}(1^k)$; return $\mathbf{pk}$

$\mathcal{O}\mathsf{reg}(id)$
    If $id$ was not previously queried,
    then run $\mathsf{Register}(\lambda, id)$ and
    update $\mathcal{U} = \mathcal{U} \cup \{(id, \mathsf{upk}_{id}, \mathsf{usk}_{id})\}$;
    return $\mathsf{upk}_{id}$ and keep $\mathsf{usk}_{id}$ secret.

$\mathcal{O}\mathsf{corruptU}(id)$
    on a registered voter $id$, output
    the credential pair $(\mathsf{upk}_{id}, \mathsf{usk}_{id})$
    and update $\mathcal{CU} = \mathcal{CU} \cup$
    $\{(id, \mathsf{upk}_{id})\}$.

$\mathcal{O}\mathsf{voteLR}(id, v_0, v_1)$
    If $v_0 \notin \mathbb{V}$ or $v_1 \notin \mathbb{V}$ then return $\perp$.
    $b_0 = \mathsf{Vote}(id, \mathsf{upk}_{id}, \mathsf{usk}_{id}, v_0)$
    $b_1 = \mathsf{Vote}(id, \mathsf{upk}_{id}, \mathsf{usk}_{id}, v_1)$.
    $\mathsf{Append}(\mathsf{BB}_0, b_0)$; $\mathsf{Append}(\mathsf{BB}_1, b_1)$

$\mathcal{O}\mathsf{cast}(id, b)$
    If $\mathsf{Valid}(\mathsf{BB}_\beta, b) = \perp$ then return $\perp$.
    Else $\mathsf{Append}(\mathsf{BB}_0, b)$ and $\mathsf{Append}(\mathsf{BB}_1, b)$.

$\mathcal{O}\mathsf{receiptLR}(id, b_0, b_1)$
    If $id \notin \mathcal{CU}$ return $\perp$.
    If $\mathsf{Valid}(\mathsf{BB}_0, b_0) = \perp$ or $\mathsf{Valid}(\mathsf{BB}_1, b_1) = \perp$
    return $\perp$.
    Else $\mathsf{Append}(\mathsf{BB}_0, b_0)$ and $\mathsf{Append}(\mathsf{BB}_1, b_1)$

$\mathcal{O}\mathsf{board}()$
    return $\mathsf{Publish}(\mathsf{BB}_\beta)$

$\mathcal{O}\mathsf{tally}()$ for $\beta = 0$
  $(r, \Pi)$
      $\leftarrow \mathsf{Tally}(\mathsf{BB}_0, \mathbf{sk})$
  return $(r, \Pi)$

$\mathcal{O}\mathsf{tally}()$ for $\beta = 1$
  $(r, \Pi) \leftarrow \mathsf{Tally}(\mathsf{BB}_0, \mathbf{sk})$
  $\Pi' \leftarrow \mathsf{SimProof}(\mathsf{BB}_1, r)$
  return $(r, \Pi')$
```

Figure 1: Oracles defining experiments $\mathsf{Exp}^{\mathsf{srf},\beta}_{\mathcal{A},\mathcal{V}}(\lambda)$ for $\beta = 0, 1$. The games differ in the way the tallying oracle creates auxiliary data, in which board is displayed to the adversary in response to $\mathcal{O}\mathsf{board}$ queries and against which board ballots are validated.

$\mathcal{O}\mathsf{voteLR}$: The left-or-right oracle $\mathcal{O}\mathsf{voteLR}$ takes two potential votes (v_0, v_1) for an honest user id, produces ballots b_0 and b_1 for these votes and places them in the ballot box (one in BB_0 and one in BB_1), provided that $v_0, v_1 \in \mathbb{V}$.

$\mathcal{O}\mathsf{cast}$: This oracle allows the adversary to cast a ballot b on behalf of any party. If the ballot is valid with respect to BB_β, it is placed in both ballot boxes.

$\mathcal{O}\mathsf{receiptLR}$: This oracle allows an adversarial voter id to cast a ballot b_1 in BB_1 and a ballot b_0 in BB_0. If each ballot b_0, b_1 is valid with respect to their respective ballot boxes, then the ballots are appended to the corresponding boxes by running $\mathsf{Append}(\mathsf{BB}_0, b_0)$ and $\mathsf{Append}(\mathsf{BB}_1, b_1)$. This allows the adversary to encode special instructions in the ballots that could later serve as the basis for a vote receipt.

$\mathcal{O}\mathsf{tally}$: This oracle allows the adversary to see the result of the election. In both games the result is obtained by tallying a valid BB_0; the additional information is however simulated in the second world.

We demand that the adversary first calls $\mathcal{O}\mathsf{init}$, then oracles $\mathcal{O}\mathsf{voteLR}, \mathcal{O}\mathsf{cast}, \mathcal{O}\mathsf{board}, \mathcal{O}\mathsf{receiptLR}$ in any order, and any number of times. Finally, $\mathcal{A}$ can call $\mathcal{O}\mathsf{tally}$; after it receives the answer to this, $\mathcal{A}$ must return a guess of the bit β. The guess bit is the result returned by the game.

Definition 1 (sRF). *Consider a voting protocol* $\mathcal{V}$ = (Setup, $\mathsf{Register}$, Vote, Valid, Append, $\mathsf{VerifyVote}$, $\mathsf{Publish}$, Tally, Verify) *for a set ID of voter identities and a result function* ρ. *We say the scheme has strong receipt-freeness if there exists an algorithm* $\mathsf{SimProof}$ *such that no efficient adversary can distinguish between games* $\mathsf{Exp}^{\mathsf{srf},0}_{\mathcal{B},\mathcal{V}}(\lambda)$ *and* $\mathsf{Exp}^{\mathsf{srf},1}_{\mathcal{B},\mathcal{V}}(\lambda)$ *defined by the oracles in Figure 1, that is, for any efficient algorithm* $\mathcal{A}$

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

is negligible in λ.

As expected, strong receipt-freeness[2] (sRF) trivially implies privacy (as defined by the BPRIV definition in [BCG+15]). This is because BPRIV is defined as strong receipt-freeness except that the adversary does not have access to $\mathcal{O}\mathsf{receiptLR}$. We call this notion *strong* receipt-freeness to distinguish it from the notion of receipt-freeness recently introduced in [KZZ15], which accounts only for honest voters: having (honestly) voted, a voter should not be able to provide a receipt to the adversary. Let us now consider receipt-freeness of two voting protocols.

Helios. Under the definition provided in [KZZ15] the Helios protocol would be declared receipt-free. In contrast, under our definition Helios is not receipt-free. Indeed, if the adversary is allowed to cast different ballots b_0, b_1 to the ballot boxes $\mathsf{BB}_0, \mathsf{BB}_1$ respectively, then distinguishing Game 0 from Game 1 is trivial. This is due to the fact that in Helios PBB contains the encryption of the votes under the public key of the election, so it suffices for an adversary to produce different encryptions c, d of the same vote and to see which one is showing up when calling oracle $\mathcal{O}\mathsf{board}$.

Civitas. Maybe surprisingly, Civitas [CCM08] is not strongly receipt-free either. Indeed, as for Helios, if the adversary is allowed to cast different ballots b_0, b_1 to the ballot boxes $\mathsf{BB}_0, \mathsf{BB}_1$ respectively, then distinguishing Game 0 from Game 1 is trivial. This reflects the fact that a voter may prove to a third party that, if his ballot is to be included in the final tally, then it corresponds to some candidate chosen by the adversary.

The fact that Civitas still enjoys coercion-resistance relies on the fact that the voter *has a strategy* to fake a credential and fool his coercer with a ballot that will be accepted in the voting phase but discarded in the tally phase. Depending on the voter's abilities, it is not always possible to follow the intended strategy. For example, if re-voting is forbidden (as is usually the case in practice) then Civitas is no longer coercion-resistant, as the only way a voter can defeat the coercer is by casting a ballot that will not be part of the final tally. So if a coerced voter wants to vote in the election and re-voting is not possible then the voter must abide to the choice made by the coercer, even if the coercer is not present at the time of ballot casting.

In contrast, our notion of strong receipt-freeness simply captures whether an advesary can detect whether a voter runs the honest program or the adversary's program. This scenario gives less power to the voter, but it is also more realistic with respect to what an average voter can understand and perform.

3 Building Blocks

Before describing our voting scheme, we first present the necessary cryptographic building blocks.

3.1 Assumptions

We will work in symmetric bilinear groups and assume the existence of a *bilinear-group generator* GrpGen, which on input 1^λ outputs $(p, \mathbb{G}, \mathbb{G}_T, e, g)$, where p is a prime of length λ, $\mathbb{G}$ and $\mathbb{G}_T$ are cyclic groups of order p, g is a generator of $\mathbb{G}$, and e is a bilinear map $e\colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ such that $e(g, g)$ generates $\mathbb{G}_T$.

Definition 2 (CDH)**.** *The computational Diffie-Hellman assumption holds for* GrpGen *if for all* $\mathcal{G} = (p, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow_{\$} \mathsf{GrpGen}(1^\lambda)$*, for* $a, b \leftarrow_{\$} \mathbb{Z}_p$*, and every probabilistic polynomial-time (p.p.t.) adversary given* $(\mathcal{G}, g^a, g^b)$*, the probability that it outputs* g^{ab} *is negligible in* λ*.*

[2]Let us emphasize that the sRF definition requires some setup assumptions (like the CRS or the RO model), which we do not include in this section for simplicity. We refer to Appendix C in [BCG+15] for an explicit RO model setup.

Definition 3 (DLIN)**.** *The decisional linear assumption holds for* GrpGen *if for* $\mathcal{G} = (p, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow_\$ \mathsf{GrpGen}(1^\lambda)$ *and for* $x, y, a, b, c \leftarrow_\$ \mathbb{Z}_p$*, no p.p.t. adversary can distinguish* $(\mathcal{G}, g^x, g^y, g^{xa}, g^{yb}, g^{a+b})$ *from* $(\mathcal{G}, g^x, g^y, g^{xa}, g^{yb}, g^c)$ *with advantage non-negligible in* λ*.*

3.2 Signatures on Randomizable Ciphertexts

The primitive introduced by Blazy et al. [BFPV11] consists of the following algorithms: Setup, which on input the security parameter 1^λ outputs the parameters (such as the bilinear group); $\mathsf{SKeyGen}$ outputs a pair of signing key and verification key (sk, vk), $\mathsf{EKeyGen}$ outputs a pair of encryption and decryption key (pk, dk). $\mathsf{SKeyGen}$ together with Sign and Verify constitutes a signature scheme and $\mathsf{EKeyGen}$ with $\mathsf{Encrypt}$ and $\mathsf{Decrypt}$ a public-key encryption scheme.

As the signature and the encryption scheme are used together, these algorithms have extensions Sign^+ and Verify^+, which additionally take the encryption key pk as input; and $\mathsf{Encrypt}^+$, $\mathsf{Decrypt}^+$, which also take the verification key vk.

Randomizability. The main feature of signatures on randomizable ciphertexts (SRC) is an algorithm Random^+, which takes pk, vk, a ciphertext c under pk and a signature σ on c valid under vk, and outputs a re-randomization c' of c together with a signature σ', valid on c'.

We require that outputs of Random^+ are distributed like a fresh encryption of the plaintext of c and a signature on it: For all $(pk, dk) \leftarrow_\$ \mathsf{EKeyGen}(\mathcal{G})$, $(vk, sk) \leftarrow_\$ \mathsf{SKeyGen}(\mathcal{G})$ and messages m the following two random variables are distributed equivalently:

$$\left[\begin{array}{l} c \leftarrow_\$ \mathsf{Encrypt}^+(pk, vk, m); \\ \sigma \leftarrow_\$ \mathsf{Sign}^+(sk, pk, c); \end{array} : (c, \sigma)\right] \approx_c \left[\begin{array}{l} c \leftarrow_\$ \mathsf{Encrypt}^+(pk, vk, m); \\ \sigma \leftarrow_\$ \mathsf{Sign}^+(sk, pk, c); \\ (c', \sigma') \leftarrow_\$ \mathsf{Random}^+(pk, vk, c, \sigma) \end{array} : (c', \sigma')\right] \quad (1)$$

Unforgeability. Unforgeability of signatures on randomizable ciphertext is defined via the following experiment: The challenger computes a signature key pair and an encryption key pair (sk, vk), (dk, pk) and runs the adversary on (vk, pk). It is also given access to an oracle $\mathsf{Sign}^+(sk, pk, \cdot)$, which it can query adaptively on ciphertexts $c_1, \ldots, c_q$ of its choice. Finally, it outputs a pair (c^*, σ^*) and wins if $\mathsf{Verify}^+(vk, pk, c^*, \sigma^*) = 1$ and $m = \mathsf{Decrypt}^+(dk, vk, c^*)$ is different from all $m_i := \mathsf{Decrypt}^+(dk, vk, c_i)$ and $m, m_1, \ldots, m_q \neq \perp$.

We now construct an efficient scheme for signatures on randomizable ciphertexts based on *Boneh-Lynn-Shacham signatures* [BLS04] and *linear encryption* [BBS04]. We note that our scheme is not suitable for homomorphic tallying, since we encrypt $H(m)$, where H is hash function modeled as a random oracle, so the scheme is not additively homomorphic.

3.3 Our SRC Construction

Linear encryption. Our construction of signatures on randomizable ciphertexts uses linear encryption, which is semantically secure under DLIN (Definition 3). For simplicity, we give the scheme in symmetric bilinear groups, but note that it could be transferred to (more efficient) asymmetric groups, by giving certain values in both groups $\mathbb{G}_1$ and $\mathbb{G}_2$.

Linear encryption is defined over a bilinear group $(p, \mathbb{G}, \mathbb{G}_T, e, g)$ and secret keys (x, y) are drawn from $\mathbb{Z}_p^2$; they define a public key in $\mathbb{G}^2$ as $X = g^x, Y = g^y$. A message $M \in \mathbb{G}$ is encrypted by choosing $r, s \leftarrow_\$ \mathbb{Z}_p$ and returning $c = (X^r, Y^s, M \cdot g^{r+s})$. Decryption is done by computing $M = c_3 \cdot c_1^{-1/x} \cdot c_2^{-1/y}$.

We now extend the scheme to be combined with BLS signatures, which we introduce below. We will use three hash functions $H \colon \{0,1\}^{\log \lambda} \to \mathbb{G}$, and $I, J \colon \mathbb{G} \to \mathbb{G}$, which will be modeled as random oracles. Our scheme only supports polynomial-size message spaces $\mathcal{M}$, which suffices perfectly when encrypting votes. We can thus define encryption of a message m as linear encryption of $M = H(m)$ and decrypt by decrypting M and then performing an exhaustive search on the message space to recover m. Encryption moreover adds "tags" which depend on the verification key vk (which is an additional input to $\mathsf{Encrypt}^+$); this will ensure a form of non-malleability, which in our voting protocol will bar vote-copying.

$\mathsf{EKeyGen}(\mathcal{G})$: Choose $dk = (x, y) \leftarrow_\$ \mathbb{Z}_p^2$ and define $pk = (X, Y) = (g^x, g^y)$.

$\mathsf{Encrypt}^+((X, Y), vk, m; r, s)$: Return $c_1 = X^r$, $c_2 = Y^s$, $c_3 = H(M) \cdot g^{r+s}$, $c_4 = I(vk)^r$ and $c_5 = J(vk)^s$.

$\mathsf{Decrypt}^+((x, y), vk, (c_1, c_2, c_3, c_4, c_5))$: If $e(c_4, X) \neq e(I(vk), c_1)$ or $e(c_5, Y) \neq e(J(vk), c_2)$, return $\perp$. Else compute $M := c_3 \cdot c_1^{-1/x} \cdot c_2^{-1/y}$, search the message space $\mathcal{M}$ and output m with $M = H(m)$.

BLS signatures on ciphertexts. BLS signatures [BLS04] consist of one bilinear-group element and are secure under CDH (Definition 2) in the random-oracle model. Given a bilinear group $(p, \mathbb{G}, g, \mathbb{G}_T, e)$, a secret key is an element $z \in \mathbb{Z}_p$ and defines the verification key as $Z = g^z$. A signature on a message m is defined as $H(m)^z$, where H is a random oracle. A signature σ is verified by checking $e(\sigma, g) = e(H(m), Z)$. We now define a signature on a linear ciphertext $c = (X^r, Y^s, H(m) \cdot g^{r+s})$ as $\sigma = (c_1^z, c_2^z, c_3^z)$. This yields $(X^{rz}, Y^{sz}, H(m)^z \cdot g^{rz+sz})$, which is a linear encryption (using randomness (rz, sz)) of a signature $H(m)^z$. In order to allow for re-randomization, we add two more elements, namely X^z and Y^z to the signature.

$\mathsf{SKeyGen}(\mathcal{G})$: Choose $sk = z \leftarrow_\$ \mathbb{Z}_p$ and define $vk = Z = g^z$.

$\mathsf{Sign}^+(z, (X, Y), (c_1, \ldots, c_5))$: If $e(c_4, X) \neq e(I(g^z), c_1)$ or $e(c_5, Y) \neq e(J(g^z), c_2)$, return $\perp$. Else return $\sigma_1 = c_1^z$, $\sigma_2 = c_2^z$, $\sigma_3 = c_3^z$, $\sigma_4 = X^z$, $\sigma_5 = Y^z$.

$\mathsf{Verify}^+(Z, (X, Y), (c_1, \ldots, c_5), (\sigma_1, \ldots, \sigma_5))$:

– check consistency of c: If $e(c_4, X) \neq e(I(Z), c_1)$ or $e(c_5, Y) \neq e(J(Z), c_2)$, return 0;
– check σ: if $\exists i \in \{1, 2, 3\}$: $e(\sigma_i, g) \neq e(c_i, Z)$; or $e(\sigma_4, g) \neq e(X, Z)$ or $e(\sigma_5, g) \neq e(Y, Z)$ then return 0; else return 1.

Given a ciphertext c under (X, Y) for verification key Z and a signature σ on c under Z then the pair (c, σ) can be randomized together:

$\mathsf{Random}^+((X, Y), Z, c, \sigma; (r', s'))$: Return $c_1' = c_1 X^{r'}$, $c_2' = c_2 Y^{s'}$, $c_3' = c_3 g^{r'+s'}$, $c_4' = c_4 I(Z)^{r'}$, $c_5' = c_5 J(Z)^{s'}$, $\sigma_1' = \sigma_1 \sigma_4^{r'}$, $\sigma_2' = \sigma_2 \sigma_5^{s'}$, $\sigma_3' = \sigma_3 Z^{r'+s'}$, $\sigma_4' = \sigma_4$, $\sigma_5' = \sigma_5$.

It is easily verified that this yields a signed ciphertext of the same message, but using randomness $(r + r', s + s')$. Our scheme has thus randomizability, as defined in Equation (1).

We now show that our scheme satisfies the unforgeability notion for SRC from [BFPV11], as defined in Section 3.2.

Theorem 1. *The signature scheme* (Setup, $\mathsf{EKeyGen}$, $\mathsf{SKeyGen}$, $\mathsf{Encrypt}^+$, $\mathsf{Decrypt}^+$, Sign^+, Verify^+, Random^+) *above is unforgeable in the random-oracle model.*

Proof. The proof is by reduction to unforgeability of BLS signatures. The simulator is given $vk = Z = g^z$ and access to a signing oracle (which on input m returns $H(m)^z$). It chooses $x, y \leftarrow_\$ \mathbb{Z}_p$ and runs $\mathcal{A}$ on input $Z, (X = g^x, Y = g^y)$. Random oracles queries to I and J are answered by choosing random values, except $I(Z) := Z^a$ and $J(Z) := Z^b$ for random $a, b \leftarrow_\$ \mathbb{Z}_p$.

Sign^+ oracle queries for c are answered as follows: if the checks of c_4 and c_5 fail then return $\perp$; else there exist r, s and M (unknown to the reduction) such that $c = (X^r, Y^s, M \cdot g^{r+s}, I(Z)^r, J(Z)^s)$; note that by the programming of I and J we have $c_4 = Z^{ar}$ and $c_5 = Z^{bs}$. The simulator computes $M = c_3 c_1^{-1/x} c_2^{-1/y}$, searches m in $\mathcal{M}$ with $M = H(m)$ and if no such m

exists, it aborts (note that in this case c is not a valid ciphertext and $\perp = \mathsf{Decrypt}^{+}(dk, vk, c)$, thus the adversary loses the game). Otherwise, the simulator queries m to its signing oracle to obtain $\hat{\sigma} = H(m)^z$. It then computes σ as $\sigma_1 = c_4^{x/a} = X^{rz}$, $\sigma_2 = c_5^{y/b} = Y^{sz}$, $\sigma_3 = \hat{\sigma} c_4^{1/a} c_5^{1/b}$, $\sigma_4 = Z^x$, $\sigma_5 = Z^y$, which is easily seen to be correctly distributed.

Suppose the adversary outputs (c^*, σ^*) such that $\mathsf{Verify}^{+}(Z, (X, Y), c^*, \sigma^*) = 1$. Let $m^* = \mathsf{Decrypt}^{+}(dk, vk, c^*)$, thus for some r^*, s^*, we have $c_1^* = X^{r^*}$, $c_2^* = Y^{s^*}$, $c_3^* = H(m^*) g^{r^*+s^*}$. Moreover, by Verify^{+}, we have $\sigma_1^* = X^{r^* z}$, $\sigma_2^* = Y^{s^* z}$, $\sigma_3^* = H(m^*)^z g^{r^* z + s^* z}$; thus $\sigma_3^* (\sigma_1^*)^{-1/x} \cdot (\sigma_2^*)^{-1/y} = H(m^*)^z$, which is a signature on m^*, which was not queried and is thus a valid forgery. □

Remark 1. Note that unforgeability also holds, when the adversary's forgery need only be of the form (c_1, c_2, c_3) and $(\sigma_1, \sigma_2, \sigma_3)$ (that is, the components $I(Z)^r$ and $J(Z)^s$ in c; and X^z and Y^z in σ are not required). This will make our voting protocol more efficient, as publishing these elements suffices to convince the voter that his vote was counted.

3.4 RCCA-Secure Encryption from SRC

As a next step in the development of our voting protocol in this section we show that our SRC scheme from the last section yields an RCCA-secure encryption scheme.

CCA-security, the standard notion for public-key encryption, states that for an efficient adversary that after choosing m_0, m_1 receives c, it should be impossible to decide whether c encrypts m_0 or m_1, even when given an oracle that decrypts any ciphertext $c' \neq c$. For randomizable schemes this notion is unachievable, as the adversary could simply submit a randomization of the challenge ciphertext to the decryption oracle. The strongest achievable notion for such schemes is RCCA, where every time the oracle receives an encryption of m_0 or m_1, it returns a special symbol $\top$.

Based on our SRC scheme we define the following encryption scheme:

$\overline{\mathsf{KeyGen}}$ is defined as $\mathsf{EKeyGen}$.

$\overline{\mathsf{Encrypt}}(pk, m)$: Choose $(vk, sk) \leftarrow_{\$} \mathsf{SKeyGen}(\mathcal{G})$;
compute $c \leftarrow_{\$} \mathsf{Encrypt}^{+}(pk, vk, m)$;
set $\sigma \leftarrow_{\$} \mathsf{Sign}^{+}(sk, pk, c)$ and return $C = (c, \sigma, vk)$.

$\overline{\mathsf{Decrypt}}(dk, C)$: If $\mathsf{Verify}^{+}(vk, pk, c, \sigma) = 0$ then return $\perp$;
else return $m = \mathsf{Decrypt}^{+}(dk, vk, c)$.

Theorem 2. *The above scheme is RCCA-secure.*

Proof. We distinguish two types of forgers: *Type 1* at some point issues a valid decryption query (that is, one that does not return $\perp$ or $\top$) which contains vk that is also contained in the challenge ciphertext. This is reduced to unforgeability of the underlying SRC scheme. The simulator receives vk, encrypts m_b as c and queries c to its Sign^{+} oracle to obtain σ (note that the ciphertext contains a valid message) and sets the challenge ciphertext as (c, σ, vk). If the adversary issues a valid decryption query containing vk but for a message different from m_0 and m_1, then the contained signature σ is a forgery.

Type 2 forgers never issue a valid decryption query containing the vk from the challenge ciphertext. This case is reduced to the decision linear assumption. Given a DLIN instance $(X, Y, R = X^r, S = Y^s, T)$, where either $T = g^{r+s}$ or T is random, the simulator defines $pk = (X, Y)$ and picks $z \leftarrow_{\$} \mathbb{Z}_p$ which it will use in the challenge query. It programs the random oracles I, J as follows: when queried on $Z = g^z$, it defines $I(Z) = X^a$ and $J(Z) = Y^b$ for some $a, b \leftarrow_{\$} \mathbb{Z}_p$; for any $Z_i \neq Z$, it chooses a_i or b_i and sets $I(Z_i) = g^{a_i}$ and $J(Z_i) = g^{b_i}$.

Decryption queries for (c, σ, vk) are answered as follows: if $\mathsf{Verify}^{+}(vk, pk, c, \sigma) = 0$ then return $\perp$; else browse the random-oracle queries to find i such that $Z_i = vk$. Let r, s and M

(unknown to the reduction) be such that $c_1 = X^r, c_2 = Y^s, c_3 = Mg^{r+s}$. By verification, we have $c_4 = I(Z_i)^r, c_5 = J(Z_i)^s$, thus $c_4 = g^{a_i r}, c_5 = g^{b_i s}$. The simulator computes $M = c_3 c_4^{-1/a_i} c_5^{-1/b_i}$, looks for m s.t. $H(m) = M$ and returns m (and $\perp$ if no m exists). The challenge ciphertext C is computed as follows: $\big(c = (R, S, H(m_b)\,T, R^a = I(Z)^r, S^b = J(Z)^s), \sigma = (c_1^z, c_2^z, c_3^z, X^z, Y^z), Z\big)$.

If $T = g^{r+s}$ then this game simulates the RCCA game for bit b; however, if T is random then the game is independent of b. Since under DLIN these two games are indistinguishable, the adversary cannot win the RCCA game with non-negligible probability. $\square$

3.5 Verifiable Secret Shuffle of Linear Encryptions

Since our scheme does not support homomorphic tallying, we need a zero-knowledge argument for proving the correctness of a shuffle of linear encryptions. Roughly speaking, on input a list of linear encryptions $\{(X^{r_i}, Y^{s_i}, m_i \cdot g^{r_i+s_i})\}$ for $i = 1, \ldots, n$, a shuffle outputs a list of linear encryptions of the form $\{(X^{r'_i}, Y^{s'_i}, m_{\mathcal{P}(i)} \cdot g^{r'_i+s'_i})\}$, where $\mathcal{P} \leftarrow_\$ \mathsf{Permutation}(1, \ldots, n)$. We need this output to be verifiable and permutation-hiding.

Definition 4. *Let* $(\mathsf{ShufflePermute}, \mathsf{ShuffleVerify})$ *be a shuffle scheme. We say it is verifiable and secret if:*

Verifiability: *let the input of* $\mathsf{ShufflePermute}$ *be a list of linear encryptions* $L_{in} = \{(c_{i,1}, c_{i,2}, c_{i,3})\}$ *of plaintexts* m_i*, and let the output of* $\mathsf{ShufflePermute}$ *be a list of linear encryptions* $L_{out} = \{(c'_{i,1}, c'_{i,2}, c'_{i,3})\}$ *of plaintexts* m'_i*, for* $i = 1, \ldots, n$*, and a proof of correct shuffling* Π_s*. Then with overwhelming probability there exists a permutation* $\mathcal{P}$ *such that, as multisets,* $\{m'_1, \ldots, m'_n\} = \{m_{\mathcal{P}(1)}, \ldots, m_{\mathcal{P}(n)}\}$.

Permutation-hiding: *a simulator* $\mathsf{ShuffleSim}$ *exists such that on inputs two lists of encryptions* L_{in} *and* L_{out} *and any permutation* $\mathcal{P} \in \mathsf{Permutation}(1, \ldots, n)$*, outputs a simulated shuffle proof* Π_s^{sim} *such that* $\mathsf{ShuffleVerify}(pk, L_{in}, L_{out}, \Pi_s^{sim})$ *accepts.*

Such a shuffle of linear encryptions can be obtained, for instance, by extending the shuffle scheme for ElGamal encryptions proposed by Bayer and Groth in [BG12].

4 BeleniosRF: Strongly Receipt-Free E-Voting

In this section we define Belenios Receipt-Free (BeleniosRF), a strongly receipt-free voting protocol that builds on [BFPV11] and [CGGI14]. We start with an overview of our protocol, which is based on the primitive instantiated in the last section. The election public/secret key pair $(\mathbf{pk}, \mathbf{sk})$ is an encryption/decryption key pair generated via $\mathsf{EKeyGen}$ and user key pairs $(\mathsf{upk}, \mathsf{usk})$ are signature keys generated by $\mathsf{SKeyGen}$. A user casts a vote by encrypting it via $\mathsf{Encrypt}^+$ w.r.t. $\mathbf{pk}$ and his upk and uses usk to then sign the ciphertext via Sign^+ (together, this corresponds to a ciphertext output by $\overline{\mathsf{Encrypt}}$ of our RCCA encryption scheme from Section 3.4).

When the ballot box receives a valid bulletin, it randomizes it via Random^+ and publishes the resulting ciphertext/signature pair on the public bulletin board PBB. Users can verify that their vote is present, since they can verify the adaptation of their signature, which is valid on their randomized ciphertexts. Tallying now follows standard techniques of e-voting: First, the encrypted votes are re-randomized and shuffled (and a proof of correct execution of this is generated) via an algorithm $\mathsf{Shuffle}$; finally, the ballots are decrypted (again accompanied with a proof that this was done correctly) and the result is published. These proofs make the tallying process publicly verifiable. Let us now give the details.

The scheme BeleniosRF $\mathcal{V}^{\mathsf{BeleniosRF}}$ consists of the algorithms (Setup, $\mathsf{Register}$, Vote, Valid, Append, $\mathsf{VerifyVote}$, $\mathsf{Publish}$, Tally, Verify), defined as follows:

$\mathsf{Setup}(1^\lambda)$: Run $\mathcal{G} = (p, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow_\$ \mathsf{GrpGen}(1^\lambda)$, choose hash functions $H, I, J \colon \{0,1\}^* \to \mathbb{G}$ and set $\mathsf{param} := (\mathcal{G}, H, I, J)$. Pick $x, y \leftarrow_\$ \mathbb{Z}_p$ and define $X := g^x, Y := g^y$. Set $pk := (X, Y)$,

$dk := (x, y)$ and output $\mathbf{pk} \leftarrow (\mathcal{G}, pk, H, I, J, \mathbb{V})$, the public key of the election (which is an implicit input to all following algorithms), and $\mathbf{sk} = (\mathbf{pk}, dk)$, the secret key of the election.

$\mathsf{Register}(id)$: On (implicit) input $(\mathcal{G}, pk, H, I, J, \mathbb{V})$ and an identifier id, choose a secret key $\mathsf{usk}_{id} = z \leftarrow_{\$} \mathbb{Z}_p$, and define the public key as $\mathsf{upk}_{id} = Z = g^z$.

$\mathsf{Vote}\,(id, \mathsf{upk} = Z, \mathsf{usk} = z, v)$ is used by a voter with secret key z to create a ballot b for vote $v \in \mathbb{V}$ as follows:

- encrypt v w.r.t. (X, Y) and Z by sampling $r, s \leftarrow_{\$} \mathbb{Z}_p$ and setting $c \in \mathbb{G}^5$ as follows:

$$c_1 = X^r \qquad c_2 = Y^s \qquad c_3 = H(v) \cdot g^{r+s} \qquad c_4 = I(Z)^r \qquad c_5 = J(Z)^s$$

- sign c, using $z \in \mathbb{Z}_p$, computing $\sigma := \big(\sigma_1 := c_1^z,\ \sigma_2 := c_2^z,\ \sigma_3 := c_3^z,\ \sigma_4 := X^z,\ \sigma_5 := Y^z\big)$.

The ballot is defined as $b = (id, Z, c, \sigma)$. The voter casts the ballot b to the ballot box BB after a successful authentication. The ballot-box manager runs $\mathsf{Append}(\mathsf{BB}, b)$ if $\mathsf{Valid}(\mathsf{BB}, b) \neq \bot$.

$\mathsf{Valid}(\mathsf{BB}, b)$ first checks that the ballot b is *valid*, i.e., that it is well-formed and the signature is correct. Formally, it parses b as (id, Z, c, σ) and checks whether

- id corresponds to an eligible voter;
- Z corresponds to upk of an eligible voter;
- c was encrypted w.r.t. Z by testing whether $e(c_4, X) = e(I(Z), c_1)$ and $e(c_5, Y) = e(J(Z), c_2)$;
- σ is valid, i.e., $e(\sigma_1, g) = e(c_1, Z)$, $e(\sigma_2, g) = e(c_2, Z)$, $e(\sigma_3, g) = e(c_3, Z)$, $e(\sigma_4, g) = e(X, Z)$ and $e(\sigma_5, g) = e(Y, Z)$.

If any step fails, it returns $\bot$; otherwise, it returns $\top$.

$\mathsf{Append}(\mathsf{BB}, b)$ proceeds as follows:

- id corresponds to an eligible voter;
- id has not cast a ballot before.[3]

If both checks are OK, it updates BB with a fully randomized version of $b = (id, Z, c, \sigma)$ as $b' = (id, Z, c', \sigma')$, computed by choosing $r', s' \leftarrow_{\$} \mathbb{Z}_p$ and setting

$$\begin{array}{lllll} c'_1 = c_1\, X^{r'} & c'_2 = c_2\, Y^{s'} & c'_3 = c_3\, g^{r'+s'} & c'_4 = c_4\, I(Z)^{r'} & c'_5 = c_5\, J(Z)^{s'} \\ \sigma'_1 = \sigma_1\, \sigma_4^{r'} & \sigma'_2 = \sigma_2\, \sigma_5^{s'} & \sigma'_3 = \sigma_3\, Z^{r'+s'} & \sigma'_4 = \sigma_4 & \sigma'_5 = \sigma_5 \end{array}$$

(It is easily verified that if c was encrypted using randomness (r, s) then c' is an encryption of the same under randomness $(r + r', s + s')$ and σ' is a signature on c'.)

$\mathsf{Publish}(\mathsf{BB})$ removes from every entry in BB the identifier id and the components $c_4, c_5, \sigma_4, \sigma_5$ and publishes the result, called PBB.[4]

$\mathsf{VerifyVote}(\mathsf{PBB}, id, \mathsf{upk}, \mathsf{usk}, b)$ returns $\top$ if there exists a *valid* entry in PBB containing $Z = \mathsf{upk}$, and $\bot$ otherwise. An entry $(Z, c_1, c_2, c_3, \sigma_1, \sigma_2, \sigma_3)$ is valid if we have $e(\sigma_1, g) = e(c_1, Z)$, $e(\sigma_2, g) = e(c_2, Z)$ and $e(\sigma_3, g) = e(c_3, Z)$.

$\mathsf{Tally}(\mathsf{BB}, \mathbf{sk})$ consists of the following steps:

- Parse each ballot $b \in \mathsf{BB}$ as $b = (id_b, Z_b, c_b, \sigma_b)$.

[3]For simplicity, we describe a version of our protocol where revoting is forbidden but it could be easily adapted to allow revoting. This would require to explicitly introduce the revote policy (e.g. only the last vote counts).

[4]Note that by Remark 1, even after removing c_4 and c_5 a user can still be assured that σ, which is valid under his key, certifies that his vote has not been changed. Moreover, σ_4 and σ_5 can also be dropped in the bulletin board, since they are only needed to re-randomize.

– Update BB by removing any ballot b such that $id \notin ID$ or c_b is not valid w.r.t. Z_b or σ_b is not valid. Let BB_c be the result and N the number of remaining ballots.
– Let $\overline{\mathsf{BB}} = \{(c_{b,1}, c_{b,2}, c_{b,3})\}_{b \in \mathsf{BB}_c}$. Shuffle $\overline{\mathsf{BB}}$ by running $(\overline{\mathsf{BB}}_s, \Pi_s) \leftarrow \mathsf{Shuffle}(\overline{\mathsf{BB}})$.
– Decrypt the ciphertexts $c_b = (c_{b,1}, c_{b,2}, c_{b,3}) \in \overline{\mathsf{BB}}_s$ and compute proofs of correct decryption. Let $\Pi_d = \{(c_b, \pi_b, v_b)\}_{b \in \overline{\mathsf{BB}}_s}$, where v_b is the decryption of c_b under decryption key dk (contained in $\mathbf{sk}$) and π_b is the corresponding proof of correct decryption.
– Compute the result r by applying the result function to valid votes.
– Output $(r, \mathsf{PBB}, \Pi_s, \Pi_d)$.

$\mathsf{Verify}(\mathsf{PBB}, r, (\Pi_s, \Pi_d))$ simply verifies Π_s and Π_d w.r.t. PBB and the result r.

Security of BeleniosRF. We now show that BeleniosRF satisfies strong receipt-freeness, as defined in Definition 1.

Theorem 3. $\mathcal{V}^{\mathsf{BeleniosRF}}$ *is strongly receipt-free.*

We first prove the following claim:

Claim 1. *DLIN is random self-reducible.*

Proof. Given a tuple $(X, Y, R = X^r, S = Y^s, T)$ where $T = g^{r+s}$ or T is random, we choose $z, a, b \leftarrow_{\$} \mathbb{Z}_p$ and define the following:

$$R' := R^z g^a = X^{rz+a} \qquad S' := S^z g^b = Y^{sz+b} \qquad T' := T^z g^a g^b = g^{tz+a+b}$$

With $r' := rz + a$ and $s' := sz + b$, which are both uniformly random, we have

$$T' = g^{(t-r-s)z+rz+sz+a+b} = g^{(t-r-s)z+r'+s'} \ .$$

Thus, if the original challenge was a linear tuple, i.e., $t = r + s$ then the new tuple is also linear with randomness r', s'; however, if $t - r - s \neq 0$ then R', S' and T' are uniformly random. □

Proof of Theorem 3. We now prove strong receipt-freeness of BeleniosRF. The challenger in the game sets up all pairs $(\mathsf{upk}_{id}, \mathsf{usk}_{id})$, meaning it knows every signing key. The only difference between Game 0 and Game 1 is that the adversary sees PBB_0 in the former and PBB_1 in the latter. Since PBB only contains encryptions of different votes, it seems that PBB_0 and PBB_1 should be indistinguishable by semantic security of linear encryption. However, at the end of both games the adversary receives the result of the election, which is computed using the secret key—which seems to contradict semantic security. We however show that—similarly to the proof of our RCCA scheme—by using random-oracle programmability, we can simulate decryption without knowing the secret key, which will enable use to embed DLIN instances into PBB.

Entry i in PBB_β is of the form $(Z_i, c'_{i,\beta}, \sigma'_{i,\beta})$, where $(c'_{i,\beta}, \sigma'_{i,\beta})$ is a randomization of $(c_{i,\beta}, \sigma_{i,\beta})$ which was submitted at the i-th query of type $\mathcal{O}\mathsf{voteLR}$, $\mathcal{O}\mathsf{cast}$, or $\mathcal{O}\mathsf{receiptLR}$, and which was stripped off its last two components. Moreover, note that $(c_{i,\beta}, \sigma_{i,\beta})$ must have been valid, otherwise Valid would have returned $\perp$ and the vote would have been discarded. Let (Z_j, z_j) be the key pair corresponding to id_i. Then by Valid, we have that $(c_{i,\beta}, \sigma_{i,\beta})$ is of the form:

$$c_{i,\beta} = (X^r, Y^s, H(m_{i,\beta}) \cdot g^{r+s}, I(Z_j)^r, J(Z_j)^s) \ , \qquad \sigma_{i,\beta} = (c_1^{z_j}, c_2^{z_j}, c_3^{z_j}, X^{z_j}, Y^{z_j}) \ ,$$

for some r, s. On the other hand, PBB_β contains

$$c'_{i,\beta} = (X^{r'}, Y^{s'}, H(m_{i,\beta}) \cdot g^{r'+s'}) \ , \qquad \sigma'_{i,\beta} = ((c'_1)^{z_j}, (c'_2)^{z_j}, (c'_3)^{z_j}) \ ,$$

where r', s' are uniformly random by randomizability of the used signatures-on-randomizable-ciphertext scheme.

We program the random oracles I, J so that we can compute Tally without using the decryption key: When queried for Z_j, we set $I(Z_j) := g^{a_j}$ and $J(Z_j) := g^{b_j}$, for uniform $a_j, b_j \leftarrow\!\!\$\, \mathbb{Z}_p$. For every valid ballot $b_0 = (id, Z, c, \sigma)$ sent by the adversary, and therefore included in BB_0, we have $c = (X^r, Y^s, H(v_0)\, g^{r+s}, I(Z_j)^r, J(Z_j)^s)$ for some r, s, v_0. We have $I(Z_j)^r = g^{a_j r}$ and $J(Z_j)^s = g^{b_j s}$; we can thus retrieve $H(v_0) = c_3\, c_4^{-1/a_j} c_5^{-1/b_j}$ and thereby get v_0 for tallying without knowing the secret key (x, y).

Assume we are given a DLIN challenge (X, Y, X^r, Y^s, T), where either $T = g^{r+s}$ or T is uniformly random in $\mathbb{G}$. By self-reducibility of DLIN (proved in Claim 1 above), from this challenge we can create arbitrarily many tuples (X, Y, R_i, S_i, T_i), where $R_i = X^{r_i}$ and $S_i = Y^{s_i}$ are uniformly random and where (depending on the original challenge) either $T_i = g^{r_i+s_i}$ or it is uniformly random.

We now simulate Game 0, answering queries to I and J as described above and also simulating decryption in order to obtain the votes $v_{i,0}$. Now, instead of rerandomizing the valid ballots received by the adversary, we include $c_i' = (R_i, S_i, H(v_{i,0}) \cdot T_i)$ and $\sigma_i' = ((c'_{i,1})^{z_j}, (c'_{i,2})^{z_j}, (c'_{i,3})^{z_j})$ in PBB_0. If our DLIN challenge was a linear tuple then this perfectly simulates Game 0, whereas if T was random (and therefore all T_i are independently uniformly random) then the adversary's view is independent of β. This means that in the modified game (which is indistinguishable from Game 0 and by an analogous argument from Game 1), the bit β is information-theoretically hidden from the adversary. This concludes the proof. □

Efficiency of BeleniosRF. Although, as we show in the next section, the voting protocol proposed in [BFPV11] is not secure, we compare its efficiency with that of BeleniosRF in terms of ballot size (which also roughly translates to the number of bilinear-group exponentiations a user must perform when casting his vote). The ballot size in BeleniosRF is independent of the bit length k of votes $v \in \mathbb{V}$, as long as k is logarithmic (as $\mathbb{V}$ needs to be polynomial-size to enable efficient exhaustive search). In contrast, the ballots in [BFPV11] grow linearly in the bitlength k of the votes.

Table 1: Comparison of ballot size

[BFPV11], DLIN-based	$9k + 33$ elements from $\mathbb{G}$
[BFPV11], DDH-based	$6k + 15$ elements from $\mathbb{G}_1$ plus $6k + 7$ el. fr. $\mathbb{G}_2$
BeleniosRF (DLIN based)	10 elements from $\mathbb{G}$

5 The Blazy et al. Voting Protocol is not Ballot-Private

Blazy et al. [BFPV11], who introduced the notion of signatures on randomizable ciphertexts, propose to use this primitive for a receipt-free e-voting protocol. Their ballot-creation and -casting protocol workflow is as follows:

- The voter sends ballot
$$b = \big(\mathrm{vk},\ c = \{v\}^r_{pk},\ \sigma_c^{\mathrm{vk},s}, \pi_{pk}^{t, v\in\{0,1\}}\big)\ ,$$
where $r, s, t \in \mathbb{Z}_p$ stand for the local randomness used for encrypting the vote v, signing the resulting ciphertext c and creating the NIZK proof π, respectively.

- The server re-randomizes the ballot b to b' as follows: $\left(vk,\ c = \{v\}_{pk}^{r'},\ \sigma_c^{\mathrm{vk},s'},\ \pi_{pk}^{t',v\in\{0,1\}}\right)$, where $r', s', t' \leftarrow_{\$} \mathbb{Z}_p$.

Thanks to the properties of the involved primitive, the server can only re-randomize legitimate signatures, meaning that any new ballot b' that contains a valid signature w.r.t. vk must originate from a ballot b that has been previously created by the voter, and thus b and b' contain the same vote. On the other hand, the voter has no control over the final randomness r', s', t' that defines b'. This forms the basis of receipt-freeness as claimed in [BFPV11] (the randomness used for encrypting the ballot can be used as a receipt to prove how a voter voted, as it suffices to re-encrypt the vote v using the local randomness).

An attack on ballot privacy. However, the above ballot casting workflow is not receipt-free, as it is not even ballot private. We prove it by showing a ballot replay attack, which is known to break ballot privacy [CS11]. Here follows a concise description of the attack:

- Honest voter sends ballot
$$b = \left(vk,\ c = \{v\}_{pk}^{r},\ \sigma_c^{\mathrm{vk},s}, \pi_{pk}^{t,v\in\{0,1\}}\right)\ .$$
- Server re-randomizes the ballot
$$b' = \left(vk,\ c = \{v\}_{pk}^{r'},\ \sigma_c^{\mathrm{vk},s'},\ \pi_{pk}^{t',v\in\{0,1\}}\right)$$
and displays it on the public bulletin board
- Dishonest voter with credentials $(\bar{vk}, \bar{sk})$ and with knowledge of target ballot b'
  - copies $c = \{v\}_{pk}^{r'},\ \pi_{pk}^{t',v\in\{0,1\}}$ and re-randomizes this to $\{v\}_{pk}^{\bar{r}},\ \pi_{pk}^{\bar{t},v\in\{0,1\}}$;
  - signs $\bar{c}$ with $(\bar{vk}, \bar{sk})$ yielding $\sigma_{\bar{c}}^{\bar{\mathrm{vk}},\bar{s}}$;
  - sends ballot
$$\bar{b} = \left(vk,\ \bar{c} = \{v\}_{pk}^{\bar{r}},\ \sigma_{\bar{c}}^{\bar{\mathrm{vk}},\bar{s}},\ \bar{\pi}_{pk}^{\bar{t},v\in\{0,1\}}\right)\ .$$

These instructions allow any voter with knowledge of a ballot b to produce an independent-looking ballot b', that will be accepted by the voting server and effectively contains a copy of the vote in b. Thus, the voting protocol [BFPV11] is not ballot-private. (Note that due to randomizability, the ballot box cannot discard copied votes, as they look like legitimate ones.)

6 Conclusions

We introduced the notion of *strong receipt-freeness*, whereby a malicious voter cannot feasibly produce a receipt proving how he voted, no matter whether the voter decided to act maliciously before, during or after casting the ballot. We only require that the actions of the voter during ballot casting are not visible to the adversary; however the voter can still present the adversary a plausible record of the actions that he has undertaken when being away from the adversary.

We see our adversarial model close to the spirit of the seminal work in receipt-freeness by Benaloh and Tuinstra [BT94]. Moreover, our definition builds on the recent work [BCG+15], and is in particular simple, concise and game-based. These definitions are well-known for easing the job of conceiving and writing security proofs; a point we confirm by giving a new e-voting protocol that satisfies our definition in bilinear groups under the Decision Linear assumption. The protocol is built using ideas from a previous work [BFPV11] that claimed to have solved this problem. We show however that the previous voting scheme was not even ballot private, which is considered to be a weaker privacy level than receipt-freeness. Still, our scheme saves by a factor $k+3$ in space and computational cost, where k is the bit length of the vote, despite improving the security. We are able to do this by working in the random-oracle model and changing to mixnet-based tallying. Although we have not discussed in depth the verifiability of our protocol, it is not hard to see that this property is inherited from the framework that we built upon, namely, [CGGI14].

As far as we know, this is the first scheme that is both receipt-free (in a strong sense) and has universal verifiability, without requiring the existence of an untappable channel, thus overcoming, with a new paradigm, the impossibility result exposed in [CFP+10]. In the latter, the authors showed that no scheme could be receipt-free and universally verifiable without an untappable channel (this is for instance the case of JCJ/Civitas [JCJ05]). Our solution overcomes this impossibility result by introducing a cryptographically advanced technique, namely, that the ballot box/bulletin board can re-randomize the ballot without changing its contents, in a publicly verifiable way.

References

[AC11] Jordi Puiggalí Allepuz and Sandra Guasch Castelló. Internet voting system with cast as intended verification. In Aggelos Kiayias and Helger Lipmaa, editors, *E-Voting and Identity - Third International Conference, VoteID 2011*, volume 7187 of *LNCS*, pages 36–52. Springer, 2011.

[Adi08] Ben Adida. Helios: Web-based Open-Audit Voting. In *17th USENIX Security Symposium*, pages 335–348, 2008. Helios website: http://heliosvoting.org.

[AdMPQ09] Ben Adida, Olivier de Marneffe, Olivier Pereira, and Jean-Jacques Quisquater. Electing a university president using open-audit voting: Analysis of real-world use of Helios. In *Proceedings of the 2009 conference on Electronic voting technology/workshop on trustworthy elections*, 2009.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 41–55. Springer, August 2004.

[BCG+15] David Bernhard, Véronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. A comprehensive analysis of game-based ballot privacy definitions. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 18-21, 2014*. IEEE Computer Society, 2015.

[Ben07] J. Benaloh. Ballot casting assurance via voter-initiated poll station auditing. In *Proceedings of the Second Usenix/ACCURATE Electronic Voting Technology Workshop*, 2007.

[BFPV11] Olivier Blazy, Georg Fuchsbauer, David Pointcheval, and Damien Vergnaud. Signatures on randomizable ciphertexts. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *PKC 2011*, volume 6571 of *LNCS*, pages 403–422. Springer, March 2011.

[BG12] Stephanie Bayer and Jens Groth. Efficient zero-knowledge argument for correctness of a shuffle. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 263–280. Springer, April 2012.

[BLS04] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. *Journal of Cryptology*, 17(4):297–319, September 2004.

[BT94] Josh Cohen Benaloh and Dwight Tuinstra. Receipt-free secret-ballot elections (extended abstract). In *26th ACM STOC*, pages 544–553. ACM Press, May 1994.

[CCM08] Michael R. Clarkson, Stephen Chong, and Andrew C. Myers. Civitas: Toward a secure voting system. In *2008 IEEE Symposium on Security and Privacy*, pages 354–368. IEEE Computer Society Press, May 2008.

[CEC$^+$08] David Chaum, Aleks Essex, Richard Carback, Jeremy Clark, Stefan Popoveniuc, Alan Sherman, and Poorvi Vora. Scantegrity: End-to-End Voter-Verifiable Optical-Scan Voting. *IEEE Security and Privacy*, 6(3):40–46, 2008.

[CFP$^+$10] Benoît Chevallier-Mames, Pierre-Alain Fouque, David Pointcheval, Julien Stern, and Jacques Traoré. On some incompatible properties of voting schemes. In David Chaum, Markus Jakobsson, Ronald L. Rivest, Peter Y. A. Ryan, Josh Benaloh, Miroslaw Kutylowski, and Ben Adida, editors, *Towards Trustworthy Elections, New Directions in Electronic Voting*, volume 6000 of *LNCS*, pages 191–199. Springer, 2010.

[CGGI14] Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachène. Election verifiability for helios under weaker trust assumptions. In Miroslaw Kutylowski and Jaideep Vaidya, editors, *ESORICS 2014, Part II*, volume 8713 of *LNCS*, pages 327–344. Springer, September 2014.

[CGS97] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 103–118. Springer, May 1997.

[GGR09] Ryan W. Gardner, Sujata Garera, and Aviel D. Rubin. Coercion resistant end-to-end voting. In Roger Dingledine and Philippe Golle, editors, *FC 2009*, volume 5628 of *LNCS*, pages 344–361. Springer, February 2009.

[Gjø13] Kristian Gjøsteen. The norwegian internet voting protocol. *IACR Cryptology ePrint Archive*, 2013:473, 2013.

[IAC] International Association for Cryptologic Research. Elections page at http://www.iacr.org/elections/.

[JCJ05] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-Resistant Electronic Elections. In *4th Workshop on Privacy in the Electronic Society (WPES 2005)*, pages 61–70. ACM, 2005.

[KZZ15] Aggelos Kiayias, Thomas Zacharias, and Bingsheng Zhang. End-to-end verifiable elections in the standard model. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 468–498. Springer, April 2015.

[RBH$^+$09] P.Y.A. Ryan, D. Bismark, J. Heather, S. Schneider, and Z. Xia. The prêt à voter verifiable election system. *IEEE Transactions on Information Forensics and Security*, 4:662–673, 2009.

[SFD$^+$14] 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 Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 14*, pages 703–715. ACM Press, November 2014.