BeleniosRF: A Non-interactive Receipt-Free Electronic Voting Scheme

Pyrros Chaidos
University College London
Gower St, London WC1E 6BT, UK
pyrros.chaidos.10@ucl.ac.uk

Véronique Cortier
LORIA, CNRS & INRIA & Université de Lorraine
54500 Vandœuvre-lès-Nancy, France
veronique.cortier@loria.fr

Georg Fuchsbauer
Inria, ENS, CNRS, PSL Research University
45, rue d'Ulm, 75005 Paris, France
georg.fuchsbauer@ens.fr

David Galindo
University of Birmingham
Edgbaston, Birmingham B15 2TT, UK
D.Galindo@cs.bham.ac.uk

ABSTRACT

We propose a new voting scheme, BeleniosRF, that offers both receipt-freeness and end-to-end verifiability. It is receipt-free in a strong sense, meaning that even dishonest voters cannot prove how they voted. We provide a game-based definition of receipt-freeness for voting protocols with non-interactive ballot casting, which we name *strong receipt-freeness* (sRF). To our knowledge, sRF is the first game-based definition of receipt-freeness in the literature, and it has the merit of being particularly concise and simple. Built upon the Helios protocol, BeleniosRF inherits its simplicity and does not require any anti-coercion strategy from the voters. We implement BeleniosRF and show its feasibility on a number of platforms, including desktop computers and smartphones.

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 she voted. While privacy protects honest voters, receipt-freeness aims at protecting vote privacy even when voters willingly interact with an attacker.
- *Coercion-resistance* should allow an honest voter to cast her vote even if she is, during some time, fully under the control of an attacker. Coercion-resistance typically requires revoting.

CCS'16, October 24 - 28, 2016, Vienna, Austria

ISBN 978-1-4503-4139-4/16/10…$15.00

DOI: http://dx.doi.org/10.1145/2976749.2978337

Conversely, verifiability ensures that voters' ballots are included in the ballot box (*individual verifiability*), that the result corresponds to the content of the ballot box (*universal verifiability*) and that ballots come only from voters entitled to vote (*eligibility verifiability*).

Helios [3, 4] is a scheme that "only" achieves privacy and verifiability and is based on a voting system by Cramer, Gennaro and Schoenmakers [25] with modifications proposed by Benaloh [7]. It has been used in several elections such as that of the president of UC Louvain in Belgium, and of the Board of Directors of the IACR since 2011 [1]. As emphasized by its authors, Helios should only be used in low-coercion environments. Indeed, a voter may reveal the randomness used to compute her ballot; one can then re-encrypt the claimed vote and check if the encryption is contained in the public bulletin board. Helios is thus not receipt-free.

To our knowledge, Civitas [21, 37] 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 [8, 20]). While the scheme is a foundational work, it seems difficult to use it in large-scale elections mainly for two 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 coercion-resistance, a voter should be able to adopt an *anti-coercion strategy* (in Civitas, a voter has to lie about her true credential) and then later *revote* for her true choice once she is freed from the attacker. We believe that this scenario is unrealistic in many cases, as it requires cryptographic skills and a heavy infrastructure to realize an untappable channel (e.g. in-person registration).

It is also worth noticing that in most countries revoting is not allowed, as for example in Australia, France, Spain, Switzerland and the United Kingdom. The only exceptions we are aware of are Estonia and the Internet voting pilots for the parliamentary elections in 2011 and 2013 in Norway. While this way of thinking might be a cultural aspect inherited from traditional paper ballot systems, it is foreseeable that it will take time before countries change their electoral rules in order to adopt a revote policy.

1.1 Our Contributions

Building upon a recent variant of Helios, called Belenios [22, 30], and a cryptographic primitive called *signatures on* *randomizable ciphertexts* [12], we propose a receipt-free version of Helios, which we call BeleniosRF. In our scheme a voter cannot prove how she voted, even if she is provided with all the ballot material by the coercer. Interestingly, our scheme does not demand any strategy of the voter; in particular, it does not require the active participation of a voter to deceive a coercer that is asking for a receipt. For example, a voter does not need to lie or produce fake credentials as in Civitas, she simply *has no way* to prove how she voted. This represents a huge improvement in usability from the voter's point of view: all that is required of the voter is to vote.

We show that our scheme BeleniosRF is receipt-free in a strong sense, meaning that even a dishonest voter using a voting client that has been tampered with cannot prove how she voted. We formalize this property, called *strong receipt-freeness* (sRF), via a game-based definition building on the privacy definition recently proposed by Bernhard *et al.* [9]. We view this formal definition of receipt-freeness, which applies to non-interactive ballot casting protocols, as the first contribution of this work. We call it *strong* receipt-freeness to emphasize that in non-interactive protocols an attacker has less room to build a receipt. Indeed, in the absence of interaction the adversary does not obtain information from the voting server apart from what is displayed on the bulletin board; hence any receipt must be built by the adversary locally and before submitting the ballot.

We claim sRF is the first game-based receipt-freeness definition in the literature accounting for a voter that is corrupted during the voting phase. Additionally, sRF has the merit of being simple and concise, potentially allowing for simpler proofs. In doing so we give a new formulation for the receipt-freeness definition by Benaloh and Tuinstra [8] and highlight that receipt-freeness can be achieved without asking the voters to vote several times and cancel previously submitted ballots, and without requiring an untappable channel. All we need to assume is that the attacker is not permanently eavesdropping the communication between the voting server and the voter, an assumption made by all previous constructions of receipt-free or coercion-resistant voting schemes.

A key ingredient of BeleniosRF is a *randomization service*, a role that we assume is played by the voting server, but which could be played by a different server. The randomization service is in charge of *re-randomizing* the ballot cast by a voter. BeleniosRF's receipt-freeness then relies on the fact that the randomness contained in the ballot displayed in the bulletin board *is not* under the control of the voter. Both the voter and the randomization service contribute to the randomness of the voter's ballot as displayed on the bulletin board. In fact, in light of the impossibility result of [19], the existence of a randomization agent is assumed in most constructions that claim to be receipt-free. Here however, we do not rely on letting voters vote multiple times or on the existence of a trusted token for each voter (such as e.g. [20, 34, 35, 43]).

The foremost challenge in achieving receipt-freeness non-interactively and via a randomization service is to prevent the latter from changing the voter's intent. The only existing non-interactive proposal [12] claiming receipt-freeness uses a powerful cryptographic 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 knowing neither the decryption key nor the signing key nor the plaintext. On the other hand, unforgeability guarantees that it is infeasible to compute a signature on a ciphertext that encrypts a message of which no encryption has been signed.

Alas, Blazy *et al.* [12] did not provide a receipt-freeness definition nor a proof. By exhibiting a ballot-copying attack adapted from [24], we demonstrate that their scheme is not receipt-free, worse, it is not even ballot-private. Our scheme fixes the Blazy *et al.* construction by binding the ciphertexts to voters, while still inheriting the randomizability from Groth-Sahai non-interactive proofs [32].

We start with giving a new instantiation of signatures on randomizable ciphertexts, which we show yields an RCCA-secure public-key encryption scheme [16], from which we build a non-interactive[1] receipt-free e-voting scheme 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 their ballot and includes a proof of knowledge to prevent ballot malleability.
- Upon receiving a ballot, the server re-randomizes the ballot and adapts the corresponding signature and proof before publishing it.

Receipt-freeness comes from the fact that a voter no longer has control over, nor knowledge of, the randomness used to form the final ballot stored in the ballot box. On the other hand, even after the voting server re-randomizes the ballot cast by the voter's voting device, the voter can still verify that her legitimate ballot is present, as the re-randomized ciphertext comes with a signature that is valid under the voter's verification key. By unforgeability of the signature primitive, the vote cannot have been altered by the ballot box, which we show implies verifiability.

Our final contribution consists of assessing the feasibility of BeleniosRF; for this purpose we implemented and measured the efficiency of a Javascript voting client (see Section 5).

1.2 Related Work

Our definition requires that an adversary cannot distinguish whether a voter votes for either a or b, even if the attacker provides the voter in advance with all the cryptographic material (such as randomness to be used to cast the ballot). Interestingly, this definition does not require the voter to follow a "strategy" to fool the coercer.

The early definitions of receipt-freeness [8, 43] introduced the idea that an attacker should not be able to distinguish between a real transcript and a fake one, but their descriptions are rather informal. A weaker definition of receipt-freeness, proposed in [15, 38], lets the attacker only interact with the voter *after* the election (in particular, the attacker cannot control the randomness of the voter's device). A simulation-based definition for receipt-freeness (UC-RF) was given in [42]. It assumes however that the voters adopt an "anti-coercion strategy" and is therefore closer to coercion-resistance as defined in [37], even if it does not cover, for instance, abstention

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

attacks. Since BeleniosRF does not require any anti-coercion strategy from the voters, and game-based security definitions are known to be easier to work with, we opted not to use UC-RF to analyze the receipt-freeness of our new protocol. The coercion-resistance and receipt-freeness definitions in [40] also assume a strategy from the voter. Our definition can be seen as a formalization of one of the other possible strategies sketched in the paper.

Similarly, the symbolic definition of receipt-freeness in [26] also requires the voter to adopt a strategy to fool the adversary. Other definitions in symbolic models aim at characterizing the notion of a receipt [14, 33, 36] but (as usual in symbolic models) they are much more abstract than standard computational models.

Previous receipt-free schemes. The scheme by Kiayias *et al.* [38] only achieves receipt-freeness for honest voters, as discussed above. Other well-known and deployed schemes include Prêt-à-voter [44] and Scantegrity [18]. These systems however are designed for elections with physical voting booths. The system used in Estonia [46] and the one deployed in Norway [5, 29] 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). Kulyk *et al.* [39] propose an extension of Helios where voters may later cancel their vote, still being indistinguishable from null votes submitted by the crowd. In addition to being difficult to deploy in practice, this scheme strongly relies on revoting. Hirt's scheme [34] heavily depends on the existence of untappable channels. Selene [45] proposes an enhancement for receipt-free schemes, applicable to BeleniosRF, to ease the verification step made by voters through tracking numbers.

BeleniosRF can be seen as realizing receipt-freeness under the assumption that the voting server, which is in charge of running the ballot box, can be trusted to re-randomize ballots and not reveal the randomness used for that procedure. In contrast, Helios [41] and [38] achieve receipt-freeness under the assumption that the voting client is not going to reveal the randomness it used for sealing the vote. The latter seems difficult to ensure in practice, unless voters are provided with secure hardware tokens. In contrast, BeleniosRF only needs the voting server to be protected against randomness leakage.

BeleniosRF has one disadvantage compared to Helios and Belenios: if the registrar and the voting server collude, they can undetectably change a voter's choice. This is due to the features that guarantee receipt-freeness, namely that signatures on different ciphertexts encrypting the same message cannot be linked, and that the registrar generates the voters' signing keys (and thus can vote on their behalf). This can be prevented by defining a less powerful registrar that simply grants voting rights to signing keys that are generated by the voters themselves (cf. Section 4.3). This solution differs from the Belenios approach, where voters receive their signing keys from the registrar for the sake of usability. This is just another manifestation of the usual tension between usability, privacy and verifiability in e-voting systems (and computer security systems in general), in the sense that increasing one of them entails a decrease of at least one of the others.

2. RECEIPT-FREENESS

We now formally define receipt-freeness and start by providing the syntax of a voting system, inspired by [9, 22].

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 received by every candidate).
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 are denoted by $id_1, \dots, id_\tau$.
5. *Ballot-box (voting server) manager*: denoted by $\mathcal{B}$, is responsible for processing and storing valid ballots in the ballot box BB, and for publishing PBB, the public view of BB, also called (public) bulletin board.

The following syntax considers *single-pass* schemes, that is, systems where voters only have to post a single message to the board, i.e. ballot casting is non-interactive. A voting protocol $\mathcal{V} = (\mathsf{Setup}, \mathsf{Register}, \mathsf{Vote}, \mathsf{Valid}, \mathsf{Append}, \mathsf{Publish}, \mathsf{VerifyVote}, \mathsf{Tally}, \mathsf{Verify})$ is 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.

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

$\mathsf{Register}(id)$, on input an identifier id, outputs the secret part of the credential usk_{id} and its public credential upk_{id}, which is added to the list $L = \{upk_{id}\}$.

$\mathsf{Vote}(id, upk, usk, v)$ is run by voter id with credentials upk, usk to cast her vote $v \in \mathbb{V}$. It outputs a ballot b, which is sent to the voting server (possibly through an authenticated channel).

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

$\mathsf{Append}(\mathsf{BB}, b)$ updates BB 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 our scheme).

$\mathsf{Publish}(\mathsf{BB})$ outputs the public view PBB of BB. Often one simply has $\mathsf{Publish}(\mathsf{BB}) = \mathsf{BB}$.

$\mathsf{VerifyVote}(\mathsf{PBB}, id, upk, usk, b)$ is 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, usk, upk, it returns $\top$ or $\bot$.

$\mathsf{Tally}(\mathsf{BB}, \mathbf{sk})$ on inputs the ballot box BB and the secret key $\mathbf{sk}$, outputs the tally r and a proof of correct tabulation Π. If the election is declared invalid then $r := \bot$.

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

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

2.2 Strong Receipt-Freeness

Intuitively, privacy ensures that an adversary cannot learn the vote of an honest voter. Receipt-freeness furthermore guarantees that a voter cannot prove how she voted, even if she willingly provides information to, or follows instructions by, the adversary. This captures the seminal intuition from Benaloh and Tuinstra [8]. The latter insisted that a reasonably private electronic voting protocol should emulate traditional voting in a voting booth: it should *allow* voters to conceal their individual votes and, at the same time, *prevent* them from revealing their vote. Voters should not be able to give away the privacy of their vote granted by the voting protocol, even if they are willing to.

Building upon a definition of privacy recently introduced [9], we argue that this requirement can be formalized for single-pass schemes by simply providing the adversary with an additional oracle $\mathcal{O}\mathsf{receiptLR}$, which allows him to submit his own ballots on behalf of a dishonest voter. Apart from immediately implying ballot privacy, this simple formalization captures several important scenarios:

- A voter who wants to convince a vote buyer of how she voted may prepare her ballot in an arbitrary way that allows him to construct a convincing receipt (e.g., consider a voter that uses biased random coins to build her ballot and to prove how she voted [28]).
- A voter that might have been corrupted before the ballot casting phase may just follow the instructions given to her by the adversary (as in [37]).
- A voter can record, but also forge, its interaction with the ballot box (as in [8]).

As in previous formal or intuitive definitions of receipt-freeness, we assume the adversary is not monitoring the interaction between the voter and the voting server. However, the voter can record this interaction, and later on present this information (or any transformation thereof) to the adversary.

Formally, we consider two games, Game 0 and Game 1, defined by the oracles in Figure 1. In both games BB_0 and 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 a fake ballot box which the adversary's task is to distinguish from BB_0. In Game β the adversary has indirect access to BB_β, that is, she can see the public part of that box at any time. The game $\mathsf{Exp}^{\mathsf{srf},\beta}_{\mathcal{A},\mathcal{V}}$ provides an adversary $\mathcal{A}$ access to the oracles defined in Figure 1, which intuitively proceed as follows:

$\mathcal{O}\mathsf{init}$ generates secret and public keys for the election; the public key is returned to the adversary. If $\beta = 1$, it also returns auxiliary information aux to be used by a simulator $\mathsf{SimProof}$ introduced below.

$\mathcal{O}\mathsf{reg}$, on input an identifier id, initializes id's credentials (upk, usk) by running $\mathsf{Register}(id)$. It gives upk to the adversary.

$\mathcal{O}\mathsf{corrupt}$ is used by the attacker to obtain the credentials (upk, usk) of a registered voter.

$\mathcal{O}\mathsf{voteLR}$, a left-or-right oracle, 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 boxes (one in BB_0 and one in BB_1), provided that $v_0, v_1 \in \mathbb{V}$.

$\mathcal{O}\mathsf{cast}$ 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.

$\mathcal{O}\mathsf{receiptLR}$ 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 its respective ballot box, then the ballots are appended 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 (e.g. as in [28]).

$\mathcal{O}\mathsf{board}$ models the adversary's ability to see the publishable part of the board. It returns $\mathsf{Publish}(\mathsf{BB}_\beta)$.

$\mathcal{O}\mathsf{tally}$ allows the adversary to see the result of the election. In both games the result is obtained by tallying a valid BB_0; the proof of correct tabulation is however simulated in the second world, i.e., for $\beta = 1$.

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

Inherited from ballot privacy [9], Definition 1 uses simulators $\mathsf{SimSetup}$ and $\mathsf{SimProof}$ to model the fact that the proof should not reveal anything, as it is "zero-knowledge".

DEFINITION 1 (sRF). *Let* $\mathcal{V} = (\mathsf{Setup}, \mathsf{Register}, \mathsf{Vote}, \mathsf{Valid}, \mathsf{Append}, \mathsf{VerifyVote}, \mathsf{Publish}, \mathsf{Tally}, \mathsf{Verify})$ *be a voting protocol for a set ID of voter identities and a result function* ρ. *We say that* $\mathcal{V}$ *has* strong receipt-freeness *if there exist algorithms* $\mathsf{SimSetup}$ *and* $\mathsf{SimProof}$ *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* $\mathcal{A}$ *the following is negligible in* λ*:*

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

In protocols with non-interactive ballot casting an adversary does not receive any output from its interaction with the ballot box (apart from the public view of the protocol run), the sRF adversary must therefore build a receipt using local data only, and before casting the ballot. An adversary might encode arbitrary instructions in b_β, for instance making those instructions dependent on the vote v_β; e.g. he could set the least significant bit of b_β equal to $v_\beta \in \{0, 1\}$. Intuitively, *strong receipt-freeness* implies that a ballot b_0 could be replaced by a ballot b_1, both submitted via the oracle $\mathcal{O}\mathsf{receiptLR}$, without the adversary noticing. Thus a receipt, i.e. a proof for a certain vote having been cast, cannot exist as $\mathcal{O}\mathsf{receiptLR}$ captures all what a RF adversary can do.

This definition does not assume that the voter is capable of successfully applying some anti-coercion strategy (in contrast to [42]). We believe this to be important in practice for two reasons. First, this is of course much easier to use: with our definition, the system *is* receipt-free by construction and there is no need to instruct voters how they should proceed to lie about their vote. Second, we need not assume that revoting is allowed (our definition accommodates any revoting policy though, including no revote). This is important since most countries forbid revoting.

$\mathcal{O}\mathsf{init}$ **for** $\beta = 0$

$(\mathbf{pk}, \mathbf{sk}) \leftarrow \mathsf{Setup}(1^k)$
return $\mathbf{pk}$

$\mathcal{O}\mathsf{init}$ **for** $\beta = 1$

$(\mathbf{pk}, \mathbf{sk}, aux) \leftarrow \mathsf{SimSetup}(1^k)$
return $\mathbf{pk}$

$\mathcal{O}\mathsf{reg}(id)$

If id was not previously queried,
then run $\mathsf{Register}(id)$ and set
$\mathcal{U} = \mathcal{U} \cup \{(id, upk_{id}, usk_{id})\}$;
return upk_{id}.

$\mathcal{O}\mathsf{corruptU}(id)$

On a registered voter id, output (upk_{id}, usk_{id})
and set $\mathcal{CU} = \mathcal{CU} \cup \{(id, upk_{id})\}$.

$\mathcal{O}\mathsf{cast}(id, b)$

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

$\mathcal{O}\mathsf{voteLR}(id, v_0, v_1)$

If $v_0 \notin \mathbb{V}$ or $v_1 \notin \mathbb{V}$ then return $\perp$.
$b_0 = \mathsf{Vote}(id, upk_{id}, usk_{id}, v_0)$
$b_1 = \mathsf{Vote}(id, upk_{id}, usk_{id}, v_1)$.
$\mathsf{Append}(\mathsf{BB}_0, b_0)$; $\mathsf{Append}(\mathsf{BB}_1, b_1)$

$\mathcal{O}\mathsf{receiptLR}(id, b_0, b_1)$

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, Π)

$\mathcal{O}\mathsf{tally}()$ **for** $\beta = 1$

$(r, \Pi) \leftarrow \mathsf{Tally}(\mathsf{BB}_0, \mathbf{sk})$
$\Pi' \leftarrow \mathsf{SimProof}_{aux}(\mathsf{BB}_1, r)$
return (r, Π')

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 the board displayed in response to $\mathcal{O}\mathsf{board}$ queries, and the board against which ballots are validated.

As expected, strong receipt-freeness trivially implies BPRIV privacy [9], since BPRIV equals sRF except that there is no oracle $\mathcal{O}\mathsf{receiptLR}$.

Helios. Under the RF definition provided in [38] the Helios protocol would be 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, so it suffices for an adversary to produce different encryptions c, d and check which one is showing up when calling oracle $\mathcal{O}\mathsf{board}$.

Interestingly, we believe that a Helios instantiation in which voting devices are built upon trusted hardware tokens that conceal the randomness used for encryption (as proposed by Magkos *et al.* [41]) satisfies sRF, when interpreting trusted tokens as preventing $\mathcal{A}$ from accessing the $\mathcal{O}\mathsf{receiptLR}$ oracle—in which case sRF collapses to ballot privacy. This shows the flexibility of our definition. Moreover, we are confident that the same result applies to [38].

3. BUILDING BLOCKS

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

3.1 Assumptions and Primitives

We will work in asymmetric bilinear groups and assume the existence of a *bilinear-group generator* GrpGen, which on input 1^λ outputs $(p, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, \mathbb{G}_T, e)$, where p is a prime of length λ, $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ are cyclic groups of order p, g_1 is a generator of $\mathbb{G}_1$, g_2 is a generator of $\mathbb{G}_2$, and e is a bilinear map $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ such that $e(g_1, g_2)$ generates $\mathbb{G}_T$. The following was discussed in [13, p. 304] and defined in [12].

DEFINITION 2 (CDH$^+$). *The CDH$^+$ assumption for* GrpGen *holds if for* $\mathcal{G} = (p, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, \mathbb{G}_T, e) \xleftarrow{\$} \mathsf{GrpGen}(1^\lambda)$, *and for* $a, b \xleftarrow{\$} \mathbb{Z}_p$, *for every p.p.t. adversary given* $(\mathcal{G}, g_1^a, g_2^a, g_1^b)$, *the probability that it outputs* g_1^{ab} *is negligible in* λ.

The next assumption implies the security of ElGamal encryption in both groups $\mathbb{G}_1$ and $\mathbb{G}_2$:

DEFINITION 3 (SXDH). *The Symmetric external Diffie-Hellman assumption (SXDH) holds for* GrpGen *if for* $\mathcal{G} = (p, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, \mathbb{G}_T, e) \xleftarrow{\$} \mathsf{GrpGen}(1^\lambda)$, $a, b, c \xleftarrow{\$} \mathbb{Z}_p$ *and for both* $i \in \{1, 2\}$, *p.p.t. adversaries only distinguish* $(\mathcal{G}, g_i^a, g_i^b, g_i^{ab})$ *from* $(\mathcal{G}, g_i^a, g_i^b, g_i^c)$ *with advantage negligible in* λ.

ElGamal Encryption. We define encryption for messages in $\mathbb{G}_1$ from an asymmetric bilinear group $\mathcal{G} = (p, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, \mathbb{G}_T, e)$ and show that it is randomizable.

$\mathsf{KeyGen}(\mathcal{G}, i)$: Choose $d \xleftarrow{\$} \mathbb{Z}_p$ and define $P := g_1^d$. Return $(pk = P, dk = d)$.

$\mathsf{Encrypt}(P, M; r)$: Using randomness $r \in \mathbb{Z}_p$, output $c = (c_1 = g_1^r, c_2 = M \cdot P^r)$.

$\mathsf{Decrypt}(d, c = (c_1, c_2))$: Output $M = c_2 \cdot c_1^{-d}$.

$\mathsf{Random}(P, c = (c_1, c_2); r')$: Using randomness $r' \in \mathbb{Z}_p$, output $c' = (c_1 \cdot g_1^{r'}, c_2 \cdot P^{r'})$.

This scheme is IND-CPA secure assuming hardness of DDH in $\mathbb{G}_1$, which follows from SXDH. It is perfectly randomizable as $\mathsf{Random}(pk, \mathsf{Encrypt}(pk, M; r); r') = \mathsf{Encrypt}(pk, M; r + r')$.

Groth-Sahai Proofs. Groth-Sahai (GS) proofs [32] allow us to prove satisfiability of equations involving group elements from $\mathbb{G}_1$ or $\mathbb{G}_2$ and scalars. We will use them to prove consistency and knowledge of encryptions. On input a bilinear group $\mathcal{G}$, $\mathsf{Setup}_{\mathrm{GS}}$ outputs a common reference string (CRS) $crs \in \mathbb{G}_1^4 \times \mathbb{G}_2^4$. The CRS is used to commit to group elements $X \in \mathbb{G}_1$, which we denote by $\mathcal{C}_1(X)$, and elements $Y \in \mathbb{G}_2$, denoted by $\mathcal{C}_2(Y)$. Moreover, $\mathcal{C}'_i(x)$ denotes a commitment to a scalar, which can be made in $\mathbb{G}_1$ $(i = 1)$ and $\mathbb{G}_2$ $(i = 2)$. The GS system lets us prove that committed values satisfy certain equations.

Under a CRS computed via $\mathsf{Setup}_{\mathrm{GS}}$, commitments are perfectly binding and the proofs are perfectly sound. That is, the values uniquely determined by the commitments satisfy the proved equation. Moreover, the committed values can be extracted using an extraction trapdoor ξ that can be computed together with the CRS. We denote this by $(crs, \xi) \xleftarrow{\$} \mathsf{Setup}^{(x)}_{\mathrm{GS}}(\mathcal{G})$.

There is an alternative CRS-generation algorithm $\mathsf{Setup}^{(h)}_{\mathrm{GS}}$, which outputs (crs', td). Commitments made under crs' contain no information about the committed value and the trapdoor td allows simulation of proofs. As CRSs output by $\mathsf{Setup}_{\mathrm{GS}}$ and $\mathsf{Setup}^{(h)}_{\mathrm{GS}}$ are indistinguishable under SXDH, GS proofs are computationally zero-knowledge. Moreover, GS proofs are randomizable [27], that is, given commitments and proofs, one can (without knowing the witness) create a fresh set of commitments and proofs.

3.2 Signatures on Randomizable Ciphertexts

The primitive introduced by Blazy *et al.* [12] consists of the following algorithms: Setup, on input the security parameter 1^λ, outputs the parameters (such as the bilinear group); SKeyGen outputs a pair of signing key and verification key (sk, vk), EKeyGen outputs a pair of encryption and decryption key (pk, dk). SKeyGen together with Sign and Verify constitutes a signature scheme and EKeyGen with Encrypt and 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'.

An output of Random^+ is distributed like a fresh encryption of the plaintext of c and a fresh signature on it; formally, for all messages m, $(pk, dk) \in [\mathsf{EKeyGen}(\mathcal{G})]$, $(vk, sk) \in [\mathsf{SKeyGen}(\mathcal{G})]$, $c \in [\mathsf{Encrypt}^+(pk, vk, m)]$, $\sigma \in [\mathsf{Sign}^+(sk, pk, c)]$, the following two random variables are equally distributed:

$$\Big[(c', \sigma') \xleftarrow{\$} \mathsf{Random}^+(pk, vk, c, \sigma) : (c', \sigma')\Big] \approx \left[\begin{array}{l} c' \xleftarrow{\$} \mathsf{Encrypt}^+(pk, vk, m); \\ \sigma' \xleftarrow{\$} \mathsf{Sign}^+(sk, pk, c') \end{array} : (c', \sigma')\right] .$$

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, dk). It is also given access to an oracle $\mathsf{Sign}^+(sk, pk, \cdot)$, which it can query adaptively on ciphertexts $c_1, \dots, c_q$ of its choice. Finally, the adversary 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 $m_i := \mathsf{Decrypt}^+(dk, vk, c_i)$.

3.3 Our SRC Construction

At a high level, we need a construction that enforces our restricted message space and is malleable enough to be re-randomized but no more. The first requirement ensures that voters can only submit valid ballots, while the second gives us privacy via randomization while preventing copying or tampering attacks. Specifically, we use GS proofs to ensure validity and prevent copying or producing ballots related to those of another user. We use signatures to ensure integrity, meaning a randomizer cannot change the ballot contents.

Asymmetric Waters signature scheme. Blazy *et al.* [12] define a variant of Waters' signature scheme [47] for asymmetric groups that is perfectly randomizable and which they prove secure under the CDH$^+$ assumption.

$\mathsf{Setup}(1^\lambda, 1^k)$: To sign messages $m = (m_1, \dots, m_k) \in \{0,1\}^k$, generate $(p, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, \mathbb{G}_T, e) \xleftarrow{\$} \mathsf{GrpGen}(1^\lambda)$, choose $z \xleftarrow{\$} \mathbb{G}_1$, $\boldsymbol{u} = (u_0, \dots, u_k) \xleftarrow{\$} \mathbb{G}_1^{k+1}$, define $\mathcal{F}(m) := u_0 \prod_{i=1}^k u_i^{m_i}$. Output $pp := (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, z, \boldsymbol{u})$.

$\mathsf{SKeyGen}(pp)$: Choose $x \xleftarrow{\$} \mathbb{Z}_p$, define $X_1 = g_1^x$, $X_2 = g_2^x$, $Y = z^x$; output the public key $vk = (pp, X_1, X_2)$ and the secret key $sk = (pp, Y)$.

$\mathsf{Sign}(sk = (pp, Y), m; s)$: For randomness $s \in \mathbb{Z}_p$, return the signature σ defined as

$$\big(\sigma_1 = Y \cdot \mathcal{F}(m)^s,\ \sigma_2 = g_1^s,\ \sigma_3 = g_2^s\big) \ .$$

$\mathsf{Verif}(vk = (pp, X_1, X_2), m, \sigma)$: Output 1 if both of the following hold and 0 otherwise:

$$e(\sigma_1, g_2) = e(z, X_2) \cdot e(\mathcal{F}(m), \sigma_3) \quad e(\sigma_2, g_2) = e(g_1, \sigma_3)$$

$\mathsf{Random}((pp, X_1, X_2), F, \sigma; s')$: For randomness $s' \in \mathbb{Z}_p$, output $\sigma' = (\sigma_1 \cdot F^{s'},\ \sigma_2 \cdot g_1^{s'},\ \sigma_3 \cdot g_2^{s'})$.

Note that for Random it suffices to know the hash $F = \mathcal{F}(m)$ of the signed message. The scheme is perfectly randomizable, as for any $((pp, X_1, X_2), (pp, Y)) \in [\mathsf{SKeyGen}(pp)]$ and m, s, s' we have $\mathsf{Random}((X_1, X_2), \mathcal{F}(m), \mathsf{Sign}(Y, m; s), s') = \mathsf{Sign}(Y, m; s + s')$.

REMARK 1. *Blazy et al. [12] show that their signature scheme also satisfies a (stronger) EUF-CMA notion, where the adversary's signing queries are of the form (m, R, T) and if $e(T, g_2) = e(R, X_2)$ then the oracle returns an additional signature element $\sigma_4 = R^s$.*

We will combine ElGamal encryption, Groth-Sahai proofs and Waters signatures to create an SRC scheme. Our construction extends that of [12], so that it immediately yields an RCCA-secure encryption scheme (defined below) and ultimately a strongly receipt-free e-voting scheme.

Our SRC scheme. Our scheme is defined for a polynomial-size message space $\mathcal{M} = \{0,1\}^k$, that is, we assume k to be logarithmic in the security parameter. Messages m are encrypted as ElGamal ciphertexts of $\mathcal{F}(m)$. Decryption works by decrypting a ciphertext to F and then looking for m with $F = \mathcal{F}(m)$. We define a function H and add a third ciphertext element $c_3 = H(vk)^r$, which will tie the ciphertext to the verification key for which it is produced.

We moreover add C_m, Groth-Sahai commitments to the message bits, and a commitment C_T to X_1^r, which is needed for the security reduction (it corresponds to T from Remark 1). Finally, we add GS proofs which show consistency of these commitments, and consistency of the additional ciphertext element c_3. In more detail, in order to show a component C_{m_i} of C_m contains a bit, we require commitments in both groups $(C_{1,m,i}, C_{2,m,i})$ and proofs π'_i. A commitment C_r to the randomness r is used to prove consistency of the values c_1 with C_r, c_2 with C_r and $\{C_{m,i}\}_{i=1}^k$, c_3 with C_r, as well as C_T with X_1 and C_r.

Now, given ciphertext elements $c_1 = g_1^r$ and $c_2 = \mathcal{F}(m)\cdot P^r$, the crucial observation is that, due to the interoperability of ElGamal encryption and Waters signatures, a signer can produce an encryption of a signature on the *plaintext*, without knowing the latter: setting $\sigma_1 = c_1^s = g_1^{rs}$ and $\sigma_2 = Y \cdot c_2^s = Y \cdot \mathcal{F}(m)^s \cdot P^{rs}$ yields an encryption under P of the first Waters signature element $Y \cdot \mathcal{F}(m)^s$. This is completed to a full signature by (g_1^s, g_2^s). Finally, in order to enable full randomization of ciphertext/signature pairs, we also include P^s in the signature.

Let Setup (for Waters signatures), $\mathsf{Setup}_{\mathrm{GS}}$ (for Groth-Sahai proofs) and KeyGen (for ElGamal encryption) be defined as above, and $H\colon \{0,1\}^\star \to \mathbb{G}_1$ be defined as

$$H(x) := h_1 \cdot h_2^{H'(vk)} \tag{1}$$

for $h = (h_1, h_2) \in \mathbb{G}_1^2$ and a collision-resistant hash function $H'\colon \{0,1\}^\star \to \mathbb{Z}_p$. Our scheme is given in Figure 2. It is based on the scheme from [12], to which we add the crucial ciphertext elements c_3 and π_V.

Correctness follows, as Verify^+ checks, via the pairings, that σ is of the form in (2) for some s. From a ciphertext/signature pair (c, σ) with randomness (r, s), Random^+ creates a fresh pair (c', σ') with randomness $r + r', s + s'$ (and with randomized proofs). We omit the specific structure of the proofs in $\boldsymbol{\pi}$ as they will not be relevant to the rest of this work.

Theorem 1. *The SRC scheme* ($\mathsf{Setup}, \mathsf{EKeyGen}, \mathsf{SKeyGen}, \mathsf{Encrypt}^+, \mathsf{Decrypt}^+, \mathsf{Sign}^+, \mathsf{Verify}^+, \mathsf{Random}^+$) *defined in Figure 2 is unforgeable under the CDH$^+$ assumption.*

Remark 2. *We will prove a stronger statement, namely that our SRC scheme is unforgeable even when the adversary only needs to output a "partial" forgery* $(c_1, c_2, \{C_{1,m,i}, C_{2,m,i}\}, C_r, \pi_r, \pi_m), (\sigma_1, \sigma_2, \sigma_3, \sigma_4)$*, i.e., it need not contain* c_3, C_T, π_T, π_V *and* σ_5*.*

Moreover, note that one can also decrypt ciphertexts using the extraction trapdoor ξ *for GS proofs to recover* m *from* C_m*, sidestepping the inefficient hash inversion. We let* $\mathsf{EKeyGen}^{(x)}$ *denote key generation that returns* ξ *instead of* dk*.*

Proof. The proof is by reduction from unforgeability of Waters signatures. The reduction obtains a verification key vk including parameters pp. It simulates $\mathsf{EKeyGen}$ by running $(crs, \xi) \xleftarrow{\$} \mathsf{Setup}_{\mathrm{GS}}^{(x)}(\mathcal{G})$, $(P, d) \xleftarrow{\$} \mathsf{KeyGen}(\mathcal{G}, 1)$ and choosing $h \xleftarrow{\$} \mathbb{G}_1^2$, and runs the adversary on $vk, pk := (pp, crs, h, P)$ and $dk := d$. If the adversary queries a signature on a valid tuple $c = (c_1, c_2, c_3, \boldsymbol{C}, \boldsymbol{\pi})$, the reduction uses ξ to extract m and $T = X_1^r$ from $\boldsymbol{C}$ (note that by soundness of $\boldsymbol{\pi}$, we have $m = \mathsf{Decrypt}^+(dk, vk, c)$). The reduction makes a special query, as defined in Remark 1, (m, c_1, T) to its signing oracle (note that c_1, T satisfy $e(T, g_2) = e(c_1, X_2)$); it obtains a signature $(\tau_1 = Y\mathcal{F}(m)^s, \tau_2 = g_1^s, \tau_3 = g_2^s, \tau_4 = c_1^s)$; it defines (letting r be the unknown randomness in (c_1, c_2, c_3)) $\sigma_1 := \tau_4 = g_1^{rs}$, $\sigma_2 := \tau_1 \cdot \tau_4^d = Y\mathcal{F}(m)^s P^{rs}$, $\sigma_3 := \tau_2 = g_1^s$, $\sigma_4 := \tau_3 = g_2^s$, $\sigma_5 := \tau_2^d = P^s$, which is distributed as an SCR signature on c.

Let $\{m_1, \ldots, m_q\}$ be the extracted (equivalently: decrypted) messages of the signing queries. Assume the adversary outputs a (partial) valid forgery, namely one which only contains $(c_1, c_2, \sigma_1, \sigma_2, \sigma_3, \sigma_4)$, commitments C_m, C_r and proofs π_r, π_m. The reduction extracts m from C_m. Then soundness of π_r and π_m ensures that for some r we have $c_1 = g_1^r$ and $c_2 = \mathcal{F}(m)P^r$ (and thus $m = \mathsf{Decrypt}^+(dk, vk, c)$).

Moreover, let s be such that $\sigma_4 = g_2^s$. Since the forgery is valid, from Verify^+ we have: $\sigma_1 = g_1^{rs}$ (from (3a)), $\sigma_2 = Y\mathcal{F}(m)^s P^{rs}$ (from (3b)) and $\sigma_3 = g_1^s$ (from (3c)). The reduction sets $\sigma_1^* := \sigma_2 \cdot \sigma_1^{-d} = Y\mathcal{F}(m)^s$, $\sigma_2^* := \sigma_3 = g_1^s$ and $\sigma_3^* := \sigma_4 = g_2^s$ and returns (m, σ^*). This is a valid Waters forgery, as σ^* is valid for m and $m \notin \{m_1, \ldots, m_q\}$ (otherwise the adversary would not have won the SRC unforgeability game). $\square$

3.4 RCCA-Secure Encryption from SRC

As a next step towards our voting protocol, we show that our SRC scheme, contrary to the one from [12], yields an RCCA-secure [16] encryption scheme, as defined next.

CCA-security is the standard notion for public-key encryption and implies that ciphertexts are non-malleable. It states that for an efficient adversary which 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 submit a randomization of the challenge ciphertext to the decryption oracle. The strongest achievable notion for randomizable schemes is RCCA, where whenever 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 for a polynomial-size message space $\{0,1\}^k$.

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

$\overline{\mathsf{Encrypt}}(pk, m)$: Run $(vk, sk) \xleftarrow{\$} \mathsf{SKeyGen}(pp)$; $c \xleftarrow{\$} \mathsf{Encrypt}^+(pk, vk, m)$; $\sigma \xleftarrow{\$} \mathsf{Sign}^+(sk, pk, c)$; return $\overline{c} = (c, \sigma, vk)$.

$\overline{\mathsf{Decrypt}}(dk, (c, \sigma, vk))$: If $\mathsf{Verify}^+(vk, pk, c, \sigma) = 1$, return $m = \mathsf{Decrypt}^+(dk, vk, c)$; else return $\perp$.

$\overline{\mathsf{Random}}$ is defined as Random^+.

Theorem 2. *The above encryption scheme for polynomial-size message spaces is RCCA-secure under the SXDH and the CDH$^+$ assumption.*

Proof Sketch. We will give a proof sketch and refer to the full version for a detailed proof. Intuitively, ciphertexts hide the message, since under SXDH we could replace the commitments and proofs in the challenge ciphertext by simulated ones and under DDH, we could replace $c_2 = \mathcal{F}(m)P^r$ by a random element, so the ciphertext would contain no more information about the message. The difficulty is that we need to simulate the decryption oracle. For this we program the hash function H: let vk^* be the key contained in the challenge ciphertext; we choose $a, b \xleftarrow{\$} \mathbb{Z}_p$ and set $h_1 = P^{-a\cdot H'(vk^*)} \cdot g_1^b$ and $h_2 = P^a$, which is distributed correctly and set $H(vk) = P^{a(H'(vk)-H'(vk^*))} \cdot g_1^b$. For a well-formed ciphertext containing $vk_i \neq vk^*$, we then have $c_2 \cdot (c_3 \cdot c_1^{-b})^{-1/(a(H'(vk)-H'(vk^*)))} = c_2 \cdot P^{-r} = c_2 \cdot c_1^{-d} = \mathsf{Decrypt}(d, (c_1, c_2))$, meaning we can use c_3 to decrypt without knowing d; for the challenge ciphertext under vk^* we have $c_3 = g_1^{br}$, so we can embed a DDH challenge.

The reduction can thus answer decryption queries containing some $vk \neq vk^*$, but not if it contains vk^*. However, if an adversary submits a valid ciphertext with vk^* which does not encrypt the challenge message, then it would break SRC unforgeability, so security of our SRC scheme implies that the adversary cannot make this type of query. $\square$

$\mathsf{EKeyGen}(1^\lambda, 1^k)$**:**

$pp := (\mathcal{G}, z, \boldsymbol{u}) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, 1^k)$; $crs \xleftarrow{\$} \mathsf{Setup}_{\mathrm{GS}}(\mathcal{G})$; $h \xleftarrow{\$} \mathbb{G}_1^2$

$(P, dk) \xleftarrow{\$} \mathsf{KeyGen}(\mathcal{G}, 1)$; return $pk := (pp, crs, h, P)$, dk.

$\mathsf{Encrypt}^+((pp, crs, h, P), vk = (pp, X_1, X_2), m; r)$**:**

Compute, with H defined by h as in (1):

$$c_1 = g_1^r \qquad c_2 = \mathcal{F}(m) \cdot P^r \qquad c_3 = H(vk)^r$$

Make commitments $\boldsymbol{C}$: For $i = 1, \ldots, k$:

$$C_{1,m,i} = \mathcal{C}_1'(m_i) \qquad C_{2,m,i} = \mathcal{C}_2'(m_i)$$
$$C_T = \mathcal{C}_1(X_1^r) \qquad C_r = \mathcal{C}_2'(r)$$

Compute GS proofs $\boldsymbol{\pi}$ for the following (with $\overline{r}$ being the value committed in C_r; $\overline{m}_i$ in $C_{2,m,i}$; and $\overline{w}$ in C_T):

- π_r proves $g_1^{\overline{r}} = c_1$.
- π_m consists of:
  π_i' proving $\overline{m}_i$ is a bit for all i;
  π_m' proving $c_2 = u_0 \prod_{i=1}^k u_i^{\overline{m_i}} \cdot P^{\overline{r}}$.
- π_T proves $X_1^{\overline{r}} = \overline{w}$.
- π_V proves $H(vk)^{\overline{r}} = c_3$.

Return $c := (c_1, c_2, c_3, \boldsymbol{C}, \boldsymbol{\pi})$.

$\mathsf{Sign}^+(sk = (pp, Y), (pp, crs, h, P), c; s)$**:**

If $\boldsymbol{\pi}$ is not valid for $\boldsymbol{C}, vk, P$, return $\perp$. Else return

$$\sigma_1 = c_1^s \qquad \sigma_2 = Y \cdot c_2^s \qquad (2)$$
$$\sigma_3 = g_1^s \qquad \sigma_4 = g_2^s \qquad \sigma_5 = P^s$$

$\mathsf{Random}^+(vk, (pp, crs, h, P), c, \sigma; (r', s'))$**:**

Let $c = (c_1, c_2, c_3, \boldsymbol{C}, \boldsymbol{\pi})$; set:

$$c_1' := c_1 \cdot g_1^{r'} \qquad c_2' := c_2 \cdot P^{r'}$$
$$c_3' := c_3 \cdot H(vk)^{r'}$$
$$\sigma_1' := \sigma_1 \cdot c_1^{s'} \cdot \sigma_3^{r'} \cdot g_1^{r' \cdot s'}$$
$$\sigma_2' := \sigma_2 \cdot c_2^{s'} \cdot \sigma_5^{r'} \cdot P^{r' \cdot s'}$$
$$\sigma_3' := \sigma_3 \cdot g_1^{s'} \qquad \sigma_4' := \sigma_4 \cdot g_2^{s'}$$
$$\sigma_5' := \sigma_5 \cdot P^{s'}$$

Set $C_r' = C_r \cdot \mathcal{C}_2(r')$, adapt π_r, π_T, π_V accordingly.
Randomize all commitments and proofs to $\boldsymbol{C}'$ and $\boldsymbol{\pi}'$.
Return $(c_1', c_2', c_3', \boldsymbol{C}', \boldsymbol{\pi}')$ and σ'.

$\mathsf{Verify}^+((pp, X_1, X_2), (pp, crs, h, P), (c_1, c_2, c_3, \boldsymbol{C}, \boldsymbol{\pi}), \sigma)$**:**

Return 1 iff $\boldsymbol{\pi}$ verifies and the following hold:

$$e(\sigma_1, g_2) = e(c_1, \sigma_4) \qquad (3a)$$
$$e(\sigma_2, g_2) = e(z, X_2)\, e(c_2, \sigma_4) \qquad (3b)$$
$$e(\sigma_3, g_2) = e(g_1, \sigma_4) \qquad e(\sigma_5, g_2) = e(P, \sigma_4) \qquad (3c)$$

$\mathsf{Decrypt}^+(dk, (pp, crs, h, P), vk, c)$**:**

Let $c = (c_1, c_2, c_3, \boldsymbol{C}, \boldsymbol{\pi})$. If $\boldsymbol{\pi}$ is not valid, return $\perp$;
else let $F = \mathsf{Decrypt}(dk, c = (c_1, c_2))$;
browse $\mathcal{M}$ and return the first m with $\mathcal{F}(m) = F$.

Figure 2: Our SRC scheme

4. BELENIOSRF

In this section we define Belenios Receipt-Free (BeleniosRF), a strongly receipt-free voting protocol that builds on [12, 22].

4.1 Overview

The election public/secret key pair $(\mathbf{pk}, \mathbf{sk})$ is an encryption/extraction key pair generated via $\mathsf{EKeyGen}^{(x)}$ (cf. Remark 2), and user key pairs (upk, usk) are signature keys generated by SKeyGen. A user casts a vote by encrypting it via $\mathsf{Encrypt}^+$ under $\mathbf{pk}$ w.r.t. his upk, and uses usk to then sign the ciphertext via Sign^+ (together, this corresponds to a ciphertext of our RCCA encryption scheme).

When the ballot box receives a valid ballot, 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 on their now-randomized ciphertexts.

Tallying follows standard techniques of e-voting: our construction allows for homomorphic tallying as well as shuffling. In the first case, we take advantage of the special structure of GS commitments, which allow us to calculate a partial tally for each option by adding the corresponding commitment across voters, and then decrypting the resulting commitment (with proof of correctness).

Using shuffling, the encrypted votes are re-randomized and shuffled (and a proof of correct execution of this is generated) via an algorithm Shuffle. Then 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.

We now describe the homomorphic tallying version, where $\mathbb{V} = \{0,1\}^k$, and the result function is simple vector addition.

The scheme $\mathcal{V}^{\mathsf{BeleniosRF}}$ is based on the SCR scheme from Section 3.3 and consists of the following algorithms:

$\mathsf{Setup}(1^\lambda, 1^k)$: Compute $(\mathbf{pk}, \mathbf{sk}) \xleftarrow{\$} \mathsf{EKeyGen}^{(x)}(1^\lambda, 1^k)$, produce a Fiat-Shamir random oracle proof Π_σ that crs, contained in $\mathbf{pk}$, is binding. Return $(\mathbf{pk}^* = (\mathbf{pk}, \Pi_\sigma), \mathbf{sk})$.

$\mathsf{Register}(id)$: On (implicit) input $\mathbf{pk} = (pp, crs, h, P)$, return $(upk_{id}, usk_{id}) \xleftarrow{\$} \mathsf{SKeyGen}(pp)$.

$\mathsf{Vote}\,(id, upk, usk, v)$ is used by a voter to create a ballot b for vote $v \in \mathbb{V}$. It computes $c \leftarrow \mathsf{Encrypt}^+(\mathbf{pk}, upk, v)$ and $\sigma \leftarrow \mathsf{Sign}^+(usk, \mathbf{pk}, c)$; and returns $b = (id, upk, c, \sigma)$.

$\mathsf{Valid}(\mathsf{BB}, b)$ 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, upk, c, σ) and checks if

- id corresponds to an eligible voter from ID and upk corresponds to the registration of user id;
- $\mathsf{Verify}^+(upk, \mathbf{pk}, c, \sigma) = 1$.

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

$\mathsf{Append}(\mathsf{BB}, b = (id, upk, c, \sigma))$ randomizes (c, σ) as $(c', \sigma') \leftarrow \mathsf{Random}^+(upk, \mathbf{pk}, c, \sigma)$ and appends to BB a randomized version $b' = (id, upk, c', \sigma')$ of b.

$\mathsf{Publish}(\mathsf{BB})$ takes every entry $b = (id, upk, c, \sigma)$ in BB and removes elements $id, c_3, C_T, \pi_T, \pi_V$ and σ_5, constructing $\hat{b} := \big(upk, (c_1, c_2, C_m, C_r, \pi_r, \pi_m), (\sigma_1, \sigma_2, \sigma_3, \sigma_4)\big)$. It then adds $\hat{b}$ to PBB,[2] and returns PBB.

[2] As noted in Remark 2, these are precisely the elements that guarantee unforgeability, which assures a voter that the plaintext of his encrypted vote was not altered.

$\mathsf{VerifyVote}(\mathsf{PBB}, id, upk, usk, b)$ browses PBB for an entry $\hat{b}$ containing upk. If none exists, it returns $\perp$. For entry $\hat{b} := \big(upk{=}(pp, X_1, X_2), (c_1, c_2, C_m, C_r, \pi_r, \pi_m), (\sigma_1, \sigma_2, \sigma_3, \sigma_4)\big)$ if π_r and π_m are valid and

$$e(\sigma_1, g_2) = e(c_1, \sigma_4) \quad e(\sigma_2, g_2) = e(z, X_2) \cdot e(c_2, \sigma_4)$$
$$e(\sigma_3, g_2) = e(g_1, \sigma_4)$$

then return $\top$, else return $\perp$.

$\mathsf{Tally}(\mathsf{BB}, \mathbf{sk})$ consists of the following steps. Let N be the number of ballots.

- Parse each ballot $b \in \mathsf{BB}$ as $b = (id^{(b)}, upk^{(b)}, c^{(b)}, \sigma^{(b)})$.
- If there is any ballot b that does not pass $\mathsf{Valid}(\mathsf{BB}, b)$, output $(r = \perp, \mathsf{PBB}, \Pi_d = \emptyset)$.
- Let $\{C^{(b)}_{1,m,i}\}_{i=1}^{k}$ be the commitments in $C^{(b)}_m$ contained in $c^{(b)}$. Compute $T_i = \sum_{b \in \mathsf{BB}} C^{(b)}_{1,m,i}$. The tally t_i for candidate i is produced by decrypting T_i with the GS extraction key $\xi = \mathbf{sk}$.
- Produce the result $r = (t_1, \ldots, t_k)$ and Π_d, a Fiat-Shamir proof of correct extraction.
- Output (r, PBB, Π_d).

$\mathsf{Verify}(\mathsf{PBB}, r, \Pi_\sigma, \Pi_d)$ verifies Π_σ w.r.t. crs and Π_d w.r.t. PBB and the result r.

4.2 Receipt-Freeness

We now show that BeleniosRF satisfies strong receipt-freeness, as defined in Definition 1. Note that this in particular implies vote privacy of BeleniosRF.

THEOREM 3. *$\mathcal{V}^{\mathsf{BeleniosRF}}$ is strongly receipt-free under the SXDH assumption in the random-oracle model.*

PROOF. The proof uses the ideas of that of Theorem 2. The main one is again to use hash-function programmability and to decrypt a ciphertext (c_1, c_2, c_3) using components c_2 and c_3 instead of the GS commitments. This will allow us to switch to a hiding CRS, for which the commitments would not be extractable. By randomizability of our SCR scheme and of Groth-Sahai proofs, instead of re-randomizing the ballots in PBB, we can simply recompute them. Finally, having switched to a hiding CRS and a simulated ROM proof thereof, we are able to replace the adversary's view with uniformly distributed values, irrespective of β.

We proceed by a sequence of hybrid games, which we show are indistinguishable:

Hybrid ($\boldsymbol{\beta}$,0) is the sRF game $\mathsf{Exp}^{\mathsf{srf},\beta}_{\mathcal{A},\mathcal{V}}$ (Definition 1 and Figure 1).

Hybrid ($\boldsymbol{\beta}$,1) is the same game as Hybrid (β,0) for $\beta = 1$; for $\beta = 0$ the difference is that the Fiat-Shamir proofs for the CRS and the tally are simulated.

Hybrid (β,0) $\rightarrow$ Hybrid (β,1): Since ROM proofs can be perfectly simulated by using random-oracle programmability, the two hybrid games are distributed equivalently.

Hybrid ($\boldsymbol{\beta}$,2) is defined as Hybrid (β,1), except for how h is chosen. For $a, b \xleftarrow{\$} \mathbb{Z}_p$ we define $h_1 = g_1^b$ and $h_2 = P^a$ (as in Theorem 2 but setting $H'(vk^*) := 0$).

Hybrid (β,1) $\rightarrow$ Hybrid (β,2): It is immediate that both games are distributed equivalently.

Hybrid ($\boldsymbol{\beta}$,3) is defined as Hybrid (β,2), but the result is computed differently: each ballot $b_i = (id_i, upk_i, c_i, \sigma_i)$ is decrypted as $F_i = c_{i,2} \cdot (c_{i,3} \cdot c_{i,1}^{-b})^{-1/(a \cdot upk_i)}$ and vote v_i is defined as the smallest $v_i \in \{0,1\}^k$ satisfying $\mathcal{F}(v_i) = F_i$. The result is $r = (t_1, \ldots, t_k)$ with $t_j = \sum_i v_{i,j}$.

Hybrid (β,2) $\rightarrow$ Hybrid (β,3): Perfect soundness of the GS proofs contained in c_i guarantees that this alternative way of decryption leads to the same result as extracting the bits of v_i from the commitments (we ignore collisions in $\mathcal{F}$ which only occur with negligible probability).

Hybrid ($\boldsymbol{\beta}$,4) is defined as Hybrid (β,3), except that PBB is computed differently: for ballot b_i, after extracting v_i, instead of re-randomizing b_i, we freshly compute $\hat{b}_i$ for user i with $usk_i = (pp, Y_i)$ as follows: we pick $r_i, s_i \xleftarrow{\$} \mathbb{Z}_p$ to set

$$c_{i,1} = g_1^{r_i} \qquad c_{i,2} = \mathcal{F}(v_i) \cdot P^{r_i} \qquad \sigma_{i,1} = c_1^{s_i}$$
$$\sigma_{i,2} = Y_i \cdot c_2^{s_i} \qquad \sigma_{i,3} = g_1^{s_i} \qquad \sigma_{i,4} = g_2^{s_i}$$

and using witnesses r_i and v_i, we compute $C_{i,m}, C_{i,r}$ and $\pi_{i,r}, \pi_{i,m}$. We set $\hat{b}_i = \big(upk_i, (c_{i,1}, c_{i,2}, C_{i,m}, C_{i,r}, \pi_{i,r}, \pi_{i,m}), (\sigma_{i,1}, \sigma_{i,2}, \sigma_{i,3}, \sigma_{i,4})\big)$.

Hybrid (β,3) $\rightarrow$ Hybrid (β,4): By re-randomizability of our SCR scheme and GS proofs, re-randomized ciphertexts, signatures and proofs are distributed exactly as freshly computed ones. The two hybrids are thus equally distributed.

Hybrid ($\boldsymbol{\beta}$,5) is defined as Hybrid (β,4), except that the CRS contained in $\mathbf{pk}$ is set up in hiding mode, i.e., computed via $\mathsf{Setup}^{(h)}_{\mathrm{GS}}$.

Hybrid (β,4) $\rightarrow$ Hybrid (β,5): By the properties of GS proofs, the two hybrids are indistinguishable under the SXDH assumption.

Hybrid ($\boldsymbol{\beta}$,6) is defined as Hybrid (β,5), except that the commitments and proofs published in PBB are simulated.

Hybrid (β,5) $\rightarrow$ Hybrid (β,6): By the properties of GS proofs, under a hiding CRS regularly computed proofs and simulated proofs are distributed equivalently; the two hybrids are thus equally distributed.

Hybrid ($\boldsymbol{\beta}$,7) is defined as Hybrid (β,6), except that for every i, when computing PBB entry $\hat{b}_i$, $c_{i,2}$ is computed as $c_{i,2} = \mathcal{F}(v_i) \cdot g_1^{w_i}$ for $w_i \xleftarrow{\$} \mathbb{Z}_p$.

Hybrid (β,6) $\rightarrow$ Hybrid (β,7): The two hybrids are indistinguishable under the DDH assumption in $\mathbb{G}_1$, which is proved as follows: we first note that in Hybrid (β,6), d (the decryption key with $P = g_1^d$) is not used anywhere, and r_i is only used to compute $c_{i,1}$ and $c_{i,2}$ (since the GS commitments and proofs are simulated).

We give a reduction from DDH to distinguishing Hybrids 6 and 7. Let $(P = g_1^d, R = g_1^r, W)$ be a DDH instance, where either W is random or $W = g_1^{d \cdot r}$. By random self-reducibility of DDH [6] we can create arbitrarily many instances (P, R_i, W_i), where $R_i = g_1^{r_i}$ for some uniformly random r_i, and W_i is independently random if W was, or $W_i = g_1^{d \cdot r_i}$ if $W = g_1^{d \cdot r}$.

The simulator now sets $\mathbf{pk} = (pp, crs, P)$, with P from the instance, and $c_{i,1} = R_i$ and $c_{i,2} = \mathcal{F}(v_i) \cdot W_i$. If $W_i = P^{r_i}$ then this is distributed as in Hybrid (β,6), whereas if W_i is random, this is distributed as in Hybrid (β,7).

Observe that Hybrid (0,7) and Hybrid (1,7) are equally distributed, since in both games every ciphertext $(c_{i,1}, c_{i,2})$ is a uniformly random pair. We have thus constructed a sequence of hybrid games Hybrid (0,0), ..., Hybrid (0,7), Hybrid (1,7), ..., Hybrid (1,0) which are indistinguishable under SXDH and of which the first one corresponds to the sRF game with $\beta = 0$ and the last is the sRF game with $\beta = 1$. This concludes the proof of strong receipt-freeness. $\square$

REMARK 3. *We note that our scheme can be easily modified and proven secure in the standard model if we assume a trusted CRS: drop Π_σ in* Setup *and use GS proofs fro Π_d.*

4.3 Verifiability

We consider strong verifiability from [22], which intuitively ensures that the result of the election reflects the votes of:

- All voters who properly checked that their ballot appears in the bulletin board at the end of the election. In BeleniosRF, a voter should check that one ballot in PBB is signed with her credential.
- A subset of the voters who did not perform that final check. A voters may stop after casting her vote, thus there is no guarantee that her ballot made it into the ballot box. However, if the ballot is present, it should not be possible to *modify* the corresponding vote.
- At most all corrupted voters. In particular, an adversary should not be able to add more votes than the number of voters he controls.

We refer the reader to [22] for the formal definition and point out that strong verifiability assumes that voting devices are honest. We first note that BeleniosRF cannot be strongly verifiable if revoting is allowed. Indeed, if a voter first casts a ballot b_1 for a candidate v_1, but later changes her mind and votes for v_2, casting a new ballot b_2, a malicious voting server may force the voter to keep the initial vote v_1 by re-randomizing b_1 instead of b_2, and the voter would not be able to detect it. Therefore, in what follows, we assume that a no-revote policy is applied. We believe that no-revoting is not a real restriction since, as discussed in the introduction, this actually corresponds to the most common setting used in practice. By slightly generalizing the strong-verifiability transformation in [22, Section 4], we are able to show:

THEOREM 4. *BeleniosRF is strongly verifiable if the underlying signature on randomizable ciphertexts scheme is unforgeable.*

The transformation to strong verifiability in [22] consists in the voter signing with her private signing key usk a ballot b obtained via an existing voting protocol that is *weakly verifiable* (roughly speaking, weak verifiability assumes that the voting server is honest, e.g., it does not modify nor erase ballots). Next, the voter sends the triple (upk, b, σ) to the voting server. The latter, after validating the ballot b and verifying its signature σ, adds the triple (upk, b, σ) to the ballot box. At the end of the election, the voter checks that her ballot (upk, b, σ) appears in PBB by a simple search.

We generalize this transformation by allowing the voting server to add a *transformed* triple (upk, b', σ') to the ballot box on input the voter's ballot (upk, b, σ), such that potentially $b \neq b'$ and $\sigma \neq \sigma'$ (in the original construction, one simply sets $b' = b$ and $\sigma' = \sigma$). In our generalized transformation, the voter on input her cast ballot (upk, b, σ) checks whether there exists an entry (upk, b', σ') in PBB such that (b', σ') verifies under her key upk. Due to unforgeability of randomizable signatures on ciphertexts (cf. Section 3.3) and because of the no-revoting policy, this check guarantees that the new ballot b' displayed in the bulletin board contains the same vote as the original ballot b cast by the voter.

Strong verifiability assumes that either the ballot box (i.e. the re-randomization server) or the registrar is honest. As pointed out in Section 1.2, the security of the generalized transformation described in the previous paragraph is jeopardized if this trust assumption is violated, as the existence of an entry (upk, b', σ') in PBB would no longer guarantee that b' contains the choice cast by the voter. In fact, an attacker controlling both the registrar and the voting server can insert entries (upk, b', σ') in PBB that pass all tests but modified the voter's choice. This is due to the fact that the registrar knows each voter's private signing key. An obvious countermeasure is to let each voter generate their own signing key pair and simply ask the registrar to include the corresponding verification key in the list of eligible keys for the election.

Alternatively, one can thresholdize the role of the registrar (who simply sends a private signing key to each voter) so it becomes less likely for the attacker to obtain a voter's private key.

5. EFFICIENCY OF BELENIOSRF

The ballot encryption scheme we introduced is somewhat involved, especially since we use bit-by-bit Groth-Sahai proofs. For this reason, we benchmarked ballot creation on a number of potential client devices. We built a JavaScript implementation [2] of the voting process (encrypt, sign, prove) using the CertiVox IoT Crypto Library [17]. We used a BN curve on a 254-bit prime field. We considered the values $k = 1, 5, 10$ and 25. For homomorphic tallying, as used in Section 4, k represents the number of candidates in an election. If we switch to shuffle-based tallying, k is the length of the message, which means we can support up to 2^k candidates.

As seen in Table 1, recent devices can complete the required cryptographic operations in reasonable time for small values of k. We see that while the linear cost associated with the message size is the dominant factor, the constant factor is not negligible for low-end devices. While slower than the current Helios or Belenios implementation (which do not use elliptic curves), performance is acceptable, especially for modern devices. Moreover, our implementation is single-threaded with only rudimentary optimizations. By constructing proofs incrementally as the ballot is filled, we could amortize the linear part of the cost. Alternatively, we may increase performance by coding a native client, e.g. a smartphone app.

We expect that server performance for BeleniosRF will be less of a bottleneck. Compared to Helios, the main additional

Device	$k=1$	$k=5$	$k=10$	$k=25$
2013 Laptop –i7-4650U	1.00s	2.43s	4.02s	9.24s
2010 Desktop –i3-530	1.49s	3.46s	5.92s	13.62s
2014 Tablet –Exynos 5420	6.97s	12.91s	21.92s	47.26s
2016 Phone –SD 810	2.75s	6.26s	10.39s	22.19s
2014 Phone –SD 801	5.55s	13.12s	22.70s	48.06s
2012 Phone –A6	9.04s	18.65s	29.96s	63.77s

Table 1: Time to encrypt, sign and perform GS proofs for ballots with a k-bit payload. This allows for up to k candidates with homomorphic tallying, or 2^k using shuffles.

cost is verifying our Groth-Sahai proofs, which is dominated by ca. $64(k+1)$ pairings, reducible to $4(k+35)$ using techniques from [11] and [31]. Given the timings provided by Beuchat *et al.* [10], we expect a throughput of roughly 5 ballots/second/core for $k = 10$. Additionally, checks can be amortized throughout the voting period, as ballots come in.

6. THE BLAZY ET AL. VOTING PROTOCOL IS NOT BALLOT-PRIVATE

Blazy *et al.* [12], who introduced the notion of signatures on randomizable ciphertexts, proposed 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

$$b = \left(vk,\ c = \{v\}_{pk}^{r},\ \sigma_c^{vk,s}, \pi_{pk}^{t, v\,\in\,\{0,1\}}\right)\ ,$$

where $r, s, t \in \mathbb{Z}_p$ denote the 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: $\left(vk,\ c = \{v\}_{pk}^{r'},\ \sigma_c^{vk,s'},\ \pi_{pk}^{t', v\,\in\,\{0,1\}}\right)$, where $r', s', t' \xleftarrow{\$} \mathbb{Z}_p$.

Similarly to BeleniosRF, 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.

An attack on ballot privacy. However, the above ballot casting workflow is not ballot private, let alone receipt-free. The following is a ballot replay attack, which is known to break ballot privacy [23]:

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

These instructions allow any voter with knowledge of a ballot b to produce an independent-looking ballot $\bar{b}$, that will be accepted by the voting server and effectively contains a copy of the vote in b. Thus, the voting protocol [12] is not ballot-private. (Note that due to re-randomizing being allowed, the ballot box cannot discard copied votes, as they look like legitimate ones.)

7. CONCLUSIONS

We introduced the notion of *strong receipt-freeness*, where a malicious voter (i.e. vote-selling or coerced) cannot produce a receipt proving how she voted, whether the voter decided to act maliciously before, during or after casting the ballot.

Our adversarial model is close to the spirit of the seminal work on receipt-freeness by Benaloh and Tuinstra [8]. Moreover, our definition builds on the recent work [9], inheriting a simple, concise, and game-based definition. Such 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 SXDH assumption in pairing groups, in the random oracle model. The protocol is built using ideas from a previous work [12] that claimed to have solved this problem. We show however that the previous voting scheme was not ballot-private, which is weaker than receipt-freeness.

To the best of our knowledge, this is the first scheme that is both receipt-free (in a strong sense) and has universal verifiability (in the sense of strong verifiability [22]), without requiring the existence of an untappable channel, or the use of secure hardware tokens. We only require that the receipt-freeness adversary is not eavesdropping the communication between the voter and the voting server, and the existence of a re-randomization service. As a result, we overcome the impossibility result [19], stating no scheme can be receipt-free and universally verifiable without an untappable channel. We achieve this by relying on a ballot box server that is entrusted to re-randomize ballots without changing its contents, in a publicly verifiable way [12]. Finally, we showed the feasibility of our approach by implementing a voting client in Javascript and measuring its performance in a number of platforms.

Acknowledgements. The authors are grateful to the anonymous reviewers for their comments and suggestions that have helped improve this work. This work has received funding from the European Research Council (ERC) under the EU's Horizon 2020 research and innovation program (grant agreement No 645865-SPOOC), the ERC FP7 programme (grant No 307937), EPSRC grant EP/G037264/1 and COST Action IC1306.

8. REFERENCES

[1] International Association for Cryptologic Research, elections. Page at http://www.iacr.org/elections/.

[2] BeleniosRF – Javascript Implementation of the Voting Client Core. https://dl.dropboxusercontent.com/u/16959859/bel-100/JS/index.html, 2016.

[3] B. Adida. Helios: Web-based Open-Audit Voting. In *USENIX 2008*, 2008. http://heliosvoting.org.

[4] B. Adida, O. de Marneffe, O. Pereira, and J.-J. Quisquater. Electing a university president using open-audit voting: Analysis of real-world use of Helios. In *EVT/WOTE 2009*, 2009.

[5] J. P. Allepuz and S. G. Castelló. Internet voting system with cast as intended verification. In *VoteID 2011*. Springer, 2011.

[6] M. Bellare, A. Boldyreva, and S. Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In *EUROCRYPT 2000*. Springer, 2000.

[7] J. Benaloh. Ballot casting assurance via voter-initiated poll station auditing. In *EVT/WOTE 2007*, 2007.

[8] J. C. Benaloh and D. Tuinstra. Receipt-free secret-ballot elections (extended abstract). In *ACM STOC 94*. ACM Press, 1994.

[9] D. Bernhard, V. Cortier, D. Galindo, O. Pereira, and B. Warinschi. A comprehensive analysis of game-based ballot privacy definitions. In *IEEE Security and Privacy 2015*. IEEE Computer Society, 2015.

[10] J.-L. Beuchat, J. E. González-Díaz, S. Mitsunari, E. Okamoto, F. Rodríguez-Henríquez, and T. Teruya. High-speed software implementation of the optimal ate pairing over barreto–naehrig curves. In *Pairing-Based Cryptography 2010*. Springer, 2010.

[11] O. Blazy, G. Fuchsbauer, M. Izabachene, A. Jambert, H. Sibert, and D. Vergnaud. Batch groth–sahai. In *ACNS 2010*. Springer, 2010.

[12] O. Blazy, G. Fuchsbauer, D. Pointcheval, and D. Vergnaud. Signatures on randomizable ciphertexts. In *PKC 2011*. Springer, 2011.

[13] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. *J. of Cryptology*, 17(4), 2004.

[14] K. Bräunlich and R. Grimm. Formalization of receipt-freeness in the context of electronic voting. In *Availability, Reliability and Security 2011*. IEEE Computer Society, 2011.

[15] R. Canetti and R. Gennaro. Incoercible multiparty computation (extended abstract). In *FOCS '96*. IEEE Computer Society, 1996.

[16] R. Canetti, H. Krawczyk, and J. B. Nielsen. Relaxing chosen-ciphertext security. In *CRYPTO 2003*. Springer, 2003.

[17] CertiVox. A Cryptographic Library for the Internet of Things. https://github.com/CertiVox/MiotCL, 2015.

[18] D. Chaum, A. Essex, R. Carback, J. Clark, S. Popoveniuc, A. Sherman, and P. Vora. Scantegrity: end-to-end voter-verifiable optical-scan voting. *IEEE Security and Privacy*, 6(3), 2008.

[19] B. Chevallier-Mames, P. Fouque, D. Pointcheval, J. Stern, and J. Traoré. On some incompatible properties of voting schemes. In *EVT/WOTE*, 2010.

[20] S. S. M. Chow, J. K. Liu, and D. S. Wong. Robust receipt-free election system with ballot secrecy and verifiability. In *Network and Distributed System Security Symposium 2008*. The Internet Society, 2008.

[21] M. R. Clarkson, S. Chong, and A. C. Myers. Civitas: Toward a secure voting system. In *IEEE Security and Privacy 2008*. IEEE Computer Society, 2008.

[22] V. Cortier, D. Galindo, S. Glondu, and M. Izabachène. Election verifiability for Helios under weaker trust assumptions. In *ESORICS 2014*. Springer, 2014.

[23] V. Cortier and B. Smyth. Attacking and fixing Helios: An analysis of ballot secrecy. In *CSF 2011*. IEEE, 2011.

[24] V. Cortier and B. Smyth. Attacking and fixing Helios: An analysis of ballot secrecy. *Journal of Computer Security*, 21(1), 2013.

[25] R. Cramer, R. Gennaro, and B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *EUROCRYPT'97*. Springer, 1997.

[26] S. Delaune, S. Kremer, and M. Ryan. Coercion-resistance and receipt-freeness in electronic voting. In *CSFW 2006*. IEEE Computer Society, 2006.

[27] G. Fuchsbauer and D. Pointcheval. Proofs on encrypted values in bilinear groups and an application to anonymity of signatures. In *Pairing-Based Cryptography–Pairing 2009*. Springer, 2009.

[28] R. W. Gardner, S. Garera, and A. D. Rubin. Coercion resistant end-to-end voting. In *FC 2009*. Springer, 2009.

[29] K. Gjøsteen. The Norwegian internet voting protocol. *IACR Cryptology ePrint Archive*, 2013, 2013.

[30] S. Glondu, V. Cortier, and P. Gaudry. Belenios – Verifiable online voting system. http://belenios.gforge.inria.fr, 2015.

[31] A. González, A. Hevia, and C. Ràfols. Qa-nizk arguments in asymmetric groups: new tools and new constructions. In *ASIACRYPT 2015*. Springer, 2015.

[32] J. Groth and A. Sahai. Efficient non-interactive proof systems for bilinear groups. In *EUROCRYPT 2008*. Springer, 2008.

[33] J. Heather and S. Schneider. A formal framework for modelling coercion resistance and receipt freeness. In *Formal Methods 2012*. Springer, 2012.

[34] M. Hirt. Receipt-free K-out-of-L voting based on ElGamal encryption. In *EVT/WOTE 2010*. Springer, 2010.

[35] M. Hirt and K. Sako. Efficient receipt-free voting based on homomorphic encryption. In *EUROCRYPT 2000*. Springer, 2000.

[36] H. L. Jonker and E. P. de Vink. Formalising receipt-freeness. In *Information Security 2006*. Springer, 2006.

[37] A. Juels, D. Catalano, and M. Jakobsson. Coercion-resistant electronic elections. In *Workshop on Privacy in the Electronic Society 2005*. ACM, 2005.

[38] A. Kiayias, T. Zacharias, and B. Zhang. End-to-end verifiable elections in the standard model. In *EUROCRYPT 2015*. Springer, 2015.

[39] O. Kulyk, V. Teague, and M. Volkamer. Extending Helios towards private eligibility verifiability. In *E-Voting and Identity 2015*. Springer, 2015.

[40] R. Küsters and T. Truderung. An epistemic approach to coercion-resistance for electronic voting protocols. In *S&P 2009*. IEEE Computer Society, 2009.

[41] E. Magkos, M. Burmester, and V. Chrissikopoulos. Receipt-freeness in large-scale elections without untappable channels. In *E-Commerce, E-Business, E-Government 2001*. Kluwer, 2001.

[42] T. Moran and M. Naor. Receipt-free universally-verifiable voting with everlasting privacy. In *CRYPTO 2006*. Springer, 2006.

[43] T. Okamoto. Receipt-free electronic voting schemes for large scale elections. In *Security Protocols 97*. Springer, 1997.

[44] P. 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, 2009.

[45] P. Y. A. Ryan, P. B. Roenne, and V. Iovino. Selene: Voting with transparent verifiability and coercion-mitigation. Cryptology ePrint Archive, Report 2015/1105, 2015. http://eprint.iacr.org/.

[46] D. Springall, T. Finkenauer, Z. Durumeric, J. Kitcat, H. Hursti, M. MacAlpine, and J. A. Halderman. Security analysis of the estonian internet voting system. In *ACM CCS 2014*. ACM Press, 2014.

[47] B. R. Waters. Efficient identity-based encryption without random oracles. In *EUROCRYPT 2005*. Springer, 2005.